

Table of Contents

Part I What's New	1
Part II General Information	5
1 Overview	6
2 Features	9
3 Requirements	12
4 Compatibility	12
5 Using Several DAC Products in One IDE	16
6 Component List	16
7 Hierarchy Chart	18
8 Editions	20
9 Licensing	22
10 Getting Support	25
11 FAQ	26
Part III Getting Started	33
1 Installation	38
2 Logging on to PostgreSQL	41
3 Creating Database Objects	46
4 Deleting Data From Tables	50
5 Inserting Data Into Tables	54
6 Retrieving Data	58
7 Modifying Data	61
8 Demo Projects	64
9 Deployment	69
Part IV Using PgDAC	70
1 Updating Data with PgDAC Dataset Components	71
2 Master/Detail Relationships	72
3 Automatic Key Field Value Generation	73
4 Data Type Mapping	74
5 Data Encryption	80
6 Working in an Unstable Network	82
7 Secure Connections	83
8 Disconnected Mode	85
9 Batch Operations	86
10 Increasing Performance	90

11	Macros	92
12	DataSet Manager	92
13	TPgLoader Component	97
14	Large Objects	98
15	REFCURSOR Data Type	100
16	National and Unicode Characters	101
17	Connection Pooling	102
18	DBMonitor	103
19	Writing GUI Applications with PgDAC	104
20	Compatibility with Previous Versions	104
21	64-bit Development with Embarcadero RAD Studio XE2	105
22	Database Specific Aspects of 64-bit Development	110

Part V Reference

111

1	CRAccess	112
	Classes	113
	TCRCursor Class.....	113
	Members	114
	Types	114
	TBeforeFetchProc Procedure Reference.....	114
	Enumerations	115
	TCRIsolationLevel Enumeration.....	115
	TCRTransactionAction Enumeration.....	116
	TCursorState Enumeration.....	116
2	CRBatchMove	117
	Classes	117
	TCRBatchMove Class.....	118
	Members	118
	Properties	120
	AbortOnKeyViol Property	121
	AbortOnProblem Property.....	122
	ChangedCount Property.....	122
	CommitCount Property.....	122
	Destination Property.....	123
	FieldMappingMode Property.....	123
	KeyViolCount Property.....	123
	Mappings Property.....	124
	Mode Property.....	124
	MovedCount Property	125
	ProblemCount Property	125
	RecordCount Property	126
	Source Property.....	126
	Methods	127
	Execute Method.....	127
	Events	127
	OnBatchMoveProgress Event.....	128
	Types	128
	TCRBatchMoveProgressEvent Procedure Reference.....	128
	Enumerations	129

TCRBatchMode Enumeration.....	129
TCRFieldMappingMode Enumeration.....	130
3 CREncryption	130
Classes	131
TCREncryptor Class.....	131
Members	131
Properties	132
DataHeader Property.....	133
EncryptionAlgorithm Property.....	133
HashAlgorithm Property.....	133
InvalidHashAction Property.....	134
Passw ord Property.....	134
Methods	135
SetKey Method.....	135
Enumerations	136
TCREncDataHeader Enumeration.....	136
TCREncryptionAlgorithm Enumeration.....	137
TCRHashAlgorithm Enumeration.....	137
TCRInvalidHashAction Enumeration.....	138
4 CRGrid	138
Classes	138
TCRDBGrid Class.....	138
Members	139
5 DAAlerter	139
Classes	139
TDAAlerter Class.....	139
Members	140
Properties	141
Active Property.....	141
AutoRegister Property.....	142
Connection Property.....	142
Methods	142
SendEvent Method.....	143
Start Method	143
Stop Method	144
Events	144
OnError Event.....	144
Types	145
TAlerterErrorEvent Procedure Reference.....	145
6 DADump	145
Classes	146
TDADump Class.....	146
Members	147
Properties	148
Connection Property.....	149
Debug Property.....	150
Options Property.....	150
SQL Property	151
TableNames Property.....	151
Methods	152
Backup Method.....	152
BackupQuery Method.....	153
BackupToFile Method.....	154

BackupToStream Method.....	154
Restore Method.....	155
RestoreFromFile Method.....	155
RestoreFromStream Method.....	156
Events	156
OnBackupProgress Event.....	157
OnError Event.....	158
OnRestoreProgress Event.....	158
TDADumpOptions Class.....	159
Members	159
Properties	159
AddDrop Property.....	160
GenerateHeader Property.....	160
QuoteNames Property.....	161
Types	161
TDABackupProgressEvent Procedure Reference.....	161
TDARestoreProgressEvent Procedure Reference.....	162
7 DALoader	162
Classes	163
TDAColumn Class.....	163
Members	164
Properties	164
FieldType Property.....	165
Name Property.....	165
TDAColumns Class.....	165
Members	166
Properties	166
Items Property(Indexer).....	166
TDALoader Class.....	167
Members	167
Properties	168
Columns Property.....	169
Connection Property	169
TableName Property.....	170
Methods	170
CreateColumns Method.....	171
Load Method	171
LoadFromDataSet Method.....	172
PutColumnData Method	172
PutColumnData Method	173
PutColumnData Method	173
Events	174
OnGetColumnData Event.....	174
OnProgress Event.....	175
OnPutData Event.....	176
TDALoaderOptions Class.....	177
Members	177
Properties	177
UseBlankValues Property.....	178
Types	178
TDAPutDataEvent Procedure Reference.....	178
TGetColumnDataEvent Procedure Reference.....	179
TLoaderProgressEvent Procedure Reference.....	179

8 DAScript	180
Classes	180
TDA Script Class.....	181
Members	181
Properties	183
Connection Property	184
DataSet Property.....	185
Debug Property.....	185
Delimiter Property.....	186
EndLine Property.....	186
EndOffset Property.....	186
EndPos Property	187
Macros Property	187
SQL Property	188
StartLine Property.....	188
StartOffset Property	188
StartPos Property.....	189
Statements Property	189
Methods	190
BreakExec Method.....	191
ErrorOffset Method.....	191
Execute Method.....	191
ExecuteFile Method.....	192
ExecuteNext Method.....	192
ExecuteStream Method.....	193
MacroByName Method.....	193
Events	194
AfterExecute Event.....	195
BeforeExecute Event.....	195
OnError Event.....	195
TDA Statement Class	196
Members	196
Properties	197
EndLine Property.....	198
EndOffset Property.....	199
EndPos Property	199
Omit Property	199
Params Property	200
Script Property.....	200
SQL Property	200
StartLine Property.....	201
StartOffset Property	201
StartPos Property.....	201
Methods	202
Execute Method.....	202
TDA Statements Class	202
Members	203
Properties	203
Items Property(Indexer).....	204
Types	204
TAfterStatementExecuteEvent Procedure Reference	204
TBeforeStatementExecuteEvent Procedure Reference.....	205
TOnErrorEvent Procedure Reference.....	205
Enumerations	206

ErrorAction Enumeration.....	206
9 DASQLMonitor	207
Classes	207
TCustomDASQLMonitor Class.....	208
Members	208
Properties	209
Active Property.....	209
DBMonitorOptions Property.....	210
Options Property.....	210
TraceFlags Property	211
Events	211
OnSQL Event	211
TDBMonitorOptions Class.....	212
Members	212
Properties	213
Host Property	213
Port Property	214
ReconnectTimeout Property.....	214
SendTimeout Property.....	214
Types	215
TDATraceFlags Set.....	215
TMonitorOptions Set.....	215
TOnSQLEvent Procedure Reference.....	216
Enumerations	216
TDATraceFlag Enumeration.....	216
TMonitorOption Enumeration.....	217
10 DBAccess	218
Classes	221
EDAEError Class.....	222
Members	223
Properties	223
Component Property.....	224
ErrorCode Property.....	224
TCRDataSource Class.....	224
Members	225
TCustomConnectDialog Class	225
Members	225
Properties	226
CancelButton Property.....	227
Caption Property	228
ConnectButton Property.....	228
DialogClass Property.....	228
LabelSet Property	229
PasswordLabel Property.....	229
Retries Property.....	230
SavePassword Property	230
ServerLabel Property.....	230
StoreLogInfo Property.....	231
UsernameLabel Property	231
Methods	231
Execute Method.....	232
GetServerList Method.....	232
TCustomDAConnection Class	233

Members	233
Properties	235
ConnectDialog Property	236
ConnectString Property.....	237
ConvertEOL Property.....	239
InTransaction Property.....	239
LoginPrompt Property.....	240
Options Property.....	240
Password Property.....	241
Pooling Property.....	241
PoolingOptions Property.....	242
Server Property.....	243
Username Property.....	243
Methods	244
ApplyUpdates Method.....	245
ApplyUpdates Method.....	246
ApplyUpdates Method.....	246
Commit Method.....	247
Connect Method.....	247
CreateSQL Method.....	248
Disconnect Method.....	248
ExecProc Method.....	249
ExecProcEx Method.....	250
ExecSQL Method.....	251
ExecSQLEx Method.....	252
GetDatabaseNames Method.....	253
GetKeyFieldNames Method.....	254
GetStoredProcNames Method.....	255
GetTableNames Method.....	255
MonitorMessage Method.....	256
Ping Method	257
RemoveFromPool Method.....	257
Rollback Method.....	257
StartTransaction Method.....	258
Events	259
OnConnectionLost Event.....	259
OnError Event.....	259
TCustomDataSet Class.....	260
Members	260
Properties	268
BaseSQL Property.....	271
Conditions Property.....	272
Connection Property.....	272
DataTypeMap Property.....	272
Debug Property.....	273
DetailFields Property.....	273
Disconnected Property.....	274
FetchRows Property.....	274
FilterSQL Property.....	274
FinalSQL Property.....	275
IsQuery Property.....	276
KeyFields Property.....	276
MacroCount Property.....	277
Macros Property.....	277

MasterFields Property.....	278
MasterSource Property.....	278
Options Property.....	279
ParamCheck Property.....	281
ParamCount Property.....	282
Params Property.....	282
ReadOnly Property.....	283
RefreshOptions Property.....	283
RowsAffected Property.....	284
SQL Property.....	284
SQLDelete Property.....	285
SQLInsert Property.....	285
SQLLock Property.....	286
SQLRecCount Property.....	287
SQLRefresh Property.....	287
SQLUpdate Property.....	288
UniDirectional Property.....	289
Methods.....	289
AddWhere Method.....	293
BreakExec Method.....	294
CreateBlobStream Method.....	294
DeleteWhere Method.....	295
Execute Method.....	295
Execute Method.....	296
Execute Method.....	296
Executing Method.....	297
Fetched Method.....	297
Fetching Method.....	298
FetchingAll Method.....	298
FindKey Method.....	299
FindMacro Method.....	299
FindNearest Method.....	299
FindParam Method.....	300
GetDataType Method.....	301
GetFieldObject Method.....	301
GetFieldPrecision Method.....	302
GetFieldScale Method.....	303
GetKeyFieldNames Method.....	303
GetOrderBy Method.....	304
GotoCurrent Method.....	304
Lock Method.....	305
MacroByName Method.....	305
ParamByName Method.....	306
Prepare Method.....	307
RefreshRecord Method.....	307
RestoreSQL Method.....	308
SaveSQL Method.....	308
SetOrderBy Method.....	309
SQLSaved Method.....	310
UnLock Method.....	310
Events.....	310
AfterExecute Event.....	311
AfterFetch Event.....	312
AfterUpdateExecute Event.....	312

BeforeFetch Event.....	312
BeforeUpdateExecute Event.....	313
TCustomDASQL Class.....	313
Members	314
Properties	315
ChangeCursor Property.....	317
Connection Property.....	317
Debug Property.....	318
FinalSQL Property.....	318
MacroCount Property.....	318
Macros Property.....	319
ParamCheck Property.....	319
ParamCount Property.....	320
Params Property.....	320
ParamValues Property(Indexer).....	321
Prepared Property.....	322
RowsAffected Property.....	322
SQL Property	323
Methods	323
Execute Method.....	324
Execute Method.....	324
Execute Method.....	325
Executing Method.....	325
FindMacro Method.....	326
FindParam Method.....	326
MacroByName Method.....	327
ParamByName Method.....	328
Prepare Method.....	328
UnPrepare Method.....	329
WaitExecuting Method.....	329
Events	330
AfterExecute Event.....	330
TCustomDAUpdateSQL Class.....	331
Members	331
Properties	332
DataSet Property.....	334
DeleteObject Property.....	334
DeleteSQL Property.....	334
InsertObject Property.....	335
InsertSQL Property.....	335
LockObject Property.....	336
LockSQL Property.....	336
ModifyObject Property.....	336
ModifySQL Property.....	337
RefreshObject Property.....	337
RefreshSQL Property.....	338
SQL Property(Indexer).....	338
Methods	339
Apply Method	339
ExecSQL Method.....	340
TDACondition Class.....	340
Members	341
Properties	341
Enabled Property.....	342

Name Property	342
Value Property	342
Methods	343
Disable Method	343
Enable Method	343
TDAConditions Class	344
Members	344
Properties	345
Condition Property(Indexer)	346
Enabled Property	346
Items Property(Indexer)	346
Text Property	347
WhereSQL Property	347
Methods	347
Add Method	348
Add Method	349
Add Method	349
Delete Method	350
Disable Method	350
Enable Method	350
Find Method	351
Get Method	351
IndexOf Method	351
Remove Method	352
TDAConnectionOptions Class	352
Members	352
Properties	353
Allow ImplicitConnect Property	354
DefaultSortType Property	354
DisconnectedMode Property	355
KeepDesignConnected Property	355
LocalFailover Property	355
TDADataSetOptions Class	356
Members	356
Properties	358
AutoPrepare Property	360
CacheCalcFields Property	361
CompressBlobMode Property	361
DefaultValues Property	361
DetailDelay Property	362
FieldsOrigin Property	362
FlatBuffers Property	363
InsertAllSetFields Property	363
LocalMasterDetail Property	363
LongStrings Property	364
MasterFieldsNullable Property	364
NumberRange Property	365
QueryRecCount Property	365
QuoteNames Property	365
RemoveOnRefresh Property	366
RequiredFields Property	366
ReturnParams Property	367
SetFieldsReadOnly Property	367
StrictUpdate Property	367

TrimFixedChar Property	368
UpdateAllFields Property.....	368
UpdateBatchSize Property.....	369
TDAEncryption Class	369
Members	369
Properties	370
Encryptor Property.....	370
Fields Property.....	371
TDAMapRule Class.....	371
Members	371
Properties	372
DBLengthMax Property.....	373
DBLengthMin Property.....	373
DBScaleMax Property.....	374
DBScaleMin Property.....	374
DBType Property.....	374
FieldLength Property.....	375
FieldName Property.....	375
FieldScale Property.....	376
FieldType Property.....	376
IgnoreErrors Property.....	376
TDAMapRules Class.....	377
Members	377
Properties	377
IgnoreInvalidRules Property.....	378
TDAMetaData Class	378
Members	379
Properties	382
Connection Property.....	383
MetaDataKind Property.....	384
Restrictions Property.....	385
Methods	385
GetMetaDataKinds Method.....	387
GetRestrictions Method.....	388
TDAParam Class	388
Members	389
Properties	391
AsBlob Property.....	392
AsBlobRef Property.....	392
AsFloat Property.....	393
AsInteger Property.....	393
AsLargeInt Property.....	393
AsMemo Property.....	394
AsMemoRef Property.....	394
AsSQLTimeStamp Property.....	395
AsString Property.....	395
AsWideString Property.....	395
DataType Property.....	396
IsNull Property.....	396
ParamType Property.....	397
Size Property	397
Value Property.....	397
Methods	398
AssignField Method.....	398

AssignFieldValue Method.....	399
LoadFromFile Method.....	399
LoadFromStream Method.....	400
SetBlobData Method.....	401
SetBlobData Method.....	401
SetBlobData Method.....	401
TDAParams Class.....	402
Members	402
Properties	403
Items Property(Indexer).....	403
Methods	404
FindParam Method.....	404
ParamByName Method.....	405
TDATransaction Class.....	405
Members	406
Properties	407
Active Property.....	407
DefaultCloseAction Property.....	407
Methods	408
Commit Method.....	408
Rollback Method.....	409
StartTransaction Method.....	409
Events	410
OnCommit Event.....	411
OnCommitRetaining Event.....	411
OnError Event.....	412
OnRollback Event.....	412
OnRollbackRetaining Event.....	413
TMacro Class.....	413
Members	414
Properties	415
Active Property.....	415
AsDateTime Property.....	416
AsFloat Property.....	416
AsInteger Property.....	417
AsString Property.....	417
Name Property.....	417
Value Property.....	418
TMacros Class.....	418
Members	418
Properties	419
Items Property(Indexer).....	419
Methods	420
AssignValues Method.....	420
Expand Method.....	421
FindMacro Method.....	421
IsEqual Method.....	422
MacroByName Method.....	422
Scan Method	423
TPoolingOptions Class.....	423
Members	424
Properties	424
ConnectionLifetime Property.....	425
MaxPoolSize Property.....	425

MinPoolSize Property	425
Validate Property	426
TSmartFetchOptions Class.....	426
Members	426
Properties	427
Enabled Property.....	427
LiveBlock Property.....	428
PrefetchedFields Property	428
SQLGetKeyValues Property.....	429
Types	429
TAfterExecuteEvent Procedure Reference.....	430
TAfterFetchEvent Procedure Reference.....	430
TBeforeFetchEvent Procedure Reference.....	431
TConnectionLostEvent Procedure Reference.....	431
TDAConnectionErrorEvent Procedure Reference.....	431
TDATransactionErrorEvent Procedure Reference.....	432
TRefreshOptions Set.....	432
TUpdateExecuteEvent Procedure Reference.....	433
Enumerations	433
TLabelSet Enumeration.....	434
TRefreshOption Enumeration.....	434
TRetryMode Enumeration.....	435
Variables	435
BaseSQLOldBehavior Variable.....	436
ChangeCursor Variable.....	436
SQLGeneratorCompatibility Variable.....	436
11 MemData	437
Classes	438
TAttribute Class.....	439
Members	439
Properties	440
AttributeNo Property	441
DataSize Property.....	441
DataType Property.....	441
Length Property	442
ObjectType Property.....	442
Offset Property.....	442
Owner Property.....	443
Scale Property.....	443
Size Property	444
TBlob Class	444
Members	445
Properties	446
AsString Property.....	447
AsWideString Property.....	447
IsUnicode Property.....	447
Size Property	448
Methods	448
Assign Method.....	449
Clear Method	450
LoadFromFile Method.....	450
LoadFromStream Method.....	450
Read Method	451
SaveToFile Method.....	452

SaveToStream Method.....	452
Truncate Method.....	453
Write Method	453
TCompressedBlob Class.....	454
Members	455
Properties	456
Compressed Property.....	457
CompressedSize Property.....	457
TDBObject Class.....	458
Members	458
TMemData Class.....	459
Members	459
TObjectType Class.....	459
Members	460
Properties	460
AttributeCount Property	461
Attributes Property (Indexer).....	461
DataType Property.....	462
Size Property	462
Methods	463
FindAttribute Method.....	463
TSharedObject Class.....	464
Members	464
Properties	465
RefCount Property	465
Methods	466
AddRef Method.....	466
Release Method.....	466
Types	467
TLocateExOptions Set.....	467
TUpdateRecKinds Set.....	467
Enumerations	468
TCompressBlobMode Enumeration.....	468
TConnLostCause Enumeration.....	469
TDANumericType Enumeration.....	470
TLocateExOption Enumeration.....	470
TSortType Enumeration.....	471
TUpdateRecKind Enumeration.....	472
12 MemDS	472
Classes	473
TMemDataSet Class	473
Members	473
Properties	476
CachedUpdates Property.....	477
IndexFieldNames Property	478
KeyExclusive Property.....	479
LocalConstraints Property	479
LocalUpdate Property	480
Prepared Property.....	480
Ranged Property.....	480
UpdateRecordTypes Property	481
UpdatesPending Property	481
Methods	482
ApplyRange Method.....	484

ApplyUpdates Method.....	484
ApplyUpdates Method.....	485
ApplyUpdates Method.....	486
CancelRange Method.....	486
CancelUpdates Method.....	487
CommitUpdates Method.....	488
DeferredPost Method.....	488
EditRangeEnd Method.....	489
EditRangeStart Method.....	489
GetBlob Method.....	490
GetBlob Method.....	490
GetBlob Method.....	491
Locate Method.....	491
Locate Method.....	492
Locate Method.....	492
LocateEx Method.....	493
LocateEx Method.....	494
LocateEx Method.....	494
Prepare Method.....	495
RestoreUpdates Method.....	496
RevertRecord Method.....	496
SaveToXML Method.....	497
SaveToXML Method.....	497
SaveToXML Method.....	498
SetRange Method.....	498
SetRangeEnd Method.....	499
SetRangeStart Method.....	500
UnPrepare Method.....	501
UpdateResult Method.....	501
UpdateStatus Method.....	502
Events	502
OnUpdateError Event.....	503
OnUpdateRecord Event.....	504
Variables	504
DoNotRaiseExcetionOnUaFail Variable.....	505
SendDataSetChangeEventAfterOpen Variable.....	505
13 PgAccess	505
Classes	508
TCustomPgDataSet Class.....	510
Members	510
Properties	520
Cursor Property.....	524
DMLRefresh Property	525
FetchAll Property	525
KeySequence Property.....	526
LastInsertOID Property.....	526
Options Property.....	527
Params Property	528
SequenceMode Property.....	528
UpdateObject Property.....	529
Methods	529
CreateProcCall Method.....	534
FindParam Method.....	535
GetPgCursor Method.....	535

GetPgDate Method.....	536
GetPgInterval Method.....	536
GetPgLargeObject Method.....	537
GetPgRow Method.....	537
GetPgTime Method.....	538
GetPgTimeStamp Method.....	538
OpenNext Method.....	539
ParamByName Method.....	539
TCustomPgStoredProc Class.....	540
Members	540
Properties	551
Overload Property.....	555
Methods	555
ExecProc Method.....	561
PrepareSQL Method.....	561
TCustomPgTable Class.....	562
Members	562
Properties	573
Limit Property	577
Offset Property.....	578
TCustomPgTimeStampField Class.....	578
Members	578
Properties	579
AsPgTimeStamp Property	579
TPgConnection Class.....	579
Members	580
Properties	584
ConnectionTimeout Property.....	586
Database Property.....	587
Options Property.....	587
Password Property.....	588
Port Property	589
ProcessID Property.....	589
ProtocolVersion Property.....	590
Schema Property.....	590
Server Property.....	590
ServerVersion Property.....	591
ServerVersionFull Property.....	591
SSLOptions Property	592
Username Property.....	592
Methods	593
BreakExec Method.....	595
CreateDataSet Method.....	596
CreateMetaData Method.....	596
CreateSQL Method.....	597
CreateTransaction Method.....	597
GetRow Type Method.....	597
GetRow Type Method.....	598
GetRow Type Method.....	598
ReleaseSavepoint Method.....	599
RollbackToSavepoint Method.....	599
Savepoint Method.....	600
StartTransaction Method.....	600
StartTransaction Method.....	601

StartTransaction Method.....	601
Events	602
OnNotice Event.....	603
OnNotification Event.....	603
TPgConnectionOptions Class.....	603
Members	604
Properties	605
ApplicationName Property.....	607
Charset Property.....	607
EnableComposites Property.....	608
EnableDomains Property.....	608
EnableGeometrics Property.....	609
EnablePgTimeStamps Property.....	609
IPVersion Property.....	610
UseUnicode Property.....	610
TPgConnectionSSLOptions Class.....	610
Members	611
Properties	611
CACert Property.....	612
Cert Property	612
CipherList Property.....	612
Key Property	613
Mode Property.....	613
TPgCursorField Class.....	613
Members	614
Properties	614
AsCursor Property.....	615
TPgDataSetOptions Class.....	615
Members	615
Properties	618
AutoDeleteBlob Property.....	621
CacheBlobs Property.....	622
CursorWithHold Property.....	622
DeferredBlobRead Property.....	623
ExtendedFieldsInfo Property.....	623
OIDAsInt Property.....	624
PrepareUpdateSQL Property.....	624
UnknownAsString Property.....	625
UnpreparedExecute Property.....	625
UseParamTypes Property.....	626
TPgDataSource Class.....	626
Members	627
TPgDateField Class.....	627
Members	627
Properties	627
AsPgDate Property.....	628
TPgEncryptor Class.....	628
Members	629
TPgGeometricField Class.....	629
Members	630
Properties	630
AsPgGeometric Property.....	630
TPgIntervalField Class.....	631
Members	631

Properties	631
AsPgInterval Property	632
TPgLargeObject Class	632
Members	633
Properties	635
Connection Property	636
TPgLargeObjectField Class	636
Members	636
Properties	637
AsLargeObject Property	637
TPgMetaData Class	637
Members	638
TPgParam Class	641
Members	642
Properties	644
AsPgDate Property	645
AsPgInterval Property	646
AsPgTime Property	646
AsPgTimeStamp Property	646
TPgParams Class	647
Members	647
Properties	648
Items Property (Indexer)	648
TPgQuery Class	649
Members	650
Properties	660
FetchAll Property	665
LockMode Property	665
UpdatingTable Property	666
TPgSQL Class	667
Members	667
Properties	669
CommandTimeout Property	671
Connection Property	671
LastInsertOID Property	672
Params Property	672
UnpreparedExecute Property	673
UseParamTypes Property	673
Methods	673
FindParam Method	674
ParamByName Method	675
TPgStoredProc Class	675
Members	676
Properties	687
CommandTimeout Property	691
LockMode Property	692
StoredProcName Property	692
TPgTable Class	692
Members	693
Properties	704
FetchAll Property	708
LockMode Property	709
OrderFields Property	710
TableName Property	710

TPgTimeField Class.....	710
Members	711
Properties	711
AsPgTime Property	712
TPgTimeStampField Class.....	712
Members	712
Properties	713
AsPgTimeStamp Property	713
TPgTransaction Class.....	713
Members	714
Properties	715
IsolationLevel Property.....	716
TPgUpdateSQL Class.....	716
Members	717
Types	718
TPgNoticeEvent Procedure Reference.....	718
TPgNotificationEvent Procedure Reference.....	719
Enumerations	720
TPgIsolationLevel Enumeration.....	720
Constants	720
PgDACVersion Constant.....	721
14 PgAlerter	721
Classes	721
TPgAlerter Class.....	721
Members	722
Properties	723
Events Property.....	724
Methods	724
SendEvent Method.....	725
SendEvent Method.....	725
SendEvent Method.....	725
Events	726
OnEvent Event.....	726
15 PgClasses	727
Classes	727
TPgCursor Class.....	727
Members	728
Properties	728
State Property.....	729
Enumerations	729
TProtocolVersion Enumeration.....	729
TSSLMode Enumeration.....	730
16 PgConnectionPool	730
Classes	731
TPgConnectionPoolManager Class.....	731
Members	731
17 PgDacVcl	732
Classes	732
TPgConnectDialog Class.....	732
Members	733
Properties	734
Connection Property.....	736
DatabaseLabel Property	736

PortLabel Property	737
Show Database Property	737
Show Port Property	738
18 PgDump	738
Classes	739
TPgDump Class	739
Members	740
Properties	741
Mode Property.....	742
ObjectTypes Property.....	743
Options Property.....	743
SchemaNames Property	744
TPgDumpOptions Class.....	744
Members	744
Properties	745
CreateConstraints Property	745
Types	746
TPgDumpObjects Set.....	746
Enumerations	746
TPgDumpMode Enumeration.....	747
TPgDumpObject Enumeration	747
19 PgError	748
Classes	748
EPgError Class	748
Members	749
Properties	750
CallStack Property.....	750
DetailMsg Property.....	751
ErrorCode Property.....	751
FileName Property.....	752
Hint Property	752
LineNumber Property	752
Position Property.....	753
ProcedureName Property.....	753
Severity Property.....	753
Enumerations	754
TPgSeverity Enumeration	754
20 PgLoader	754
Classes	755
TPgLoader Class.....	755
Members	756
Properties	757
BufferSize Property.....	757
Connection Property	758
TextMode Property.....	758
TPgLoaderColumn Class	759
Members	759
Properties	760
Row TypeName Property.....	760
21 PgObjects	760
Classes	762
TCustomPgTimeStamp Class.....	763
Members	764

Properties	765
AsDateTime Property.....	766
AsSQLTimeStamp Property.....	766
AsString Property.....	767
Days Property.....	767
HasTimeZone Property.....	768
IsInfinity Property.....	768
IsNegInfinity Property.....	768
IsPosInfinity Property.....	769
Ticks Property.....	769
TimeZoneOffset Property.....	769
Methods	770
Assign Method.....	771
Compare Method.....	771
DecodeDate Method.....	772
DecodeDateTime Method.....	772
DecodeTime Method.....	773
EncodeDate Method.....	774
EncodeDateTime Method.....	774
EncodeTime Method.....	775
TPgAttribute Class.....	776
Members	776
TPgBox Class.....	777
Members	777
Properties	778
LowerLeft Property.....	779
UpperRight Property.....	779
TPgCircle Class.....	779
Members	780
Properties	780
Center Property.....	781
Radius Property.....	781
TPgDate Class.....	782
Members	782
TPgGeometric Class.....	784
Members	785
Properties	785
AsString Property.....	786
Methods	786
Assign Method.....	787
TPgInterval Class.....	787
Members	787
Properties	788
AsString Property.....	789
Days Property.....	789
MonthsFull Property.....	790
SecondsFull Property.....	790
Methods	790
Assign Method.....	791
Compare Method.....	791
DecodeInterval Method.....	792
EncodeInterval Method.....	793
TPgLSeg Class.....	794
Members	794

Properties	795
EndPoint Property.....	795
StartPoint Property.....	796
TPgPath Class.....	796
Members	797
Properties	797
Count Property.....	798
IsClosedPath Property.....	798
Points Property.....	799
TPgPoint Class.....	799
Members	799
Properties	800
X Property	801
Y Property	801
Methods	801
Assign Method.....	802
TPgPointsArray Class.....	802
Members	803
TPgPolygon Class.....	803
Members	804
Properties	805
Count Property.....	805
Points Property.....	805
TPgRefCursor Class.....	806
Members	806
Properties	807
CursorName Property	808
TPgRow Class.....	808
Members	808
Properties	810
AsString Property.....	812
AttrAsPgBox Property(Indexer).....	812
AttrAsPgCircle Property(Indexer).....	812
AttrAsPgCursor Property(Indexer).....	813
AttrAsPgDate Property(Indexer).....	813
AttrAsPgInterval Property(Indexer).....	814
AttrAsPgLargeObject Property(Indexer).....	814
AttrAsPgLSeg Property(Indexer).....	815
AttrAsPgPath Property(Indexer).....	815
AttrAsPgPoint Property(Indexer).....	815
AttrAsPgPolygon Property(Indexer).....	816
AttrAsPgRow Property(Indexer).....	816
AttrAsPgTime Property(Indexer).....	817
AttrAsPgTimeStamp Property(Indexer).....	817
AttrIsNull Property(Indexer).....	818
AttrValue Property(Indexer).....	818
Row Type Property.....	818
Methods	819
Assign Method.....	819
TPgRow Type Class.....	820
Members	820
Methods	820
Describe Method.....	821
Describe Method.....	821

Describe Method.....	821
TPgSQLLargeObject Class.....	822
Members	822
Properties	824
Cached Property.....	825
Connection Property.....	826
OID Property	826
Methods	826
CloseObject Method.....	828
CreateObject Method.....	828
OpenObject Method.....	828
ReadBlob Method.....	829
UnlinkObject Method.....	829
WriteBlob Method.....	829
TPgTime Class.....	830
Members	830
Properties	832
TimeZoneOffset Property.....	834
TPgTimeStamp Class.....	834
Members	835
22 PgScript	837
Classes	837
TPgScript Class.....	837
Members	838
23 PgSQLMonitor	839
Classes	840
TPgSQLMonitor Class.....	840
Members	841
24 VirtualDataSet	841
Classes	841
TCustomVirtualDataSet Class.....	842
Members	842
TVirtualDataSet Class.....	845
Members	845
Types	848
TOnDeleteRecordEvent Procedure Reference.....	849
TOnGetFieldValueEvent Procedure Reference.....	849
TOnGetRecordCountEvent Procedure Reference.....	850
TOnModifyRecordEvent Procedure Reference.....	850
25 VirtualTable	850
Classes	850
TVirtualTable Class.....	851
Members	851

1 What's New

09-Jul-2018 New Features in PgDAC 5.2:

- Lazarus 1.8.4 is supported
- Performance of batch operations is improved
- Support for HTTP/HTTPS tunnel is added
- Demo projects for IntraWeb 14 are added
- Memory leak in NEXTGEN is fixed

19-Sep-2017 New Features in PgDAC 5.1:

- SSPI authentication is supported
- Processing GUID data type for the TGuidField class is improved

05-Apr-2017 New Features in PgDAC 5.0:

- RAD Studio 10.2 Tokyo is supported
- Linux in RAD Studio 10.2 Tokyo is supported
- Lazarus 1.6.4 and Free Pascal 3.0.2 is supported

25-Apr-16 New Features in PgDAC 4.7:

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

09-Sep-15 New Features in PgDAC 4.6:

- RAD Studio 10 Seattle is supported
- INSERT, UPDATE and DELETE batch operations are supported
- Support of bit and bit varying data types is improved
- Now Trial for Win64 is a fully functional Professional Edition

14-Apr-15 New Features in PgDAC 4.5:

- RAD Studio XE8 is supported
- AppMethod is supported
- PostgreSQL 9.4 is supported

15-Sep-14 New Features in PgDAC 4.4:

- RAD Studio XE7 is supported
- Lazarus 1.2.4 is supported
- Demo projects for FastReport 5 are added
- The TCustomDADataset.GetKeyFieldNames method is added
- The ConstraintColumns metadata kind for the TDAMetadata component is added

29-Apr-14 New Features in PgDAC 4.3:

- Delphi XE6 is supported
- Android in C++Builder XE6 is supported
- Lazarus 1.2.2 and FPC 2.6.4 is supported
- SmartFetch mode for TDataSet descendants is added
- Now update queries inside TDataSet descendants have correct owner
- The TPgDataSetOptions.MasterDetailNullable property is added

25-Dec-13 New Features in PgDAC 4.2:

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

18-Sep-13 New Features in PgDAC 4.1:

- RAD Studio XE5 is supported
- Application development for Android is supported
- Lazarus 1.0.12 is supported
- Performance is improved
- Automatic checking for new versions is added
- Flexible management of conditions in the WHERE clause is added
- The possibility to use conditions is added
- PostgreSQL 9.3 is supported

- IPv6 protocol support is added
- Support of the IN keyword in the TDataSet.Filter property is added
- Like operator behaviour when used in the Filter property is now similar to TClientDataSet
- The possibility to use ranges is added
- The Ping method for the Connection component is added
- The AllowImplicitConnect option for the Connection component is added
- The SQLRecCount property for the Query and StoredProc components is added
- The ScanParams property for the Script component is added
- The RowsAffected property for the Script component is added
- The EnableDomains option is added for TPgConnection
- ConnectionTimeout is now used when disconnecting after connection loss
- The TPgTable.TableName and TPgStoredProc.StoredProcName property editors are improved

25-Apr-13 New Features in PgDAC 4.0:

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

12-Dec-12 New Features in PgDAC 3.6:

- Rad Studio XE3 Update 1 is now required
- C++Builder 64-bit for Windows is supported

05-Sep-12 New Features in PgDAC 3.5:

- Rad Studio XE3 is supported
- Windows 8 is supported

23-Nov-11 New Features in PgDAC 3.1:

- Update 2 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
- Mac OS X and iOS in RAD Studio XE2 is supported

- FireMonkey support is improved
- Lazarus 0.9.30.2 and FPC 2.4.4 are supported
- Mac OS X in Lazarus is supported
- Linux x64 in Lazarus is supported
- FreeBSD in Lazarus is supported
- PostgreSQL 9.1 is supported

15-Sep-11 New Features in PostgreSQL Data Access Components 3.00:

- Embarcadero RAD Studio XE2 is supported
- Application development for 64-bit Windows is supported
- FireMonkey application development platform is supported
- Support of master/detail relationship for TVirtualTable is added
- OnProgress event in TVirtualTable is added
- TDatasetOptions.SetEmptyStrToNull property that allows inserting NULL value instead of empty string is added

28-Apr-11 New Features in PostgreSQL Data Access Components 2.20:

- Lazarus 0.9.30 and FPC 2.4.2 is supported
- Application Name connection option is supported
- Payload parameter for PostgreSQL notification is supported (TPgNotificationEvent changed: EventMessage parameter is added)

26-Jan-11 New Features in PostgreSQL Data Access Components 2.10:

- PostgreSQL 9.0 supported
- Improved performance
- Improved table names detecting inside SQL queries for the UpdatingTable property
- Case sensitive schema name
- Checking that dataset is open on calling the TDataSet.Locate method

13-Sep-10 New Features in PostgreSQL Data Access Components 2.00:

- Embarcadero RAD Studio XE supported

- Support of ONLY lexeme in the FROM statement
- Support for dbMonitor 3
- Added OnStart, OnCommit, OnRollback events to TDATTransaction
- Ability to lock records in the CachedUpdate mode
- Ability to send call stack information to the dbMonitor component
- Changed the LocateEx method behavior: now LocateEx centers records equal to Locate
- CursorWithHold option for TCustomPgDataSet to use FetchAll=False mode without transaction
- Now Required flag is set for UpdatingTable fields only
- Now the AssignConnect method copies transaction state

10-Sep-09 New Features in PostgreSQL Data Access Components 1.20:

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

02-Apr-09 New Features in PostgreSQL Data Access Components 1.10:

- Free Pascal under Linux supported
- DMLRefresh supported
- Added NoPreconnect property to TPgScript for executing CONNECT and CREATE DATABASE commands

2 General Information

This section contains general information about PostgreSQL Server Data Access Components

- [Overview](#)
- [Features](#)
- [Requirements](#)

- [Compatibility](#)
- [Using Several DAC Products in One IDE](#)
- [Component List](#)
- [Hierarchy Chart](#)
- [Editions](#)
- [Licensing and Subscriptions](#)
- [Getting Support](#)

2.1 Overview

PostgreSQL Data Access Components (PgDAC) is a library of components that provides direct access to PostgreSQL database servers from Delphi, C++Builder and Lazarus (Free Pascal). PgDAC is designed to help programmers develop really lightweight, faster and cleaner PostgreSQL database applications without deploying any additional libraries. PgDAC is a complete replacement for standard PostgreSQL connectivity solutions and presents an efficient alternative to the Borland Database Engine for access to PostgreSQL. The PgDAC library is actively developed and supported by the Devart Team. If you have questions about PgDAC, email the developers at pgdac@devart.com or visit PgDAC online at <https://www.devart.com/pgdac/>.

Advantages of PgDAC Technology

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

Wide Coverage of PostgreSQL Features

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

Native Connection Options

PgDAC does not require PostgreSQL client software installed what heightens its performance. PgDAC-based database applications are easy to deploy, do not require

installation of other data provider layers (such as BDE), and tend to be faster than those that use standard data connectivity solutions.

Optimized Code

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

Compatibility with other Connectivity Methods

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

Development and Support

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

A description of the PgDAC components is provided in the [Component List](#).

Key Features

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

- Direct access to server data without using client library. Does not require installation of other data provider layers (such as BDE and ODBC)
- Full support of [the latest versions of PostgreSQL Server](#)
- Support for all PostgreSQL Server data types
- Disconnected Model with automatic connection control for working with data offline
- Local Failover for detecting connection loss and implicitly reexecuting certain operations
- All types of local sorting and filtering, including by calculated and lookup fields
- [Automatic data updating](#) with [TPgQuery](#), [TPgTable](#), and [TPgStoredProc](#) components
- Unicode and national charset support
- Supports many PostgreSQL-specific features, such as notifications, notices, and sequences
- Advanced script execution functionality with the [TPgScript](#) component
- Support for [using macros](#) in SQL

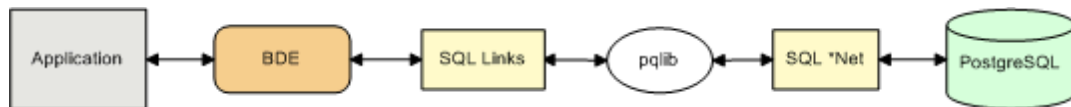
- Lets you use Professional Edition of [Delphi and C++Builder](#) to develop client/server applications
- Includes annual [PgDAC Subscription](#) with [Priority Support](#)
- Licensed royalty-free per developer, per team, or per site

The full list of PgDAC features are available in the [Features](#) topic.

How does PgDAC work?

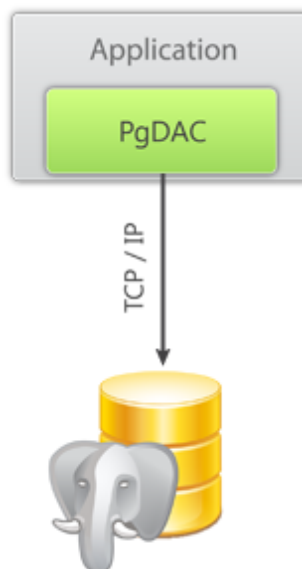
PgDAC connects to PostgreSQL directly without using client software.

In comparison, the Borland Database Engine (BDE) uses several layers to access PostgreSQL, and requires additional data access software to be installed on client machines. The BDE data transfer protocol is shown below.



BDE Connection Protocol

PgDAC optimal transfer route:



PgDAC Connection Flow

2.2 Features

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

General usability:

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

Network and connectivity:

- Does not require PostgreSQL client software and works directly through TCP/IP
- [Disconnected Model](#) with automatic connection control for working with data offline
- [Local Failover](#) for detecting connection loss and implicitly reexecuting certain operations
- Connection timeout and command timeout management
- SSL encrypted connection support with OpenSSL and Devart SecureBridge
- Ability to search for installed PostgreSQL Server databases in a local network

Compatibility:

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

PostgreSQL Server technology support:

- Support for fast record insertion with the [TPgLoader component](#)
- Support for PostgreSQL Asynchronous Notification with the [TPgAlerter component](#)
- PostgreSQL sequences support
- Supports the possibility of retrieving last inserted OID value

- Advanced errors support
- Support for the PostgreSQL notices

PostgreSQL DataTypes:

- Support for PostgreSQL Protocol 2 and Protocol 3
- PostgreSQL Composite types support
- PostgreSQL domain types support
- Full support of the [DATE](#) , [TIME](#) , [TIMESTAMP](#) , and [INTERVAL](#) data types
- Advanced [LARGE OBJECT](#) support
- Advanced support of the [REFCURSOR](#) type
- Wrapper classes for geometric types support

Performance:

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

Local data storage operations:

- Database-independent data storage with [TVirtualTable](#) component
- [CachedUpdates](#) operation mode
- Local [sorting](#) and filtering, including by calculated and lookup fields
- [Local master/detail](#) relationship
- Master/detail relationship in CachedUpdates mode

Data access and data management automation:

- Automatic [data updating](#) with [TPgQuery](#) , [TPgTable](#) , and [TPgStoredProc](#) components
- Automatic record [refreshing](#) and [locking](#)
- [Automatic query preparing](#)
- Automatic checking for row modifications by another user
- Support for ftWideMemo field type in Delphi 2006 and higher

Extended data access functionality:

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

Data exchange:

- Transferring data between all types of TDataSet descendants with [TCRBatchMove](#) component
- Data [export](#) and `M:Devart.Dac.TVirtualTable.LoadFromFile(System.String,System.Boolean)` to/from XML (ADO format)
- Ability to [synchronize positions in different DataSets](#)
- Extended data management with [TPgDump](#) component

Script execution:

- Advanced script execution features with the [TPgScript](#) component
- Support for executing [individual statements](#) in scripts
- Support for [executing huge scripts stored in files](#) with dynamic loading
- Ability to break long-running query execution

SQL Execution monitoring:

- Extended SQL tracing capabilities provided by [TPgSQLMonitor](#) component and [DBMonitor](#)
- Borland SQL Monitor support
- Ability to [send messages to DBMonitor](#) from any point in your program
- Ability to retrieve information about the last query execution

Visual extensions:

- Includes source code of enhanced TCRDBGrid data-aware grid control
- Customizable [connection dialog](#)
- [Cursor changes](#) during non-blocking execution

Design-time enhancements:

- [DataSet Manager](#) tool to control DataSet instances in the project
- Advanced design-time component and property editors
- Automatic design-time component linking
- More convenient data source setup with the [TPgDataSource](#) component
- Syntax highlighting in design-time editors

Product clarity:

- Complete documentation sets
- Printable documentation in PDF format
- A large amount of helpful [demo projects](#)

Licensing and support:

- Included annual [PgDAC Subscription](#) with [Priority Support](#)
- Licensed royalty-free per developer, per team, or per site

2.3 Requirements

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

2.4 Compatibility

PostgreSQL Compatibility

PgDAC supports PostgreSQL server versions since 8.0 up to 10.4.

Azure Database for PostgreSQL Compatibility

PgDAC also supports Azure Database for PostgreSQL.

Amazon Aurora Compatibility

PgDAC additionally supports Amazon Aurora.

IDE Compatibility

PgDAC is compatible with the following IDEs:

- Embarcadero RAD Studio 10.2 Tokyo
 - Embarcadero Delphi 10.2 Tokyo for Windows 32-bit & 64-bit
 - Embarcadero Delphi 10.2 Tokyo for macOS
 - Embarcadero Delphi 10.2 Tokyo for Linux 64-bit
 - Embarcadero Delphi 10.2 Tokyo for iOS 32-bit & 64-bit
 - Embarcadero Delphi 10.2 Tokyo for Android
 - Embarcadero C++Builder 10.2 Tokyo for Windows 32-bit & 64-bit
 - Embarcadero C++Builder 10.2 Tokyo for macOS
 - Embarcadero C++Builder 10.2 Tokyo for iOS 32-bit & 64-bit
 - Embarcadero C++Builder 10.2 Tokyo for Android
- Embarcadero RAD Studio 10.1 Berlin
 - Embarcadero Delphi 10.1 Berlin for Windows 32-bit & 64-bit
 - Embarcadero Delphi 10.1 Berlin for macOS
 - Embarcadero Delphi 10.1 Berlin for iOS 32-bit & 64-bit
 - Embarcadero Delphi 10.1 Berlin for Android
 - Embarcadero C++Builder 10.1 Berlin for Windows 32-bit & 64-bit
 - Embarcadero C++Builder 10.1 Berlin for macOS
 - Embarcadero C++Builder 10.1 Berlin for iOS 32-bit & 64-bit
 - Embarcadero C++Builder 10.1 Berlin for Android
- Embarcadero RAD Studio 10 Seattle
 - Embarcadero Delphi 10 Seattle for Windows 32-bit & 64-bit
 - Embarcadero Delphi 10 Seattle for macOS
 - Embarcadero Delphi 10 Seattle for iOS 32-bit & 64-bit
 - Embarcadero Delphi 10 Seattle for Android
 - Embarcadero C++Builder 10 Seattle for Windows 32-bit & 64-bit
 - Embarcadero C++Builder 10 Seattle for macOS
 - Embarcadero C++Builder 10 Seattle for iOS 32-bit & 64-bit
 - Embarcadero C++Builder 10 Seattle for Android
- Embarcadero RAD Studio XE8
 - Embarcadero Delphi XE8 for Windows 32-bit & 64-bit
 - Embarcadero Delphi XE8 for macOS
 - Embarcadero Delphi XE8 for iOS 32-bit & 64-bit
 - Embarcadero Delphi XE8 for Android
 - Embarcadero C++Builder XE8 for Windows 32-bit & 64-bit
 - Embarcadero C++Builder XE8 for macOS

- Embarcadero C++Builder XE8 for iOS 32-bit & 64-bit
- Embarcadero C++Builder XE8 for Android
- Embarcadero RAD Studio XE7
 - Embarcadero Delphi XE7 for Windows 32-bit & 64-bit
 - Embarcadero Delphi XE7 for macOS
 - Embarcadero Delphi XE7 for iOS
 - Embarcadero Delphi XE7 for Android
 - Embarcadero C++Builder XE7 for Windows 32-bit & 64-bit
 - 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 32-bit & 64-bit
 - Embarcadero Delphi XE6 for macOS
 - Embarcadero Delphi XE6 for iOS
 - Embarcadero Delphi XE6 for Android
 - Embarcadero C++Builder XE6 for Windows 32-bit & 64-bit
 - 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 32-bit & 64-bit
 - Embarcadero Delphi XE5 for macOS
 - Embarcadero Delphi XE5 for iOS
 - Embarcadero Delphi XE5 for Android
 - Embarcadero C++Builder XE5 for Windows 32-bit & 64-bit
 - Embarcadero C++Builder XE5 for macOS
 - Embarcadero C++Builder XE5 for iOS
- Embarcadero RAD Studio XE4
 - Embarcadero Delphi XE4 for Windows 32-bit & 64-bit
 - Embarcadero Delphi XE4 for macOS
 - Embarcadero Delphi XE4 for iOS
 - Embarcadero C++Builder XE4 for Windows 32-bit & 64-bit
 - Embarcadero C++Builder XE4 for macOS
- Embarcadero RAD Studio XE3 (Requires [Update 2](#))
 - Embarcadero Delphi XE3 for Windows 32-bit & 64-bit
 - Embarcadero Delphi XE3 for macOS

- Embarcadero C++Builder XE3 for Windows 32-bit & 64-bit
- Embarcadero C++Builder XE3 for macOS
- Embarcadero RAD Studio XE2 (Requires [Update 4 Hotfix 1](#))
 - Embarcadero Delphi XE2 for Windows 32-bit & 64-bit
 - Embarcadero Delphi XE2 for macOS
 - Embarcadero C++Builder XE2 for Windows 32-bit
 - 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
- CodeGear RAD Studio 2006
 - CodeGear Delphi 2006
 - CodeGear 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](#) 1.8.4 and [Free Pascal](#) 3.0.4 for Windows, Linux, macOS, FreeBSD for 32-bit and 64-bit platforms

Only Architect, Enterprise, and Professional IDE editions are supported. For Delphi/C++Builder XE and higher PgDAC additionally supports Starter Edition.

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

- Linux, 32-bit (only in Lazarus and Free Pascal) and 64-bit
- iOS, 32-bit and 64-bit
- Android
- FreeBSD (only in Lazarus and Free Pascal) 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.

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 PostgreSQL Data Access Components library. Click on the name of each component for more information.



These components are added to the PgDAC page of the Component palette except for [TCRBatchMove](#) and [TVirtualTable](#) components. [TCRBatchMove](#) and [TVirtualTable](#)




components are added to the Data Access page of the Component palette. Basic PgDAC components are included in all PgDAC editions. PgDAC Professional Edition components are not included in PgDAC Standard Edition.

Basic PgDAC components

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

PgDAC Professional Edition components

	TPgEncryptor	Represents data encryption and decryption in client application.
	TPgLoader	Provides quick loading of external data into the server database.

	TPgDump	Serves to store a database or its parts as a script and also to restore database from received script.
	TPgMetaData	Retrieves metadata on specified SQL object.
	TCRBatchMove	Retrieves metadata on database objects from the server.

See Also

- [Hierarchy chart](#)

2.7 Hierarchy Chart

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

TObject

```

|-TPersistent
|-TComponent
|   |-TCustomConnection
|   |   |-TCustomDAConnection
|   |   |-TPgConnection
|   |-TDataSet
|   |   |-TMemDataSet
|   |   |-TCustomDADataSet
|   |   |   |-TCustomPgDataSet
|   |   |   |   |-TPgQuery
|   |   |   |   |-TCustomPgTable
|   |   |   |   |   |-TPgTable
|   |   |   |   |   |-TCustomPgStoredProc
|   |   |   |   |   |   |-TPgStoredProc
|   |   |   |-TDAMetaData
|   |   |   |   |-TPgMetaData
|   |   |   |-TVirtualTable
|   |-TDataSource
|   |   |-TCRDataSource
|   |   |-TPgDataSource

```

```

|      |-DADDataAdapter
|      |      |-PgDataAdapter
|      |-TCRBatchMove
|      |-TCustomConnectDialog
|      |      |-TPgConnectDialog
|      |-TCustomDASQL
|      |      |-TPgSQL
|      |-TCustomDASQLMonitor
|      |      |-TPgSQLMonitor
|      |-TDALoader
|      |      |-TPgLoader
|      |-TDAScript
|      |      |-TPgScript
|      |-TDADump
|      |      |-TPgDump
|      |-TDATransaction
|      |      |-TPgTransaction
|      |-TDAAlerter
|      |      |-TPgAlerter
|      |-TCREncryptor
|      |      |-TPgEncryptor
|-TSharedObject
|      |-TBlob
|      |      |-TCompressedBlob
|      |      |      |-TPgSQLLargeObject
|      |      |      |-TPgLargeObject
|      |-TObjectType
|      |      |-TPgRowType
|      |-TDBObject
|      |      |-TPgRow
|      |-TCRCursor
|      |      |-TPgCursor
|      |      |-TPgRefCursor
|      |-TCustomPgTimeStamp
|      |      |-TPgTimeStamp
|      |      |-TPgDate
|      |      |-TPgTime

```

```

| - TPgInterval
| - TPgGeometric
|   | - TPgPoint
|   | - TPgCircle
|   | - PgPointsArray
|     | - TPgLSeg
|     | - TPgBox
|     | - TPgPath
|     | - TPgPolygon

```

2.8 Editions

PostgreSQL Data Access Components comes in three editions: PgDAC Standard Edition, PgDAC Professional Edition, and PgDAC Trial Edition.

PgDAC Standard Edition includes the PgDAC basic connectivity components. PgDAC Standard Edition is a cost-effective solution for database application developers who are looking for overall high performance connectivity to PostgreSQL.

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

PgDAC Trial Edition is the evaluation version of PgDAC. It includes all the functionality of PgDAC Professional Edition with a trial limitation of 60 days. C++Builder has additional trial limitations.

You can get **Source Access** to the PgDAC Standard and PgDAC Professional Editions by purchasing the special PgDAC Standard Edition with Source Code or PgDAC Professional Edition with Source Code. The Standard and Professional editions include the source code for all component classes. The source code of DataSet Manager is not distributed.

FreePascal support is available in Editions with **Source Code** and **Trial Edition**.

PgDAC Edition Matrix

Feature	Standard	Professional
Direct connectivity		
Connection without PostgreSQL client library	✓	✓
Desktop Application Development		

Windows	✓	✓
macOS	✓	✓
Linux	✗	✓
Mobile Application Development		
iOS	✗	✓
Android	✗	✓
Data Access Components		
Base Components: TPgConnection TPgQuery TPgSQL TPgTable TPgStoredProc TPgUpdateSQL TPgDataSource	✓	✓
Script executing TPgScript	✗	✓
Fast data loading into the server TPgLoader	✗	✓
PostgreSQL Specific Components		
Messaging between sessions and applications TPgAlerter	✗	✓
Obtaining metainformation about database objects TPgMetaData	✗	✓
Storing a database as a script TPgDump	✗	✓
DataBase Activity Monitoring		
Monitoring of per-component SQL execution TPgSQLMonitor	✓	✓

Additional components			
Advanced connection dialog TPgConnectDialog	✓	✓	
Data encryption and decryption TPgEncryptor	✗	✓	
Data storing in memory table TVirtualTable	✓	✓	
Advanced DBGrid with extended functionality TCRDBGrid	✗	✓	
Records transferring between datasets TCRBatchMove	✗	✓	
Design-Time Features			
Enhanced component and property editors	✓	✓	
DataSet Manager	✗	✓	
Cross IDE Support			
Lazarus and Free Pascal Support *	✗	✓	

* Available only in editions with source code.

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 PgDAC software application, source code, demos, intermediate files, printed materials, and online or electronic

documentation contained in this installation file. For the purpose of this Agreement, the software program(s) and supporting documentation will be referred to as the "Software".

LICENSE

1. GRANT OF LICENSE

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

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

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

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

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

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

No other rights or privileges are granted in this Agreement.

2. LIMITATIONS

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

- 2.1. You may not reverse engineer, decompile, or disassemble the Software.
- 2.2. You may not build any other components through inheritance for public distribution or commercial sale.
- 2.3. You may not use any part of the source code of the Software (original or modified) to build any other components for public distribution or commercial sale.
- 2.4. You may not reproduce or distribute any Software documentation without express written permission from Devart.
- 2.5. You may not distribute and sell any portion of the Software without integrating it into your Applications as Executable Code, except Trial edition that can be distributed for free as original Devart's PgDAC Trial package.
- 2.6. You may not transfer, assign, or modify the Software in whole or in part. In particular, the Software license is non-transferable, and you may not transfer the Software installation package.
- 2.7. You may not remove or alter any Devart's copyright, trademark, or other proprietary rights notice contained in any portion of Devart units, source code, or other files that bear such a notice.

3. REDISTRIBUTION

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

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

4. TRANSFER

You may not transfer the Software to any individual or entity without express written permission from Devart. In particular, you may not share copies of the Software under "Single Developer License" and "Team License" with other co-developers without obtaining proper license of these copies for each individual.

5. TERMINATION

Devart may immediately terminate this Agreement without notice or judicial resolution in the event of any failure to comply with any provision of this Agreement. Upon such termination you must destroy the Software, all accompanying written materials, and all copies.

6. EVALUATION

Devart may provide evaluation ("Trial") versions of the Software. You may transfer or distribute Trial versions of the Software as an original installation package only. If the Software

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

7. WARRANTY

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

8. SUBSCRIPTION AND SUPPORT

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

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

9. COPYRIGHT

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

2.10 Getting Support

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

Support Options

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

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

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

PgDAC Priority Support

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

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

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

2.11 FAQ

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

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

Installation and Deployment

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

You may be having a compatibility issue that shows up in one or more of the following forms:

- Get a "Setup has detected already installed DAC packages which are incompatible with current version" message during PgDAC installation.
- Get a "Procedure entry point ... not found in ..." message when starting IDE.
- Get a "Unit ... was compiled with a different version of ..." message on compilation.

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

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

Usually, you do not need any additional files. The only exceptions to this rule are listed below:

- If you are connecting in Client mode, (TPgConnection.Options.Direct = False), you need *PostgreSQL client library*.
- If you are using SSL (TPgConnection.Options.Protocol = mpSSL), you need the OpenSSL library files - *ssleay32.dll* and *libeay32.dll*.

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

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

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

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

Licensing and Subscriptions

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

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

2. Can I create components using PgDAC?

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

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

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

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

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

Users with a valid PgDAC Subscription get the following benefits:

- Access to new versions of PgDAC when they are released
- Access to all PgDAC updates and bug fixes
- Product support through the PgDAC Priority Support program
- Notification of new product versions

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

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

Yes, you can. PgDAC version licenses are perpetual.

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

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

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

7. How do I upgrade?

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

Performance

1. How productive is PgDAC?

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

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

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

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

How To

1. How can I quickly convert a project from BDE to PgDAC?

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 PgDAC menu of your IDE.

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

You can determine your PgDAC version number in several ways:

- During installation of PgDAC, consult the PgDAC Installer screen.
- After installation, see the *history.html* file in your PgDAC installation directory.
- At design-time, select PostgreSQL | About PgDAC from the main menu of your IDE.
- At run-time, check the value of the PgDACVersion and DACVersion constants.

3. 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.

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

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

In special cases, you can set these properties to perform more complicated processing during a query. These properties are automatically processed by PgDAC during the execution

of the Post, Delete, or RefreshRecord methods, and are used to construct the query to the server. Their values can contain parameters with names of fields in the underlying data source, which will be later replaced by appropriate data values.

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

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

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

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

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

Use the TPgConnection.GetDatabaseNames method.

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

Use the TPgConnection.GetTableNames method.

7. Some questions about the visual part of PgDAC

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 PgDACVcl (for Windows) or PgDACClx (for Linux) unit to the uses clause of your project.

General Questions

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

[DbxMda](#) is our dbExpress driver for PostgreSQL. dbExpress technology serves for providing a more or less uniform way to access different servers (SQL Server, PostgreSQL, Oracle

and so on). It is based on drivers that include server-specific features. Like any universal tool, in many specialized cases dbExpress providers lose some functionality. For example, the dbExpress design-time is quite poor and cannot be expanded.

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

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

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

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

2. Are the PgDAC connection components thread-safe?

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

3. Behaviour of my application has changed when I upgraded PgDAC. How can I restore the old behaviour with the new version?

We always try to keep PgDAC compatible with previous versions, but sometimes we have to change behaviour of PgDAC in order to enhance its functionality, or avoid bugs. If either of changes is undesirable for your application, and you want to save the old behaviour, please refer to the "Compatibility with previous versions" topic in PgDAC help. This topic describes such changes, and how to revert to the old PgDAC behaviour.

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

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

TCustomPgDataSet.Options.StrictUpdate topic in PgDAC help for more information.

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

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

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

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

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

- Client side: The value of TPgConnection.ConnectionTimeout or TCustomPgDataSet.CommandTimeout is too small. To check this hypothesis, try setting TCustomPgDataSet.CommandTimeout to 0 (infinite) and TPgConnection.ConnectionTimeout to 300.
- Server side: PostgreSQL server has closed the connection. Almost always it is because the value of wait_timeout variable is too small. Try increasing it. If this solution is not possible (for example, because you don't have enough rights), you should invoke PgConnection.Ping with an interval less than wait_timeout. Use TTimer in TPgConnection thread to accomplish this task.
- Unstable connection (GPRS etc). In case of unstable connection you can adapt PgDAC to work in such conditions by changing some of its settings. For more information please see the "Working in Unstable Networks" article in the PgDAC help documentation.

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

7. Some problems using TCustomDADataset.FetchAll=False mode

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

- I have problems working with temporary tables.

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

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

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

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

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

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

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

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

3 Getting Started

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

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

What is PgDAC?

PostgreSQL Data Access Components (PgDAC) is a component library which provides direct connectivity to PostgreSQL for Delphi, C++Builder and Lazarus (FPC), and helps you

develop fast PostgreSQL-based database applications with these environments.

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

An introduction to PgDAC is provided in the [Overview](#) section.

A list of the PgDAC features you may find useful is listed in the [Features](#) section.

An overview of the PgDAC component classes is provided in the [Components List](#) section.

Installing PgDAC

To install PgDAC, complete the following steps.

1. Choose and download the version of the PgDAC installation program that is compatible with your IDE. For instance, if you are installing PgDAC 1.00, you should use the following files:

For BDS 2006 and Turbo - **pgdac100d10*.exe**

For Delphi 7 - **pgdac1100d7*.exe**

For more information, visit the [PgDAC download page](#).

2. Close all running Borland applications.
3. Launch the PgDAC installation program you downloaded in the first step and follow the instructions to install PgDAC.

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

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

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

To find out what gets installed with PgDAC or to troubleshoot your PgDAC installation, visit the [Installation](#) topic.

Working with the PgDAC demo projects

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

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

PgDacDemo Walkthrough

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

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

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

4. Select Run | Run or press F9 to compile and launch the demo project. *PgDacDemo* should start, and a full-screen PgDAC Demo window with a toolbar, an explorer panel, and a view panel will open. The explorer panel will contain the list of all demo sub-projects included in *PgDacDemo*, and the view panel will contain an overview of each included demo.

At this point, you will be able to browse through the available demos, read their descriptions, view their source code, and see the functionality provided by each demo for interacting with PostgreSQL. However, you will not be able to actually retrieve data from PostgreSQL or execute commands until you connect to the database.

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

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

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

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

An error has occurred:

#42S01Table 'dept' already exists

You can manually create objects required for demo by using the following file: %PgDAC%\Demos\InstallDemoObjects.sql

%PgDAC% is the PgDAC installation path on your computer.

Ignore this exception?

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

7. Choose a demo that demonstrates an aspect of working with PostgreSQL that you are interested in, and play with the demo frame in the view window on the right. For example, to find out more about how to work with PostgreSQL tables, select the Table demo from the "Working with Components" folder. A simple PostgreSQL table browser will open in the view panel which will let you open a table in your database by specifying its name and clicking on the Open button.
8. Click on the "Demo source" button in the *PgDacDemo* toolbar to find out how the demo you selected was implemented. The source code behind the demo project will appear in the view panel. Try to find the places where PgDAC components are used to connect to the database.
9. Click on the "Form as text" button in the *PgDacDemo* toolbar to view the code behind the interface to the demo. Try to find the places where PgDAC components are created on the demo form.
10. Repeat these steps for other demos listed in the explorer window. The available demos are organized in three folders.

Working with components

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

General demos

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

PostgreSQL-specific demos

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

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

Other PgDAC demo projects

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

Compiling and deploying your PgDAC project

Compiling PgDAC-based projects

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

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

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

Deploying PgDAC-based projects

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

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

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

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

Using the PgDAC documentation

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

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

corresponding topic in your IDE documentation.

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

How to get help with PgDAC

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

- If you have a question about PgDAC installation or licensing, consult the [Licensing](#) section.
- You can get community assistance and PgDAC technical support on the [PgDAC Support Forum](#).
- To get help through the PgDAC [Priority Support](#) program, send an email to the PgDAC development team at pgdac@devart.com.
- If you have a question about ordering PgDAC or any other Devart product, contact sales@devart.com.

For more information, consult the [Getting Support](#) topic.

3.1 Installation

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

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

Before installing PgDAC ...

Two versions of PgDAC cannot be installed in parallel for the same IDE, and, since the Devart Data Access Components products have some shared bpl files, newer versions of PgDAC may be incompatible with older versions of ODAC, IBDAC, SDAC, and UniDAC. So before installing a new version of PgDAC, uninstall all previous versions of PgDAC you may have, and check if your new install is compatible with other Devart Data Access Components products you have installed. For more information please see [Using several products in one IDE](#). If you run into problems or have any compatibility questions, please email pgdac@devart.com

Note: You can avoid performing PgDAC uninstallation manually when upgrading to a new version by directing the PgDAC installation program to overwrite previous versions. To do

this, execute the installation program from the command line with a / f or ce parameter (Start | Run and type pgdacXX. exe / f or ce, specifying the full path to the appropriate version of the installation program) .

Installed packages

Note: %PgDAC%denotes the path to your PgDAC installation directory.

Delphi/C++Builder Win32 project packages

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

Additional packages for using PgDAC managers and wizards

<i>Name</i>	<i>Description</i>	<i>Location</i>
datasetmanagerXX.bpl	DataSet Manager package	Delphi\Bin

Environment Changes

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

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

Delphi

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

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

C++Builder

C++Builder 6:

- `$(BCB) \ PgDAC \ Li b` should be included in the Library Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.
- `$(BCB) \ PgDAC \ I nc l ude` should be included in the Include Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.

C++Builder 2006, 2007:

- `$(BCB) \ PgDAC \ Li b` should be included in the Library search path of the Default Project Options accessible from Project | Default Options | C++Builder | Linker | Paths and Defines.
- `$(BCB) \ PgDAC \ I nc l ude` should be included in the Include search path of the Default Project Options accessible from Project | Default Options | C++Builder | C++ Compiler | Paths and Defines.

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

Lazarus

The PgDAC installation program only copies PgDAC files. You need to install PgDAC packages to the Lazarus IDE manually. Open `%PgDAC%\Source\Lazarus1\dcldpgdac10.lpk` (for Trial version `%PgDAC%\Packages\dcldpgdac10.lpk`) file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilt with PgDAC packages.

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

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

DBMonitor

DBMonitor is an easy-to-use tool to provide visual monitoring of your database applications. It is provided as an alternative to Borland SQL Monitor which is also supported by PgDAC.

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

3.2 Logging on to PostgreSQL

This tutorial describes how to connect to PostgreSQL.

Contents

1. [Requirements](#)
2. [General information](#)
3. [Creating Connection](#)
 - 3.1 [Design time creation](#)
 - 3.1.1 [Using TPgConnection Editor](#)
 - 3.1.2 [Using Object Inspector](#)
 - 3.2 [Run time creation](#)
4. [Opening connection](#)
5. [Closing connection](#)
6. [Modifying connection](#)
7. [Additional information](#)
8. [See Also](#)

Requirements

In order to connect to PostgreSQL, you need the server itself running, PgDAC installed, and IDE running. Also, you need to know the server name (if the server is run on the remote computer), the port that the server listens to (if you use not the 5432 standard port), and the database name.

General information

To establish connection to the server, you have to provide some connection parameters to PgDAC. This information is used by the TPgConnection component to establish connection to the server. The parameters are represented by the properties of the TPgConnection component (Server, Port, Username, Password, Database, Schema).

Note: All these options can be specified at once using the `ConnectionString` property.

Creating Connection

Design time creation

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

1. Open the Tool Palette and find the TPgConnection component in the PgDAC category.

2. Double-click on the component. Note that the new object appears on the form. If this is the first time you create TPgConnection in this application, it is named PgConnection1.

After you have done these steps, you should set up the newly created PgConnection1 component. You can do as follows:

Using TPgConnection Editor

1. Double-click on the PgConnection1 object.
2. In the **Server** edit box specify a DNS name or IP address of the computer, where PostgreSQL resides (for example, **localhost**). If not the 5432 standard port must be used, it can be specified in the Port edit box in the following format: Port (for example, **5432**).
3. Specify a login (for example, **postgres** in the **Username** edit box.
4. Specify a password (for example, **password**) in the **Password** edit box. If a login does not have a password, leave the **Password** edit box blank.
5. In the Database edit box specify the database name (for example, **postgres**). If **Database** is not specified, the **postgres** system database is used.
6. In the Schema edit box specify the schema name (for example, **public**).

Using Object Inspector

1. Click on the PgConnection1 object and press **F11** to focus on object's properties.
2. Set the **Server** property to a DNS name or IP address of the computer, where PostgreSQL resides (for example, **localhost**). If not the 5432 standard port must be used, it can be specified in the Port edit box in the following format: Port (for example, **5432**).
3. Specify a login (for example, **postgres** in the **Username** edit box.
4. Specify a password (for example, **password**) in the **Password** edit box. If a login does not have a password, leave the **Password** edit box blank.
5. In the Database edit box specify the database name (for example, **postgres**). If **Database** is not specified, the **postgres** system database is used.
6. In the Schema edit box specify the schema name (for example, **public**).

Run time creation

The same operations performed in runtime look as follows:

[Delphi]

```
var
  con: TPgConnection;
begin
  con := TPgConnection.Create(nil);
  try
    con.Server := 'localhost';
    con.Port := 5432;
```

```

con.Username := 'postgres';
con.Password := 'password';
con.Database := 'postgres';
con.Schema := 'public';
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 PgAccess unit to the USES clause of your unit.

[C++Builder]

```

{
TPgConnection* con = new TPgConnection(NULL);
try
{
con->Server = "localhost";
con->Port = 5432;
con->Username = "postgres";
con->Password = "password";
con->Database = "postgres";
con->Schema = "public";
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 PgAccess.hpp header file to your unit.

And using the ConnectString property:

[Delphi]

```

var
con: TPgConnection;
begin
con := TPgConnection.Create(nil);
try
con.ConnectString := 'Data Source=localhost;Port=5432;Database=postgres';
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 PgAccess units to the USES clause of your unit.

[C++ Builder]

```
{
    TPgConnection* con = new TPgConnection(NULL);
    try
    {
        con->ConnectionString = "Data Source=localhost;Port=5432;Database=postgres;";
        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 PgAccess.hpp header file to your unit.

Opening connection

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

[Delphi]

```
con.Open;
```

[C++ Builder]

```
con->Open();
```

Another way to open a 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, PgConnection1 must have valid connection options assigned earlier. When you call Open, PgDAC tries to find the host and connect to the server. If any problem occurs, it raises an exception with brief explanation on what is wrong. If no problem is encountered, PgDAC tries to establish connection. Finally, when connection is established, the Open method returns and the Connected property is changed to True.

Closing connection

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

[Delphi]

```
con.Close;
```

[C++ Builder]

```
con.Close();
```

or:

[Delphi]

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

[C++ Builder]

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

Modifying connection

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

Additional information

PgDAC 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
- Notice

See Also

- [TPgConnection](#)
- [Server](#)
- [Port](#)
- [Database](#)
- [Username](#)
- [Password](#)
- [Schema](#)
- [LoginPrompt](#)

3.3 Creating Database Objects

This tutorial describes how to create tables, stored procedures and other objects on PostgreSQL.

1. [Requirements](#)
2. [General information](#)
3. [Creating tables](#)
 - 3.1 [Design-time creation](#)
 - 3.2 [Run-time creation](#)
4. [Creating Stored Procedures](#)
 - 4.1 [Design Time Creation](#)
 - 4.2 [Run Time Creation](#)
5. [Additional information](#)

Requirements

In order to create database objects you have to connect to PostgreSQL. This process is described in details in the tutorial ["Connecting To PostgreSQL"](#).

General information

Database objects are created using Data Definition Language (DDL), which is a part of SQL. The DDL statements can be executed on server by an account that has the necessary privileges. There are two ways to create database objects. You can build DDL statements manually and execute them using a component like TPgSQL. Another way is to use visual database tools like pgAdmin. This topic covers the first way - using components. There are two ways to execute DDL statements in components like TPgSQL: in design-time and in run-time. Both these ways are described below.

Note: The following assumes that you have the IDE running, you are currently focused on the form designer, and you have already set up the TPgConnection component on the form.

Creating tables

To create tables, the TPgSQL component is used here.

Design-time creation

- Open the Tool Palette and find the TPgSQL component in the PgDAC category.
- Double-click on the component. Note that a new object appears on the form. If this is the first time you create TPgSQL in this application, it is named PgSQL1. Note that the

PgSQL1.Connection property is already set to an existent (on the form) connection.

- Double-click on the PgSQL1 object.
- Type the following lines:

```
CREATE TABLE dept (  
deptno serial not null,  
dname varchar(14),  
loc varchar(13),  
primary key (deptno)  
);  
CREATE TABLE emp (  
empno serial not null,  
ename varchar(10),  
job varchar(9),  
mgr integer,  
hiredate timestamp,  
sal real,  
comm real,  
deptno int references dept,  
primary key (empno)  
);
```

- Click on the Execute button. This will create two tables that we will use for tutorial purposes.

Run-time creation

Same operations performed in runtime look as follows:

[Delphi]

```
var  
  sql: TPgSQL;  
begin  
  sql := TPgSQL.Create(nil);  
  try  
    sql.Connection := con; // con is TPgConnection already set up  
    sql.SQL.Clear;  
    sql.SQL.Add('CREATE TABLE dept (');  
    sql.SQL.Add('  deptno serial not null,');  
    sql.SQL.Add('  dname varchar(14),');  
    sql.SQL.Add('  loc varchar(13),');  
    sql.SQL.Add('  primary key (deptno)');  
    sql.SQL.Add(');');  
    sql.SQL.Add('CREATE TABLE emp (');  
    sql.SQL.Add('  empno serial not null,');  
    sql.SQL.Add('  ename varchar(10),');  
    sql.SQL.Add('  job varchar(9),');  
    sql.SQL.Add('  mgr integer,');  
    sql.SQL.Add('  hiredate timestamp,');  
    sql.SQL.Add('  sal real,');  
    sql.SQL.Add('  comm real,');  
  end try;  
end;
```

```

    sql.SQL.Add(' deptno int references dept,');
    sql.SQL.Add(' primary key (empno)');
    sql.SQL.Add(');');
    sql.Execute;
finally
    sql.Free;
end;
end;

```

[C++Builder]

```

{
    TPgSQL* sql = new TPgSQL(NULL);
    try
    {
        sql->Connection = con; // con is TPgConnection already set up
        sql->SQL->Clear();
        sql->SQL->Add("CREATE TABLE dept (");
        sql->SQL->Add(" deptno serial not null,");
        sql->SQL->Add("  dname varchar(14),");
        sql->SQL->Add("  loc varchar(13),");
        sql->SQL->Add(" primary key (deptno)");
        sql->SQL->Add(");");
        sql->SQL->Add("CREATE TABLE emp (");
        sql->SQL->Add(" empno serial not null,");
        sql->SQL->Add("  ename varchar(10),");
        sql->SQL->Add("  job varchar(9),");
        sql->SQL->Add("  mgr integer,");
        sql->SQL->Add("  hiredate timestamp,");
        sql->SQL->Add("  sal real,");
        sql->SQL->Add("  comm real,");
        sql->SQL->Add("  deptno int references dept,");
        sql->SQL->Add(" primary key (empno)");
        sql->SQL->Add(");");
        sql->Execute();
    }
    __finally
    {
        sql->Free();
    }
}

```

Creating Stored Procedures

To create tables, the TPgScript component is used here.

Design-time creation

- Open the Tool Palette and find the TPgScript component in the PgDAC category.
- Double-click on the component. Note that a new object appears on the form. If this is the first time you create TPgScript in this application, it is named PgScript1. Note that the PgScript1.Connection property is already set to existent (on the form) connection.
- Double-click on the PgScript1 object.

- Type the following lines:

```
CREATE FUNCTION "Ten Most High-Paid Employees"()
RETURNS SETOF Emp AS $$
  SELECT * FROM emp ORDER BY emp.sal DESC LIMIT 10
$$ LANGUAGE 'sql';
CREATE FUNCTION "GetEmpNumberInDept"(
  IN pdeptno integer,
  OUT pempnumb integer)
RETURNS integer AS
$BODY$
BEGIN
  pempnumb := (SELECT count(*) FROM emp WHERE deptno = pdeptno);
END;
$BODY$
LANGUAGE plpgsql VOLATILE
```

- Click on the Execute button. This will create five stored procedures that we will use for tutorial purposes.

Run-time creation

The same operations performed in runtime look as follows:

[Delphi]

```
var
  script: TPgScript;
begin
  script := TPgScript.Create(nil);
  try
    script.Connection := con; // con is TPgConnection already set up
    script.SQL.Clear;
    script.SQL.Add('CREATE FUNCTION "Ten Most High-Paid Employees"()');
    script.SQL.Add('RETURNS SETOF Emp AS $$');
    script.SQL.Add('  SELECT * FROM emp ORDER BY emp.sal DESC LIMIT 10');
    script.SQL.Add('$$ LANGUAGE ''sql'';');
    script.SQL.Add('');
    script.SQL.Add('CREATE FUNCTION "GetEmpNumberInDept"(');
    script.SQL.Add('  IN pdeptno integer,');
    script.SQL.Add('  OUT pempnumb integer)');
    script.SQL.Add('RETURNS integer AS');
    script.SQL.Add('$BODY$');
    script.SQL.Add('BEGIN');
    script.SQL.Add('  pempnumb := (SELECT count(*) FROM emp WHERE deptno = p');
    script.SQL.Add('END;');
    script.SQL.Add('$BODY$');
    script.SQL.Add('  LANGUAGE plpgsql VOLATILE');
    script.Execute;
  finally
    script.Free;
  end;
end;
```


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

[C++Builder]

```
{
    TPgSQL* sql = new TPgSQL(NULL);
    try
    {
        sql->Connection = con; // con is TPgConnection already set up
        sql->SQL->Clear();
        sql->SQL->Add("CREATE TABLE dept (");
        sql->SQL->Add("    deptno serial not null,");
        sql->SQL->Add("    dname varchar(14),");
        sql->SQL->Add("    loc varchar(13),");
        sql->SQL->Add("    primary key (deptno)");
        sql->SQL->Add(");");
        sql->SQL->Add("CREATE TABLE emp (");
        sql->SQL->Add("    empno serial not null,");
        sql->SQL->Add("    ename varchar(10),");
        sql->SQL->Add("    job varchar(9),");
        sql->SQL->Add("    mgr integer,");
        sql->SQL->Add("    hiredate timestamp,");
        sql->SQL->Add("    sal real,");
        sql->SQL->Add("    comm real,");
        sql->SQL->Add("    deptno int references dept,");
        sql->SQL->Add("    primary key (empno)");
        sql->SQL->Add(");");
        sql->Execute();
    }
    finally
    {
        sql->Free();
    }
}
```

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

Additional information

Actually, there are lots of ways to create database objects on server. Any tool or component that is capable of running an SQL query, can be used to manage database objects. For example, TPgSQL suits fine for creating objects one by one, while TPgScript is designed for executing series of DDL/DML statements. For information on DDL statements syntax refer to the PostgreSQL documentation.

3.4 Deleting Data From Tables

This tutorial describes how to delete data from tables using the [TPgQuery](#) and [TPgTable](#) 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 (tutorial ["Connecting To PostgreSQL"](#)), 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 TPgQuery. Another way is to use the dataset functionality (the Delete method) of the TPgQuery and TPgTable 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 TPgQuery and TPgTable components allows deleting data without using DML statements. DML statements are generated by PgDAC components internally. The code below demonstrates using this method:

[Delphi]

```
var
  q: TPgQuery;
begin
  q := TPgQuery.Create(nil);
  try
    // con is TPgConnection 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]

```
{
    TPgQuery* q = new TPgQuery(NULL);
    try
    {
        // con is either TPgConnection 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]**

```
var
    q: TPgQuery;
begin
    q := TPgQuery.Create(nil);
    try
        // con is either TPgConnection 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]

```
{
    TPgQuery* q = new TPgQuery(NULL);
    try
    {
        // con is either TPgConnection 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();
}
}
```

DML Statements As Plain Text

[Delphi]

```
var
    q: TPgQuery;
begin
    q := TPgQuery.Create(nil);
    try
        // con is either TPgConnection 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]

```
{
    TPgQuery* q = new TPgQuery(NULL);
    try
    {
        // con is either TPgConnection 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".

3.5 Inserting Data Into Tables

This tutorial describes how to insert data into tables using the [TPgQuery](#) and [TPgTable](#) components.

1. [Requirements](#)
2. [General information](#)
3. [Design Time](#)
4. [Run Time](#)
 - o 4.1 [Using DataSet Functionality](#)
 - o 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 (tutorial ["Connecting To PostgreSQL"](#)) 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 [TPgQuery](#). Another way is to use the dataset functionality (the Insert, Append, and Post methods) of the [TPgQuery](#) and [TPgTable](#) components. We will discuss both ways.

The goal of this tutorial is to insert the following data into tables [dept](#) and [emp](#):

Table dept

deptno	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS

30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Table emp

empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	17-12-1980	800	NULL	20
7499	ALLEN	SALESMAN	7698	20-02-1981	1600	300	30
7521	WARD	SALESMAN	7698	22-02-1981	1250	500	30
7566	JONES	MANAGER	7839	02-04-1981	2975	NULL	20
7654	MARTIN	SALESMAN	7698	28-09-1981	1250	1400	30
7698	BLAKE	MANAGER	7839	01-05-1981	2850	NULL	30
7782	CLARK	MANAGER	7839	09-06-1981	2450	NULL	10
7788	SCOTT	ANALYST	7566	13-07-1987	3000	NULL	20
7839	KING	PRESIDENT	NULL	17-11-1981	5000	NULL	10
7844	TURNER	SALESMAN	7698	08-09-1981	1500	0	30
7876	ADAMS	CLERK	7788	13-07-1987	1100	NULL	20
7900	JAMES	CLERK	7698	03-12-1981	950	NULL	30
7902	FORD	ANALYST	7566	03-12-1981	3000	NULL	20
7934	MILLER	CLERK	7782	23-01-1982	1300	NULL	10

Design time

- Open the Tool palette and find the [TPgQuery](#) component in the PgDAC category.
- Double-click on the component. Note that a new object appears on the form. If this is the first time you create [TPgQuery](#) in this application, it is named PgQuery1. Note that the PgQuery1.Connection property is already set to an existent (on the form) connection.

- Double-click on the PgQuery1 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 [TPgQuery](#) and [TPgTable](#) components allow inserting data not using DML statements. DML statements are generated by PgDAC 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]

```
var
  q: TPgQuery;
begin
  q := TPgQuery.Create(nil);
  try
    q.Connection := con; // con is TPgConnection already set up
    q.SQL.Text := 'SELECT * FROM dept';
    q.Open;
    q.Append;
    q.FieldName('deptno').AsInteger := 10;
    q.FieldName('dname').AsString := 'ACCOUNTING';
    q.FieldName('loc').AsString := 'NEW YORK';
    q.Post;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TPgQuery* q = new TPgQuery(NULL);
  try
  {
    q->Connection = con; // con is TPgConnection already set up
    q->SQL->Text = "SELECT * FROM dept";
    q->Open();
    q->Append();
    q->FieldName("deptno")->AsInteger = 10;
    q->FieldName("dname")->AsString = "ACCOUNTING";
    q->FieldName("loc")->AsString = "NEW YORK";
    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]

```

var
    q: TPgQuery;
begin
    q := TPgQuery.Create(nil);
    try
        q.Connection := con; // con is TPgConnection set up
        q.SQL.Clear;
        q.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (:deptno, :dname, :loc)');
        q.ParamByName('deptno').AsInteger := 10;
        q.ParamByName('dname').AsString := 'ACCOUNTING';
        q.ParamByName('loc').AsString := 'NEW YORK';
        q.Execute;
    finally
        q.Free;
    end;
end;

```

[C++Builder]

```

{
    TPgQuery* q = new TPgQuery(NULL);
    try
    {
        q->Connection = con; // con is TPgConnection already set up
        q->SQL->Clear();
        q->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (:deptno, :dname, :loc)");
        q->ParamByName("deptno")->AsInteger = 10;
        q->ParamByName("dname")->AsString = "ACCOUNTING";
        q->ParamByName("loc")->AsString = "NEW YORK";
        q->Execute();
    }
    finally
    {
        q->Free();
    }
}

```

DML Statements As Plain Text

[Delphi]

```
var
  q: TPgQuery;
begin
  q := TPgQuery.Create(nil);
  try
    q.Connection := con; // con is TPgConnection already set up
    q.SQL.Clear;
    q.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (10, 'ACCOUNTING'
    q.Execute;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TPgQuery* q = new TPgQuery(NULL);
  try
  {
    q->Connection = con; // con is TPgConnection already set up
    q->SQL->Clear();
    q->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (10, 'ACCOUNTING
    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, [TPgLoader](#) is the fastest way to insert data, [TPgScript](#) 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 manipulation logic is defined on the server. You can find more about using stored procedures in the tutorial "Stored Procedures".

3.6 Retrieving Data

1. [Requirements](#)
2. [General Information](#)
3. [TPgQuery](#)
4. [TPgTable](#)
5. [Additional information](#)

Requirements

This walkthrough supposes that you know how to connect to server (tutorial ["Connecting To SQL 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. PgDAC provides several components that can be used for data retrieving, such as TPgQuery and TPgTable. We will discuss data retrieving using these components.

The goal of this tutorial is to retrieve data from a table [dept](#).

TPgQuery

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

[Delphi]

```
var
  q: TPgQuery;
begin
  q := TPgQuery.Create(nil);
  try
    // con is TPgConnection 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;
end;
```

[C++Builder]

```
{
  TPgQuery* q = new TPgQuery(NULL);
  try
  {
    // con is TPgConnection 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();  
}  
}
```

TPgTable

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

[Delphi]

```
var  
    tbl: TPgTable;  
begin  
    tbl := TPgTable.Create(nil);  
    try  
        // con is TPgConnection 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;  
end;
```

[C++Builder]

```
{  
    TPgTable* tbl = new TPgTable(NULL);  
    try  
    {  
        // con is TPgConnection 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();  
    }  
}
```

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".

3.7 Modifying Data

This tutorial describes how to modify data into tables using the [TPgQuery](#) and [TPgTable](#) components.

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

Requirements

This walkthrough supposes that you know how to connect to server (tutorial ["Connecting To PostgreSQL"](#)), 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 TPgQuery. Another way is to use the dataset functionality (the Edit and Post methods) of the TPgQuery and TPgTable components. We will discuss both ways. The goal of this tutorial is to modify the following record of the table [dept](#):

10	ACCOUNTING	NEW YORK
----	------------	----------

to make it look as follows:

10	RESEARCH	LOS ANGELES
----	----------	-------------

Using DataSet Functionality

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

[Delphi]

```
var  
q: TPgQuery;
```

```
begin
  q := TPgQuery.Create(nil);
  try
    // con is TPGConnection 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([10]);
    // modify record
    q.Edit;
    q.FieldName('dname').AsString := 'RESEARCH';
    q.FieldName('loc').AsString := 'LOS ANGELES';
    q.Post;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```
{
  TPgQuery* q = new TPgQuery(NULL);
  try
  {
    // con is TPGConnection 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(ARRAYOFCOCONST((10)));
    // modify record
    q->Edit();
    q->FieldName("dname")->AsString = "RESEARCH";
    q->FieldName("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]

```
var
```

```

q: TPgQuery;
begin
  q := TPgQuery.Create(nil);
  try
    // con is TPgConnection 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');
    // set parameters
    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]

```

{
  TPgQuery* q = new TPgQuery(NULL);
  try
  {
    // con is TPgConnection 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");
    // set parameters
    q->ParamByName("deptno")->AsInteger = 10;
    q->ParamByName("dname")->AsString = "RESEARCH";
    q->ParamByName("loc")->AsString = "LOS ANGELES";
    // execute query
    q->Execute();
  }
  __finally
  {
    q->Free();
  }
}

```

DML Statements As Plain Text

[Delphi]

```

var
  q: TPgQuery;
begin
  q := TPgQuery.Create(nil);
  try
    // con is TPgConnection 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' W
// execute query
q.Execute;
finally
q.Free;
end;
end;
```

[C++Builder]

```
{
TPgQuery* q = new TPgQuery(NULL);
try
{
// con is TPgConnection 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' WHE
// 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".

3.8 Demo Projects

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

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

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

Where are the PgDAC demo projects located?

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

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

The structure of the demo project directory depends on the IDE version you are using. For most new IDEs the structure will be as follows.

Demos

```
| -PgDacDemo [The main PgDAC demo project]
| -TechnologySpecific
|   | - SecureBridge [A component and a demo for integration with
the SecureBridge library]
|   | -ThirdParty
|   | - [A collection of demo projects on integration with third-
party components]
|   | -Miscellaneous
|   | - [Some other demo projects on design technologies]
```

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

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

Instructions for using the PgDAC demo projects

To explore a PgDAC demo project,

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

The included sample applications are fully functional. To use the demos, you have to first set up a connection to PostgreSQL. You can do so by clicking on the "Connect" button. Many demos may also use some database objects. If so, they will have two object

manipulation buttons, "Create" and "Drop". If your demo requires additional objects, click "Create" to create the necessary database objects. When you are done with a demo, click "Drop" to remove all the objects used for the demo from your database.

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

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

Demo project descriptions

PgDacDemo

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

Working with components

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

General demos

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

PostgreSQL-specific demos

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

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

Working with Components

Name	Description
Alerter	Uses TPgAlerter to send notifications between connections.
ConnectDialog	Demonstrates how to customize the PgDAC connect dialog . Changes the standard PgDAC connect dialog to two custom connect dialogs. The first customized sample dialog is inherited from the TForm class, and the second one is inherited from the default PgDAC connect dialog class.
CRDBGrid	Demonstrates how to work with the TCRDBGrid component. Shows off the main TCRDBGrid features, like filtering, searching, stretching, using compound headers, and more.
Dump	Demonstrates how to backup data from tables with the TPgDump

	component. Shows how to use scripts created during back up to restore table data. This demo lets you back up a table either by specifying the table name or by writing a SELECT query.
Loader	Uses the TPgLoader component to quickly load data into a server table. This demo also compares the two TPgLoader data loading handlers: GetColumnData and PutData .
Query	Demonstrates working with TPgQuery , which is one of the most useful PgDAC components. Includes many TPgQuery usage scenarios. Demonstrates how to execute queries in both standard and NonBlocking mode and 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 PgDAC.
Sql	Uses TPgSQL to execute SQL statements. Demonstrates how to work in a separate thread, in standard mode, in NonBlocking mode, and how to break long-duration query execution.
StoredProc	Uses TPgStoredProc to access an editable recordset from a PostgreSQL stored procedure in the client application.
Table	Demonstrates how to use TPgTable to work with data from a single table on the server without writing any SQL queries manually. Performs server-side data sorting and filtering and retrieves results for browsing and editing.
UpdateSQL	Demonstrates using the TPgUpdateSQL component to customize update commands. Lets you optionally use T:Devart.PgDac.TPgCommand and TPgQuery objects for carrying out insert, delete, query, and update commands.
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

Name	Description
CachedUpdates	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.
FilterAndIndex	Demonstrates PgDAC's local storage functionality. This sample shows how to perform local filtering, sorting and locating by multiple fields, including by calculated and lookup fields.

MasterDetail	Uses PgDAC functionality to work with master/detail relationships . This sample shows how to use local master/detail functionality. Demonstrates different kinds of master/detail linking, including linking by SQL, by simple fields, and by calculated fields.
Lock	Demonstrates the recommended approach for managing transactions with the TPgConnection component. The TPgConnection interface provides a wrapper for PostgreSQL server commands like START TRANSACTION, COMMIT, ROLLBACK.

PostgreSQL-specific Demos

Name	Description
Pictures	Uses PgDAC functionality to work with graphics. The sample demonstrates how to retrieve binary data from PgSQL server database and display it on visual components. Sample also shows how to load and save pictures to files and to the database.
Text	Uses PgDAC functionality to work with text. The sample demonstrates how to retrieve text data from SQL Server database and display it on visual components. Sample also shows how to load and save text to files and to the database.

Supplementary Demo Projects

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

Location	Name	Description
ThirdParty	FastReport	Demonstrates how PgDAC can be used with FastReport components. This project consists of two parts. The first part is several packages that integrate PgDAC components into the FastReport editor. The second part is a demo application that lets you design and preview reports with PgDAC technology in the FastReport editor.
Technology Specific	SecureBridge	The demo project demonstrates how to integrate the SecureBridge components with PgDAC to ensure secure connection to PostgreSQL server through an SSH tunnel and SSL. This demo consists of three parts. The first part is a package that contains the TPgSSHIOHandler and TPgSSLIOHandler component. These components provide integration with the

		SecureBridge library. The second part is two sample projects that demonstrate how to connect to PostgreSQL server through an SSH server and through SSL, connect to the SSH server with SecureBridge by password or by public key, generate reliable random numbers, enable local port forwarding. For more information see the <i>Readme.html</i> file in the demo directory.
Miscellaneous	Dll	Demonstrates creating and loading DLLs for PgDAC-based projects. This demo project consists of two parts - an Pg_Dll project that creates a DLL of a form that sends a query to the server and displays its results, and an Pg_Exe project that can be executed to display a form for loading and running this DLL. Allows you to build a dll for one PgDAC-based project and load and test it from a separate application.
	FailOver	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.
PgDacDemo	PgDacDemo	<i>[Win32 version of the main PgDAC demo project - see above]</i>

3.9 Deployment

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

Deploying Windows applications built without run-time packages

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

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

Trial Limitation Warning

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

dacXX.bpl	always
pgdacXX.bpl	always

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

rtlXX.bpl	always
dbrtlXX.bpl	always
vcldbXXX.bpl	always

Deploying Windows applications built with run-time packages

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

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

dacXX.bpl	always
pgdacXX.bpl	always
dacvclXX.bpl	if your application uses the PgDacVcl unit
pgdacvclXX.bpl	if your application uses the PgDacVcl unit
crcontrolsXX.bpl	if your application uses the CRDBGrid component

4 Using PgDAC

This section describes basics of using PostgreSQL Data Access Components

- [Updating Data with PgDAC Dataset Components](#)
- [Master/Detail Relationships](#)
- [Automatic Key Field Value Generation](#)
- [Data Type Mapping](#)
- [Data Encryption](#)
- [Working in an Unstable Network](#)
- [Secure Connections](#)
- [Disconnected Mode](#)
- [Increasing Performance](#)
- [Macros](#)
- [DataSet Manager](#)
- [TPgLoader Component](#)

- [Large Objects](#)
- [REFCURSOR Data Type](#)
- [National and Unicode Characters](#)
- [Connection Pooling](#)
- [DBMonitor](#)
- Writing GUI Applications with PgDAC
- [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 PgDAC Dataset Components

PgDAC dataset components which descend from [TCustomDADataset](#) provide different ways for reflecting local changes on the server.

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

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

See Also

- [TPgQuery](#)
- [TPgStoredProc](#)

- [TPgTable](#)
- [TPgUpdateSQL](#)

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 PgDAC implements this feature.

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

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

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

It is necessary to display and edit these tables.

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

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

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.

```
procedure TForm1.Form1Create(Sender: TObject);
var
  Master, Detail: TPgQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TPgQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  // create detail dataset
  Detail := TPgQuery.Create(Self);
  Detail.SQL.Text := 'SELECT * FROM Employee';
  // setup MD
  Detail.MasterFields := '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](#)

4.3 Automatic Key Field Value Generation

When editing a dataset it is often convenient to generate key field(s) values automatically instead of filling them manually. In the most common way application developer generates

primary key value basing it on a previously created sequence. There are three ways of doing it.

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

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

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

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

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

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

```
...
PgQuery.SQL.Text := 'SELECT DepNo, DepName, Location FROM Department';
PgQuery.KeyFields := 'DepNo'; // key field
PgQuery.KeySequence := 'seq_deptno'; // sequence that will generate values
...
```

See Also

- [KeySequence](#)

4.4 Data Type Mapping

Overview

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

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

Data Type Mapping Rules

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

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

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

And Data Type Mapping should be used so that:

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

The above in the form of a table:

PostgreSQL data type	Default Delphi field type	Destination Delphi field type
numeric(4,0)	ftFloat	ftSmallint
numeric(10,0)	ftFloat	ftInteger
numeric(15,0)	ftFloat	ftLargeint
numeric(5,2)	ftFloat	ftFloat
numeric(10,4)	ftFloat	ftBCD
numeric(15,6)	ftFloat	ftFMTBCD

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

```
var
  DBType: word;
  MinPrecision: Integer;
  MaxPrecision: Integer;
  MinScale: Integer;
  MaxScale: Integer;
  FieldType: TfieldType;
begin
```

```

DBType      := pgNumeric;
MinPrecision := 0;
MaxPrecision := 4;
MinScale    := 0;
MaxScale    := 0;
FieldType   := ftSmallint;
PgConnection.DataTypeMap.AddDBTypeRule(DBType, MinPrecision, MaxPrecision,
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:

```

// clear existing rules
PgConnection.DataTypeMap.Clear;
// rule for numeric(4,0)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, 4, 0, 0, ftSmallint);
// rule for numeric(10,0)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 5, 10, 0, 0, ftInteger);
// rule for numeric(15,0)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 11, r1Any, 0, 0, ftLargeInteger);
// rule for numeric(5,2)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, 9, 1, r1Any, ftFloat);
// rule for numeric(10,4)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 10, r1Any, 1, 4, ftBCD);
// rule for numeric(15,6)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 10, r1Any, 5, r1Any, ftFMTBCD);

```

Rules order

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

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

```

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

```

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

PostgreSQL data type	Default Delphi field type	Destination field type
NUMBER(5,2)	ftFloat	ftFloat
NUMBER(10,4)	ftFloat	ftBCD
NUMBER(15,6)	ftFloat	ftFMTBCD

If rules are set in the following way:

```
PgConnection.DataTypeMap.Clear;
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, 9, r1Any, r1Any, ft
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, r1Any, 0, 4, ft
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, r1Any, 0, r1Any, ft
```

it will lead to the following result:

PostgreSQL data type	Delphi field type
NUMBER(5,2)	ftFloat
NUMBER(10,4)	ftBCD
NUMBER(15,6)	ftFMTBCD

But if rules are set in the following way:

```
PgConnection.DataTypeMap.Clear;
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, r1Any, 0, r1Any, ft
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, r1Any, 0, 4, ft
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, 9, r1Any, r1Any, ft
```

it will lead to the following result:

PostgreSQL data type	Delphi field type
NUMBER(5,2)	ftFMTBCD
NUMBER(10,4)	ftFMTBCD
NUMBER(15,6)	ftFMTBCD

This happens because the rule

```
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, r1Any, 0, r1Any, ft
```

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

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

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

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

Defining rules for Connection and Dataset

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

For example, such table is created in SQL Server:

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

It is exactly known that the birth_dttm field contains birth day, and this field should be ftDate in Delphi, and not ftDateTime. If such rule is set:

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

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

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

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

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

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

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

Rules for a particular field

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

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

```
CREATE TABLE item
(
  id integer          NOT NULL,
  name character(50) NOT NULL,
  guid character(38),
  CONSTRAINT pk_item PRIMARY KEY (id)
```

The **guid** field contains a unique identifier. For convenient work, this identifier is expected to be mapped to the TGUIDField type in Delphi. But there is one problem, if to set the rule like this:

```
PgQuery.DataTypeMap.Clear;
PgQuery.DataTypeMap.AddDBTypeRule(pgCharacter, ftGuid);
```

then both **name** and **guid** fields will have the ftGuid type in Delphi, that does not correspond to what was planned. In this case, the only way is to use Data Type Mapping for a particular field:

```
PgQuery.DataTypeMap.Clear;
PgQuery.DataTypeMap.AddFieldRule('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:

Database value	Destination field type	Error
'text value'	ftInteger	String cannot be converted to Integer
1000000	ftSmallint	Value is out of range
15,1	ftInteger	Cannot convert float to integer

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

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

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

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

Therefore ignoring of conversion errors should be used only if the conversion results are expected.

4.5 Data Encryption

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

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

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

If data is stored without additional information, it is impossible to determine whether the data is encrypted or not. In this case, only the encrypted data should be stored in the column, otherwise, there will be confusion because of the inability to distinguish the nature of the data. Also in this way, the similar source data will be equivalent in the encrypted form, that is not

good from the point of view of the information protection. The advantage of this method is the size of the initial data equal to the size of the encrypted data.

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:

```
CREATE TABLE emp (
empno integer,
ename character(2000),
hiredate character(200),
sal character(200),
foto bytea,
CONSTRAINT pk_emp PRIMARY KEY (empno)
```



```
);
```

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.

```
PgQuery.SQL.Text := 'SELECT * FROM emp';
PgQuery.Encryption.Encryptor := PgEncryptor;
PgQuery.Encryption.Fields := 'ename, hiredate, sal, foto';
PgEncryptor.Password := '11111';
PgQuery.DataTypeMap.AddFieldNamedRule ('ename', ftString);
PgQuery.DataTypeMap.AddFieldNamedRule ('hiredate', ftDateTime);
PgQuery.DataTypeMap.AddFieldNamedRule ('sal', ftFloat);
PgQuery.Open;
```

4.6 Working in an Unstable Network

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

```
TCustomDACConnection.Options.LocalFailover = True
TCustomDACConnection.Options.DisconnectedMode = True
TDataSet.CachedUpdates = True
TCustomDADataset.FetchAll = True
TCustomDADataset.Options.LocalMasterDetail = True
AutoCommit = True
```

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

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

It is not recommended to prepare queries explicitly. Use the [CachedUpdates](#) mode for DataSet data editing. Use the [TCustomDADataset.Options.UpdateBatchSize](#) property to reduce the number of requests to the server.

If a connection breaks, a fatal error occurs, and the [OnConnectionLost](#) event will be raised if the following conditions are fulfilled:

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

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

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

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

The using of [Pooling](#) in Disconnected Mode allows to speed up most of the operations because of connecting duration reducing.

See Also

- [FailOver demo](#)
- [Working with Disconnected Mode](#)
- [TCustomDAConnection.Options](#)
- [TCustomDAConnection.Pooling](#)

4.7 Secure Connections

Session security depends on several factors, including whether the connection to the host is a trusted connection. If it is not, confidential information can not be transmitted through this connection.

PgDAC supports two different ways to increase connection security. They are SSH and SSL. Both SSH and SSL can be implemented with SecureBridge components.

Devart SecureBridge is a non visual component library that provides functionality for SSH tunneling and SSL connections. Usage of SecureBridge is the handiest and fastest way to ensure protected connection to PostgreSQL server. You can read more about SecureBridge at the [SecureBridge home page](#). The detailed step-by-step instructions on setting up SecureBridge you will find in the SecureBridge documentation.

To create an SSL connection with PostgreSQL one of the following modes can be used: `smRequire`, `smPrefer`, or `smAllow`.

For example:

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

[SSLMode](#) determines whether or with what priority an SSL connection will be negotiated with the server.

- **smDisable** will attempt only an unencrypted SSL connection, then if that fails, raising exception.
- **smRequire** will try only an SSL connection, then if that fails, raising exception.

- **smAllow** will negotiate, trying first a non-SSL connection, then if that fails, trying an SSL connection.
- **smPrefer** will negotiate, trying first an SSL connection, then if that fails, trying a regular non-SSL connection.

1. SSL using SecureBridge

SecureBridge also allows you to embed functionality of an SSL client into your application. The following sequence of steps describes how to protect your connection to PostgreSQL server with SSL using SecureBridge:

- Place the TCRSSLIOHandler component onto the form.
- Select a storage object in the Storage property. More information about storage setup you will find in the SSL client setup topic of SecureBridge help.
- Specify the server certificate in the CACertName property.
- Specify the client certificate in the CertName property.
- Place the TPgConnection component onto the form and setup it to connect to the PostgreSQL server.
- Assign the TCRSSLIOHandler object to the IOHandler property of TPgConnection.
- Connect to PostgreSQL server by setting TPgConnection.Connected to True.

2. SSL using OpenSSL library

The description of the SSL connection features without IOHandler usage:

The following options should 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: For using SSL protocol ssleay32.dll and libeay32.dll files are needed.

The detailed description of SSL connection you can find in PostgreSQL Documentation:

- Server settings: "Secure TCP/IP Connections with SSL"
- Client settings: "SSL Support"

3. SSH using SecureBridge

SecureBridge allows you to embed functionality of an SSH client into your application. The following sequence of steps describes how to protect your connection to PostgreSQL server through an SSH tunnel with SecureBridge:

- configure your SSH server like described in the server documentation, or use SecureBridge to make your own SSH server. SecureBridge includes a demo project that implements functionality of an SSH server;
- place the TScSSHClient component of SecureBridge onto your form;
- setup TScSSHClient (assign host name, SSH server port, user name, password) to connect to the SSH server and check the connection;
- place the TCRSSHIOHandler component onto your form. This component is included into PgDAC as a demo project;
- place the [TPgConnection](#) component onto your form, and link to its IOHandler property the instance of TCRSSHIOHandler added on the previous step;
- setup [TPgConnection](#) to connect to PostgreSQL server and check the connection.

Now you have an encrypted connection between PostgreSQL server and your application.

4.8 Disconnected Mode

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

[TCustomDAConnection.Pooling](#).

To enable disconnected mode set [TCustomDAConnection.Options.DisconnectedMode](#) to True.

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

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

<code>TDataSet.CachedUpdates = True</code>
<code>TCustomDADataSet.FetchAll = True</code>
<code>TCustomDADataSet.Options.LocalMasterDetail = True</code>

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

fetches 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

- [TCustomDACConnection.Options](#)
- [FetchAll](#)
- [Devart.PgDac.TPgQuery.LockMode](#)
- [TCustomDACConnection.Pooling](#)
- [TCustomDACConnection.Connect](#)
- [TCustomDACConnection.Disconnect](#)
- [Working in unstable network](#)

4.9 Batch Operations

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

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

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

Batch_Test table generating scripts

```
CREATE TABLE BATCH_TEST
(
  ID      INTEGER,
  F_INTEGER INTEGER,
  F_FLOAT DOUBLE PRECISION,
  F_STRING VARCHAR(250),
```

```

F_DATE DATE,
CONSTRAINT PK_BATCH_TEST PRIMARY KEY (ID)
)

```

Batch operations execution

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

```
INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, :F_FLOAT, :F_STRING, :F_DATE)
```

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

Parameters				
:ID	:F_INTEGER	:F_FLOAT	:F_STRING	:F_DATE
1	100	2.5	'String Value 1'	01.09.2015

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

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

Parameters				
:ID	:F_INTEGER	:F_FLOAT	:F_STRING	:F_DATE
1	100	2.5	'String Value 1'	01.09.2015
2	200	3.15	'String Value 2'	01.01.2000
3	300	5.08	'String Value 3'	09.09.2010
4	400	7.5343	'String Value 4'	10.10.2015
5	500	0.4555	'String Value 5'	01.09.2015

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:

```

var
  i: Integer;
begin
  // describe the SQL query
  PgQuery1.SQL.Text := 'INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, :F_FLOAT, :F_STRING, :F_DATE)';
  // define the parameter types passed to the query :
  PgQuery1.Params[0].DataType := ftInteger;
  PgQuery1.Params[1].DataType := ftInteger;
  PgQuery1.Params[2].DataType := ftFloat;
  PgQuery1.Params[3].DataType := ftString;
  PgQuery1.Params[4].DataType := ftDateTime;
  // specify the array dimension:
  PgQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:

```

```

for i := 0 to PgQuery1.Params.ValueCount - 1 do begin
  PgQuery1.Params[0][i].AsInteger := i + 1;
  PgQuery1.Params[1][i].AsInteger := i + 2000 + 1;
  PgQuery1.Params[2][i].AsFloat := (i + 1) / 12;
  PgQuery1.Params[3][i].AsString := 'Values ' + IntToStr(i + 1);
  PgQuery1.Params[4][i].AsDateTime := Now;
end;
// insert 1000 rows into the BATCH_TEST table
PgQuery1.Execute(1000);
end;

```

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

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

All 1000 rows at a time:

```
PgQuery1.Execute(1000);
```

2×500 rows:

```

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

```

500 rows, then 300, and finally 200:

```

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

```

Batch UPDATE operation sample

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

```

var
  i: Integer;
begin
  // describe the SQL query
  PgQuery1.SQL.Text := 'UPDATE BATCH_TEST SET F_INTEGER=:F_INTEGER, F_FLOAT=:F_FLOAT';
  // define parameter types passed to the query:
  PgQuery1.Params[0].DataType := ftInteger;
  PgQuery1.Params[1].DataType := ftFloat;
  PgQuery1.Params[2].DataType := ftString;
  PgQuery1.Params[3].DataType := ftDateTime;
  PgQuery1.Params[4].DataType := ftInteger;
  // specify the array dimension:
  PgQuery1.Params.ValueCount := 1000;

```

```

// populate the array with parameter values:
for i := 0 to 1000 - 1 do begin
    PgQuery1.Params[0][i].AsInteger := i - 2000 + 1;
    PgQuery1.Params[1][i].AsFloat := (i + 1) / 100;
    PgQuery1.Params[2][i].AsString := 'New Values ' + IntToStr(i + 1);
    PgQuery1.Params[3][i].AsDateTime := Now;
    PgQuery1.Params[4][i].AsInteger := i + 1;
end;
// update 1000 rows in the BATCH_TEST table
PgQuery1.Execute(1000);
end;

```

Batch DELETE operation sample

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

```

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

```

Performance comparison

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

Operation Type	25 000 records	
	Standard Operation (sec.)	Batch Operation (sec.)
Insert	346.7	1.69
Update	334.4	4.59
Delete	373.7	2.05
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:

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

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

4.10 Increasing Performance

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

Connect

If your application performs Connect/Disconnect operations frequently, additional performance can be gained using pooling mode (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 [TDADatasetOptions.AutoPrepare](#) property to increase performance. For example, it can be enabled for Detail dataset in Master/Detail relationship or for update objects in TDAUpdateSQL. The performance gain achieved this way can be anywhere from several percent to several times, depending on the situation.

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

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

Fetch

In some situations you can increase performance a bit by using [TCustomDADataset.Options.CompressBlobMode](#).

You can also tweak your application performance by using the following properties of [TCustomDADataset](#) descendants:

- [FetchRows](#)
- [Options.FlatBuffers](#)
- [Options.LongStrings](#)
- [UniDirectional](#)

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

Navigate

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

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

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

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

Update

If your application updates datasets in the CachedUpdates mode, then setting the [TCustomDADataset.Options.UpdateBatchSize](#) option to more than 1 can improve performance several hundred times more by reducing the number of requests to the server. You can also increase the data sending performance a bit (several percents) by using `Dataset.UpdateObject.ModifyObject`, `Dataset.UpdateObject`, etc. Little additional performance improvement can be reached by setting the [AutoPrepare](#) property for these objects.

Insert

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

4.11 Macros

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

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

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

```
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;
```

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

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

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

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

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

See Also

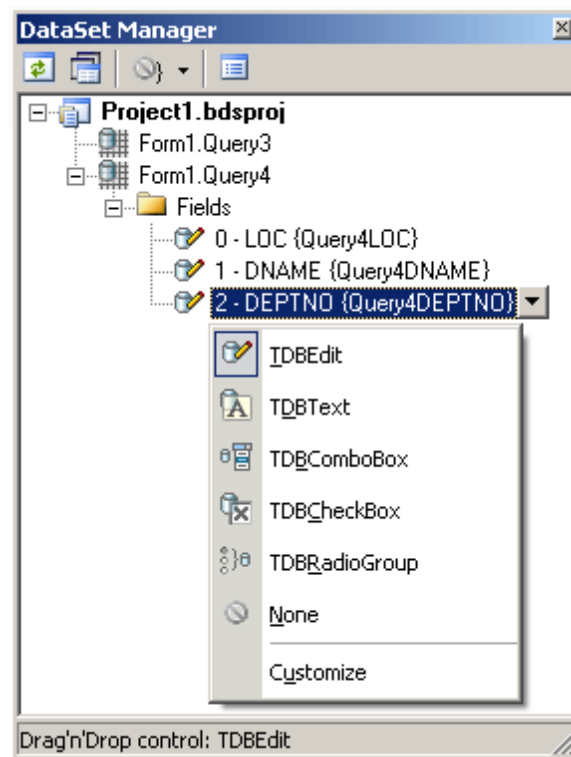
- [TMacro](#)
- [TCustomDADataset.MacroByName](#)
- [TCustomDADataset.Macros](#)

4.12 DataSet Manager

DataSet Manager window

The DataSet Manager window displays the datasets in your project. You can use the DataSet

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



Using the DataSet Manager window, you can:

- Create forms that display data by dragging items from the DataSet Manager window onto forms.
- Customize the list of controls available for each data type in the DataSet Manager window.
- Choose which control should be created when dragging an item onto a form in your Windows application.
- Create and delete TField objects in the DataSets of your project.

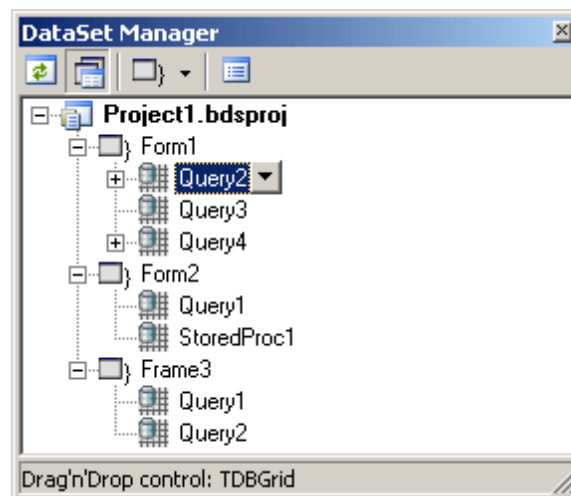
Opening the DataSet Manager window

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

Observing project DataSets in the DataSet Manager Window

By default DataSet Manager shows DataSets of currently open forms. It can also extract DataSets from all forms in the project. To use this, click *Extract DataSets from all forms in project* button. This settings is remembered. Note, that using this mode can slow down opening of the large projects with plenty of forms and DataSets. Opening of such projects can be very slow in Borland Delphi 2005 and Borland Developer Studio 2006 and can take up to several tens of minutes.

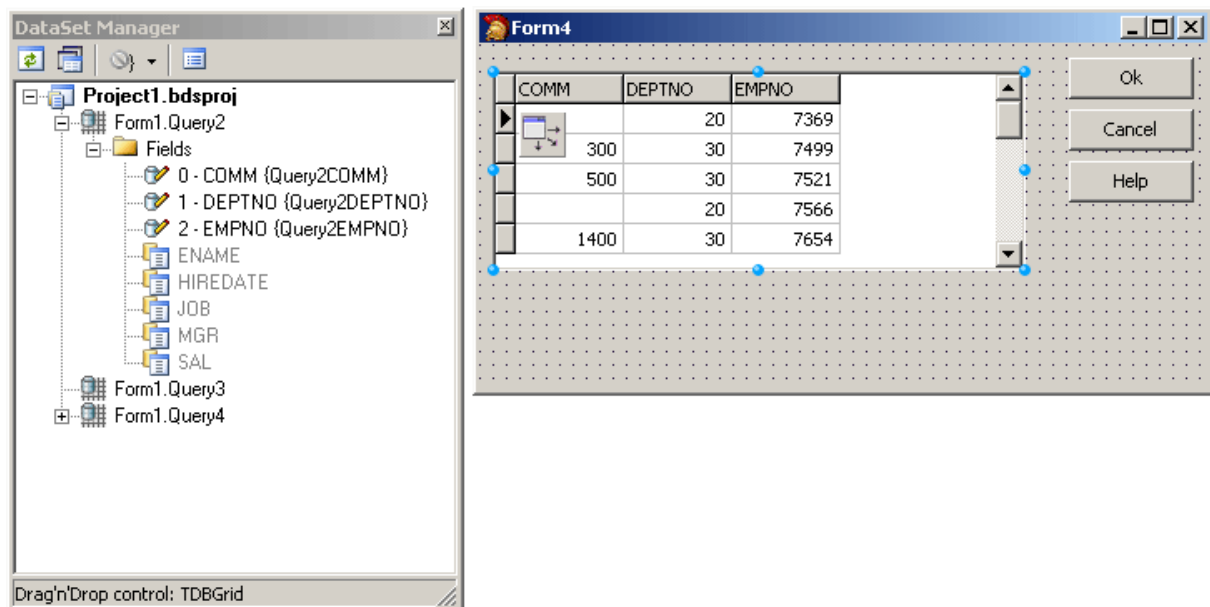
DataSets can be grouped by form or connection. To change DataSet grouping click the *Grouping mode* button or click a down. You can also change grouping mode by selecting required mode from the DataSet Manager window popup menu.



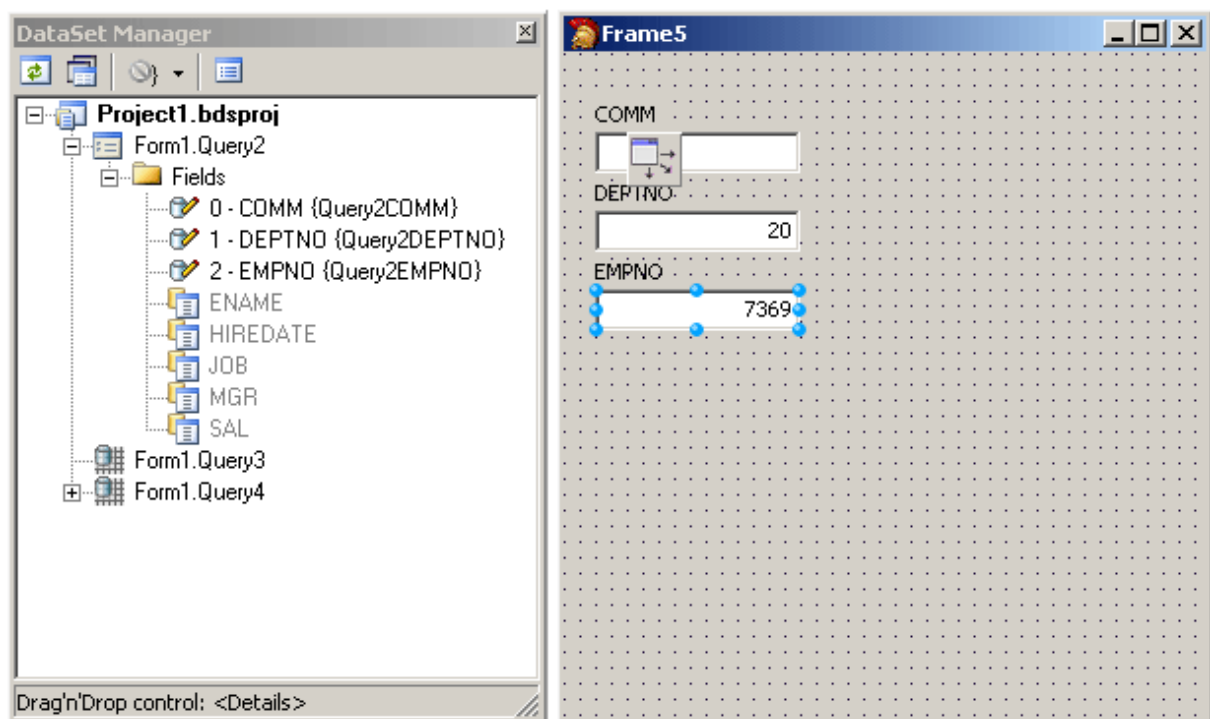
Creating Data-bound Controls

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

To use grid layout drag the dataset node on the form. By default TDataSource and TDBGrid components are created. You can choose the control to be created prior to dragging by selecting an item in the DataSet Manager window and choosing the control from the item's drop-down control list.

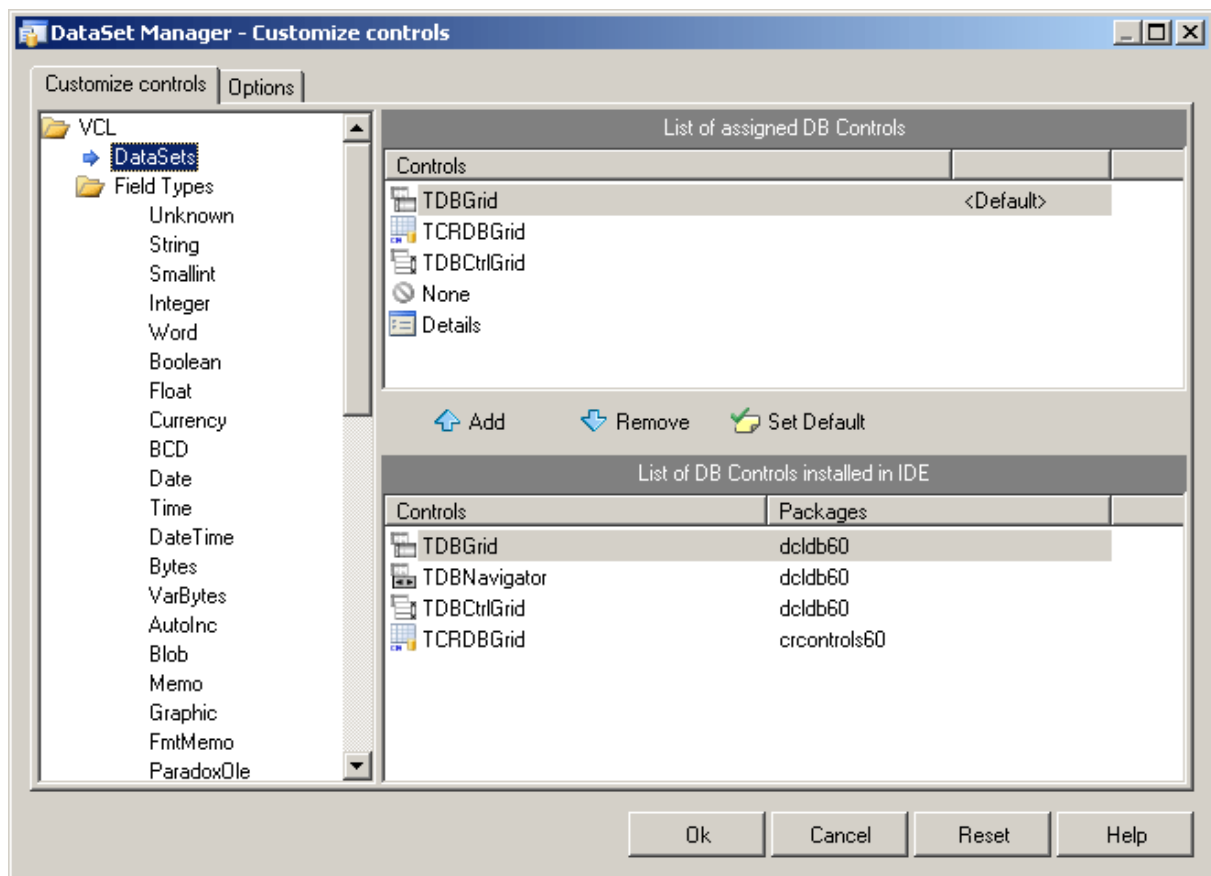


To use Details layout choose Details from the DataSet node drop-down control list in the DataSet Manager window. Then select required controls in the drop-down control list for each DataSet field. DataSet fields must be created. After setting required options you can drag the DataSet to the form from the DataSet wizard. DataSet Manager will create TDataSource component, and a component and a label for each field.



Adding custom controls to the DataSet Manager window

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



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

Working with TField objects

DataSet Manager allows you to create and remove TField objects. DataSet must be active to work with its fields in the DataSet Manager. You can add fields, based on the database table columns, create new fields, remove fields, use drag-n-drop to change fields order.

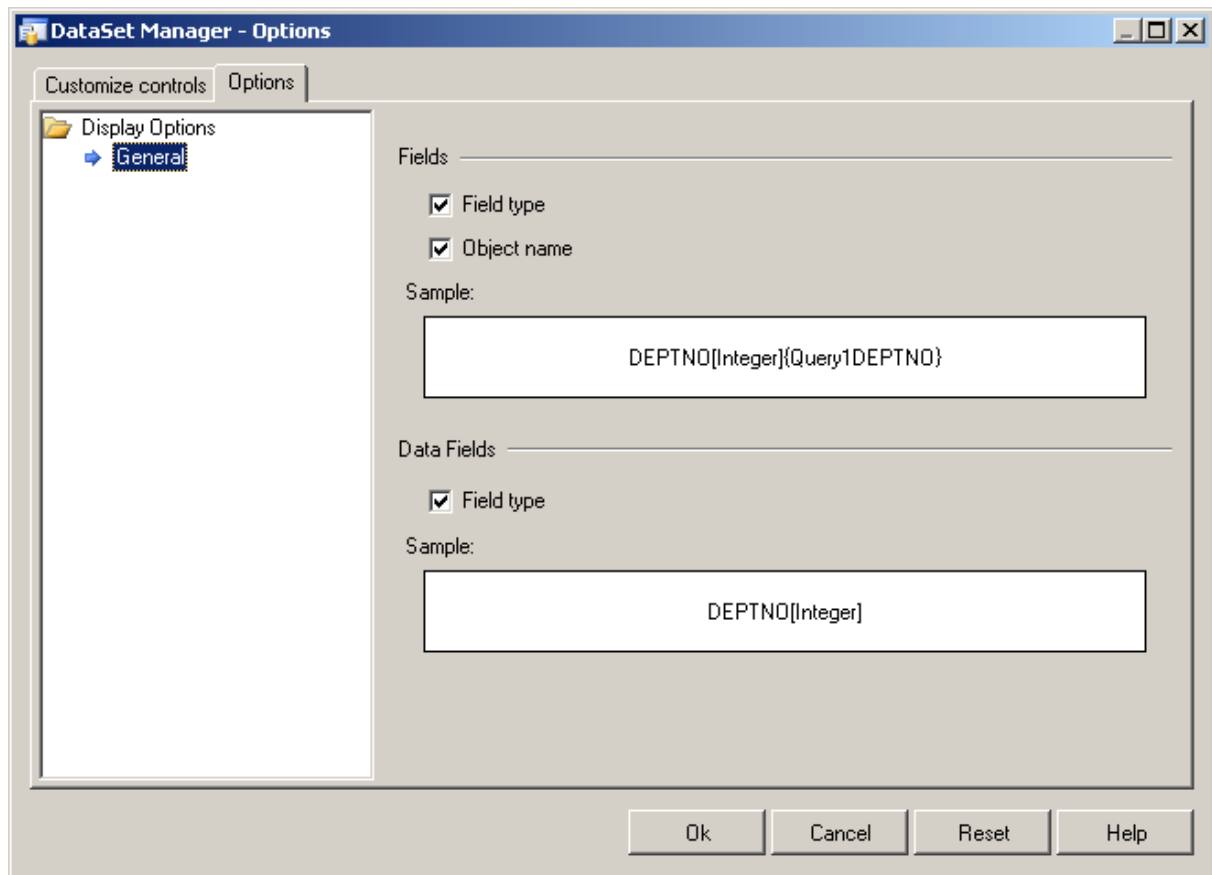
To create a field based on the database table column right-click the Fields node and select *Create Field* from the popup menu or press <Insert>. Note that after you add at least one field

manually, DataSet fields corresponding to data fields will not be generated automatically when you drag the DataSet on the form, and you can not drag such fields on the form. To add all available fields right-click the Fields node and select *Add all fields* from the popup menu.

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

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

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



You can chose 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.

4.13 TPgLoader Component

There are cases when you need to put large amount of data to a PostgreSQL database. Of course, you may construct INSERT SQL statement and execute it with the [TPgSQL](#) component. But it takes a lot of time. PostgreSQL provides COPY FROM STDIN command that allows to load data much faster. PgDAC simplifies using this command by the [TPgLoader](#) component.

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

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

To write your own loader you should:

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

See Also

- [TPgLoader](#)
- [TPgLoaderColumn](#)

4.14 Large Objects

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

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

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

Creating a new object:

```
var
  LargeObject: TPgLargeObject;
  AData: array [1..10] of byte;
  ...
PgConnection.StartTransaction;
```

```
LargeObject := TPgLargeObject.Create(PgConnection);
try
    LargeObject.CreateObject;
    LargeObject.Write(0, 10, &AData);
    LargeObject.WriteBlob;
    LargeObject.CloseObject;
finally
    LargeObject.Free;
end;
PgConnection.Commit;
```

Reading an existent object:

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

Note that manipulations with large objects require a transaction. So [StartTransaction](#) is called in the example.

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

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

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

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

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

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

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

4.15 REFCURSOR Data Type

Rather than executing a whole query at once, it is possible to set up a cursor that encapsulates the query, and then read the query result a few rows at a time. One reason for doing this is to avoid memory overrun when the result contains a large number of rows. PL/pgSQL functions can return cursors to the caller. This is useful to return multiple rows or columns, especially with very large result sets. To do this, the function opens the cursor and returns the cursor name to the caller (or simply opens the cursor using a portal name specified by or otherwise known to the caller). The caller can then fetch rows from the cursor. The cursor can be closed by the caller, or it will be closed automatically when the transaction closes.

PgDAC supports reading cursors returned from stored procedures. The [TPgStoredProc](#) component opens automatically the first REFCURSOR returned from a stored procedure. For example, consider the following procedure:

```
CREATE FUNCTION cursor_func() RETURNS REFCURSOR AS $$  
DECLARE  
    ref REFCURSOR;  
BEGIN  
    OPEN ref FOR SELECT * FROM test;  
    RETURN ref;  
END;  
$$ LANGUAGE plpgsql;
```

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

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

Note that using cursors requires a transaction. So that [StartTransaction](#) is called before the Open method of TPgStoredProc.

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

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

You can read data using the following code:

```

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

```

You can open both cursors at the same time by assigning a [TPgRefCursor](#) instance to the Cursor property of a dataset:

```

var
  Cursor: TPgRefCursor;
  PgQuery: TPgQuery;
...
PgConnection.StartTransaction;
PgStoredProc.StoredProcName := 'cursor_func';
PgStoredProc.Open;
Cursor := PgStoredProc.ParamByName('C2').AsCursor;
PgQuery.Cursor := Cursor;
PgQuery.Open; // open the second cursor
Value1 := PgStoredProc.Fields[0].AsInteger;
Value2 := PgQuery.Fields[0].AsInteger;
...
PgStoredProc.Close;
PgQuery.Close;
PgConnection.Commit;

```

4.16 National and Unicode Characters

On transferring data between client and server sides, server must know the encoding used at the client. By default client encoding is the same as the database encoding. You can set the encoding using [TPgConnection.Options.Charset](#) or [TPgConnection.Options.UseUnicode](#) properties. The Charset and UseUnicode options are mutually exclusive, thus on setting the UseUnicode property to True a value of Charset will be ignored.

If the Charset property is set, then on establishing a connection "SET client_encoding = <Charset>" query is automatically passed to the server to explicitly notify the server about the character set of the client. Pay attention that on setting Charset to UTF8 values of all string fields will be converted to this encoding that in most cases can make impossible to use DataAware components.

When you set the UseUnicode option to True, PgDAC also uses UTF8 encoding but it automatically converts all string values to Unicode (UTF-16). TWideStringField and TWideMemoField field types are used instead of TStringField and TMemoField. Setting the UseUnicode option to True lets you work simultaneously almost with all languages. This

behaviour is suitable, for example, when creating a database of books in the library, when next to the title of a book you should also store its title in the original language.

See Also

- [TPgConnection.Options](#)

4.17 Connection Pooling

Connection pooling enables an application to use a connection from a pool of connections that do not need to be reestablished for each use. Once a connection has been created and placed in a pool, an application can reuse that connection without performing the complete connection process.

Using a pooled connection can result in significant performance gains, because applications can save the overhead involved in making a connection. This can be particularly significant for middle-tier applications that connect over a network or for applications that connect and disconnect repeatedly, such as Internet applications.

To use connection pooling set the Pooling property of the [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](#), [Database](#), [Port](#), [ProtocolVersion](#), [Charset](#), [UseUnicode](#), [Schema](#), [ConnectionTimeout](#), [SSLOptions](#). When a connection component disconnects from the database the connection actually remains active and is placed into the pool. When this or another connection component connects to the database it takes a connection from the pool. Only when there are no connections in the pool, new connection is established.

Connections in the pool are validated to make sure that a broken connection will not be returned for the [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 [TCustomDAConnection](#) component disconnects, its internal connection will be freed instead of placing to the pool if this connection is active during the time longer than the value of the [PoolingOptions.ConnectionLifeTime](#) property. This property is designed to make load balancing work with the connection pool.

To force freeing of a connection when the [TCustomDAConnection](#) component disconnects, the [RemoveFromPool](#) method of TCustomDAConnection can be used. You can also free all connection in the pool by using the class procedures Clear or AsyncClear of TPgConnectionPoolManager. These procedures can be useful when you know that all connections will be broken for some reason.

It is recommended to use connection pooling with the [DisconnectMode](#) option of the [TCustomDAConnection](#) component set to True. In this case internal connections can be shared between [TCustomDAConnection](#) components. When some operation is performed on the TCustomDAConnection component (for example, an execution of SQL statement) this component will connect using pooled connection and after performing operation it will disconnect. When an operation is performed on another [TCustomDAConnection](#) component it can use the same connection from the pool.

See Also

- [TCustomDAConnection.Pooling](#)
- [TCustomDAConnection.PoolingOptions](#)
- [Working with Disconnected Mode](#)

4.18 DBMonitor

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

PgDAC.

DBMonitor is an easy-to-use tool to provide visual monitoring of your database applications.

DBMonitor has the following features:

- multiple client processes tracing;
- SQL event filtering (by sender objects);
- SQL parameter and error tracing.

DBMonitor is intended to hamper an application being monitored as little as possible.

To trace your application with DB Monitor you should follow these steps:

- drop [TPgSQLMonitor](#) component onto the form;
- turn [moDBMonitor](#) option on;
- set to True the Debug property for components you want to trace;
- start DBMonitor before running your program.

4.19 Writing GUI Applications with PgDAC

PgDAC GUI part is standalone. This means that to make GUI elements such as SQL cursors, connect form, connect dialog etc. available, you should explicitly include PgDacVcl unit in your application. This feature is needed for writing console applications.

Delphi and C++Builder

By default PgDAC does not require Forms, Controls and other GUI related units. Only [TPgConnectDialog](#) component require the Forms unit.

4.20 Compatibility with Previous Versions

We always try to keep PgDAC compatible with previous versions, but sometimes we have to change the behaviour of PgDAC in order to enhance its functionality, or avoid bugs. This topic describes such changes, and how to revert the old PgDAC behaviour. We strongly recommend not to turn on the old behaviour of PgDAC. Use options described below only if changes applied to PgDAC crashed your existent application.

Values of the options described below should be assigned in the **initialization** section of one of the units in your project.

DBAccess.BaseSQLOldBehavior:

The [BaseSQL](#) property is similar to the SQL property, but it does not store changes made by [AddWhere](#), [DeleteWhere](#), and [SetOrderBy](#) methods. After assigning an SQL text and modifying it by one of these methods, all subsequent changes of the SQL property will not be reflected in the BaseSQL property. This behavior was changed in PgDAC . To restore old

behavior, set the BaseSQLOldBehavior variable to True.

DBAccess.SQLGeneratorCompatibility:

If the manually assigned [RefreshSQL](#) property contains only "WHERE" clause, PgDAC uses the value of the [BaseSQL](#) property to complete the refresh SQL statement. In this situation all modifications applied to the SELECT query by functions [AddWhere](#), [DeleteWhere](#) are not taken into account. This behavior was changed in PgDAC . To restore the old behavior, set the BaseSQLOldBehavior variable to True.

MemDS.SendDataSetChangeEventAfterOpen:

Starting with PgDAC , the DataSetChangeEvent is sent after the dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled. To disable sending this event, change the value of this variable to False.

MemDS.DoNotRaiseExcetionOnUaFail:

Starting with PgDAC , if the [OnUpdateRecord](#) event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged. To restore the old behaviour, set DoNotRaiseExcetionOnUaFail to True.

4.21 64-bit Development with Embarcadero RAD Studio XE2

RAD Studio XE2 Overview

RAD Studio XE2 is the major breakthrough in the line of all Delphi versions of this product. It allows deploying your applications both on Windows and Mac OS platforms. Additionally, it is now possible to create 64-bit Windows applications to fully benefit from the power of new hardware. Moreover, you can create visually spectacular applications with the help of the FireMonkey GPU application platform.

Its main features are the following:

- Windows 64-bit platform support;
- Mac OS support;
- FireMonkey application development platform;
- Live data bindings with visual components;
- VCL styles for Windows applications.

For more information about RAD Studio XE2, please refer to [World Tour](#).

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:

TYPE	32-bit	64-bit
Extended	10 bytes	8 bytes

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.

TYPE	32-bit	64-bit
Pointer	4 bytes	8 bytes

At the same time, the size of the Integer type remains the same for both platforms:

TYPE	32-bit	64-bit
Integer	4 bytes	4 bytes

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

```
Ptr := Pointer(Integer(Ptr) + Offset);
```

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

```
Ptr := Pointer(Int64(Ptr) + Offset);
```

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

TYPE	32-bit	64-bit
NativeInt	4 bytes	8 bytes
NativeUInt	4 bytes	8 bytes

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

```
Ptr := Pointer(NativeInt(Ptr) + Offset);
```

However, you need to be extra-careful when developing applications for several versions of Delphi, in which case you should remember that in the previous versions of Delphi the NativeInt type had different sizes:

TYPE	Delphi Version	Size
NativeInt	D5	N/A
NativeInt	D6	N/A
NativeInt	D7	8 bytes
NativeInt	D2005	8 bytes
NativeInt	D2006	8 bytes
NativeInt	D2007	8 bytes
NativeInt	D2009	4 bytes
NativeInt	D2010	4 bytes
NativeInt	Delphi XE	4 bytes
NativeInt	Delphi XE2	4 or 8 bytes

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:

```
procedure MyProc(out value: NativeInt);
begin
  value := 12345;
end;
var
  value1: NativeInt;
{$IFDEF WIN32}
  value2: Integer;
{$ENDIF}
{$IFDEF WIN64}
  value2: Int64;
{$ENDIF}
begin
  MyProc(value1); // will be compiled;
  MyProc(value2); // will not be compiled !!!
end;
```

Win API

If you pass pointers to `SendMessage/PostMessage/TControl.Perform`, the `wParam` and `lParam` parameters should be type-casted to the `WPARAM/LPARAM` type and not to `Integer/Longint`.

Correct:

```
SendMessage(hwnd, WM_SETTEXT, 0, LPARAM(@MyCharArray));
```

Wrong:

```
SendMessage(hwnd, WM_SETTEXT, 0, Integer(@MyCharArray));
```

Replace `SetWindowLong/GetWindowLog` with `SetWindowLongPtr/GetWindowLongPtr` for `GWLP_HINSTANCE`, `GWLP_ID`, `GWLP_USERDATA`, `GWLP_HWNDPARENT` and `GWLP_WNDPROC` as they return pointers and handles. Pointers that are passed to `SetWindowLongPtr` should be type-casted to `LONG_PTR` and not to `Integer/Longint`.

Correct:

```
SetWindowLongPtr(hwnd, GWLP_WNDPROC, LONG_PTR(@MywindowProc));
```

Wrong:

```
SetWindowLong(hwnd, GWL_WNDPROC, Longint(@MywindowProc));
```

Pointers that are assigned to the `TMessage.Result` field should use a type-cast to `LRESULT` instead of `Integer/Longint`.

Correct:

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

Wrong:

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

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

```
Msg: UINT; wParam: WPARAM; lParam: LPARAM; Result: LRESULT)
```

Assembler

In order to make your application (that uses assembly code) work, you will have to make several changes to it:

- 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.

You can learn more about Assembly code here: http://docwiki.embarcadero.com/RADStudio/en/Using_Inline_Assembly_Code You can also look at the following article that will help you to make your application support the 64-bit platform: http://docwiki.embarcadero.com/RADStudio/en/Converting_32-bit_Delphi_Applications_to_64-bit_Windows

Exception handling

The biggest difference in exception handling between Delphi 32 and 64-bit is that in Delphi XE2 64-bit you will gain more performance because of different internal exception mechanism. For 32-bit applications, the Delphi compiler (dcc32.exe) generates additional code that is executed any way and that causes performance loss. The 64-bit compiler (dcc64.exe) doesn't generate such code, it generates metadata and stores it in the PDATA section of an executable file instead.

But in Delphi XE2 64-bit it's impossible to have more than 16 levels of nested exceptions. Having more than 16 levels of nested exceptions will cause a Run Time error.

Debugging

Debugging of 64-bit applications in RAD Studio XE2 is remote. It is caused by the same reason: RAD Studio XE2 IDE is a 32 application, but your application is 64-bit. If you are trying to debug your application and you cannot do it, you should check that the **Include remote debug symbols** project option is enabled.

To enable it, perform the following steps:

1. Open Project Options (in the main menu **Project->Options**).
2. In the Target combobox, select **Debug configuration - 64-bit Windows platform**. If there is no such option in the combobox, right click "Target Platforms" in Project Manager and select **Add platform**. After adding the 64-bit Windows platform, the **Debug configuration -**

64-bit Windows platform option will be available in the Target combobox.

3. Select **Linking** in the left part of the Project Options form.
4. enable the **Include remote debug symbols** option.

After that, you can run and debug your 64-bit application.

To enable remote debugging, perform the following steps:

1. Install Platform Assistant Server (PAServer) on a remote computer. You can find PAServer in the %RAD_Studio_XE2_Install_Directory%\PAServer directory. The setup_paserver.exe file is an installation file for Windows, and the setup_paserver.zip file is an installation file for MacOS.
2. Run the PAServer.exe file on a remote computer and set the password that will be used to connect to this computer.
3. On a local computer with RAD Studio XE2 installed, right-click the target platform that you want to debug in Project Manager and select **Assign Remote Profile**. Click the **Add** button in the displayed window, input your profile name, click the **Next** button, input the name of a remote computer and the password to it (that you assigned when you started PAServer on a remote computer).

After that, you can test the connection by clicking the **Test Connection** button. If your connection failed, check that your firewalls on both remote and local computers do not block your connection, and try to establish a connection once more. If your connection succeeded, click the Next button and then the Finish button. Select your newly created profile and click **OK**.

After performing these steps you will be able to debug your application on a remote computer. Your application will be executed on a remote computer, but you will be able to debug it on your local computer with RAD Studio XE2.

For more information about working with Platform Assistant Server, please refer to http://docwiki.embarcadero.com/RADStudio/en/Installing_and_Running_the_Platform_Assistant_on_the_Target_Platform

4.22 Database Specific Aspects of 64-bit Development

PostgreSQL Connectivity Aspects

Since PgDAC does not require that the PostgreSQL client be installed to work with the database, the development of applications for the x64 platform does not differ from the development of application for Windows x86.

5 Reference

This page shortly describes units that exist in PgDAC.

Units

Unit Name	Description
CRAccess	This unit contains base classes for accessing databases.
CRBatchMove	This unit contains implementation of the TCRBatchMove component.
CREncryption	This unit contains base classes for data encryption.
CRGrid	Description is not available at the moment.
DAAlerter	This unit contains the base class for the TPgAlerter component.
DADump	This unit contains the base class for the TPgDump component.
DALoader	This unit contains the base class for the TPgLoader component.
DAScript	This unit contains the base class for the TPgScript component.
DASQLMonitor	This unit contains the base class for the TPgSQLMonitor component.
DBAccess	This unit contains base classes for most of the components.
Devart.Dac.DataAdapter	This unit contains implementation of the DADDataAdapter class.
Devart.PgDac.DataAdapter	This unit contains implementation of the PgDataAdapter class.
MemData	This unit contains classes for storing data in memory.
MemDS	This unit contains implementation of the TMemDataSet class.

PgAccess	This unit contains main components of PgDAC.
PgAlerter	This unit contains the implementation of the TPgAlerter component.
PgClasses	This unit contains the implementation of internal PgDAC classes and types.
PgConnectionPool	This unit contains the TPgConnectionPoolManager class for managing connection pool.
PgDacVcl	This unit contains the visual constituent of PgDAC.
PgDump	This unit contains the implementation of the TPgDump component.
PgError	This unit contains the EPgError exception class.
PgLoader	This unit contains the implementation of the TPgLoader component.
PgObjects	This unit contains classes for PostgreSQL specific data types.
PgScript	This unit contains the implementation of the TPgScript component.
PgSQLMonitor	This unit contains the implementation of the TPgSQLMonitor component.
VirtualDataSet	Description is not available at the moment.
VirtualTable	Description is not available at the moment.

5.1 CRAccess

This unit contains base classes for accessing databases.

Classes

Name	Description
TCRCursor	A base class for classes that work with database cursors.

Types

Name	Description
TBeforeFetchProc	This type is used for the TCustomDADDataSet.BeforeFetch event.

Enumerations

Name	Description
TCRIsolationLevel	Specifies how to handle transactions containing database modifications.
TCRTransactionAction	Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.
TCursorState	Used to set cursor state

5.1.1 Classes

Classes in the **CRAccess** unit.

Classes

Name	Description
TCRCursor	A base class for classes that work with database cursors.

5.1.1.1 TCRCursor Class

A base class for classes that work with database cursors.

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

Unit

[CRAccess](#)

Syntax

```
TCRCursor = class (TsharedObject);
```

Remarks

TCRCursor is a base class for classes that work with database cursors.

Inheritance Hierarchy

[TSharedObject](#)

TCRCursor

5.1.1.1.1 Members

[TCRCursor](#) class overview.

Properties

Name	Description
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.1.2 Types

Types in the **CRAccess** unit.

Types

Name	Description
TBeforeFetchProc	This type is used for the TCustomDADataset.BeforeFetch event.

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.

Enumerations

Name	Description
TCRIsolationLevel	Specifies how to handle transactions containing database modifications.
TCRTransactionAction	Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.
TCursorState	Used to set cursor state

5.1.3.1 TCRIsolationLevel Enumeration

Specifies how to handle transactions containing database modifications.

Unit

[CRAccess](#)

Syntax

```
TCRIsolationLevel = (ilReadCommitted);
```

Values

Value	Meaning
ilReadCommitted	The default transaction behavior. If the transaction contains DML that requires row locks held by another transaction, then the DML statement waits until the row locks are released.

5.1.3.2 TCRTTransactionAction 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

```
TCRTTransactionAction = (taCommit, taRollback);
```

Values

Value	Meaning
taCommit	Transaction is committed.
taRollback	Transaction is rolled back.

5.1.3.3 TCursorState Enumeration

Used to set cursor state

Unit

[CRAccess](#)

Syntax

```
TCursorState = (csInactive, csOpen, csParsed, csPrepared, csBound, csExecuteFetchAll, csExecuting, csExecuted, csFetching, csFetchingAll, csFetched);
```

Values

Value	Meaning
csBound	Parameters bound
csExecuted	Statement successfully executed
csExecuteFetchAll	Set before FetchAll
csExecuting	Statement is set before executing
csFetched	Fetch finished or canceled
csFetching	Set on first
csFetchingAll	Set on the FetchAll start
csInactive	Default state
csOpen	statement open

csParsed	Statement parsed
csPrepared	Statement prepared

5.2 CRBatchMove

This unit contains implementation of the TCRBatchMove component.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

Types

Name	Description
TCRBatchMoveProgressEvent	This type is used for the TCRBatchMove.OnBatchMoveProgress event.

Enumerations

Name	Description
TCRBatchMode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
TCRFieldMappingMode	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.

5.2.1 Classes

Classes in the **CRBatchMove** unit.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

5.2.1.1 TCRBatchMove Class

Transfers records between datasets.

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

Unit

[CRBatchMove](#)

Syntax

```
TCRBatchMove = class(TComponent);
```

Remarks

The TCRBatchMove component transfers records between datasets. Use it to copy dataset records to another dataset or to delete datasets records that match records in another dataset. The [TCRBatchMove.Mode](#) property determines the desired operation type, the [TCRBatchMove.Source](#) and [TCRBatchMove.Destination](#) properties indicate corresponding datasets.

Note: A TCRBatchMove component is added to the Data Access page of the component palette, not to the PgDAC page.

5.2.1.1.1 Members

[TCRBatchMove](#) class overview.

Properties

Name	Description
AbortOnKeyViol	Used to specify whether the batch operation should be terminated immediately after key or integrity violation.
AbortOnProblem	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.
ChangedCount	Used to get the number of records changed in the destination dataset.
CommitCount	Used to set the number of records to be batch moved before commit occurs.

Destination	Used to specify the destination dataset for the batch operation.
FieldMappingMode	Used to specify the way fields of destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.
KeyViolCount	Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.
Mappings	Used to set field matching between source and destination datasets for the batch operation.
Mode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
MovedCount	Used to get the number of records that were read from the source dataset during the batch operation.
ProblemCount	Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.
RecordCount	Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.
Source	Used to specify the source dataset for the batch operation.

Methods

Name	Description
Execute	Performs the batch operation.

Events

Name	Description
------	-------------

[OnBatchMoveProgress](#)

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

5.2.1.1.2 Properties

Properties of the **TCRBatchMove** class.

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

Public

Name	Description
<u>ChangedCount</u>	Used to get the number of records changed in the destination dataset.
<u>KeyViolCount</u>	Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.
<u>MovedCount</u>	Used to get the number of records that were read from the source dataset during the batch operation.
<u>ProblemCount</u>	Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.

Published

Name	Description
<u>AbortOnKeyViol</u>	Used to specify whether the batch operation should be terminated immediately after key or integrity violation.
<u>AbortOnProblem</u>	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.
<u>CommitCount</u>	Used to set the number of records to be batch moved before commit occurs.

Destination	Used to specify the destination dataset for the batch operation.
FieldMappingMode	Used to specify the way fields of destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.
Mappings	Used to set field matching between source and destination datasets for the batch operation.
Mode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
RecordCount	Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.
Source	Used to specify the source dataset for the batch operation.

See Also

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

5.2.1.1.2.1 AbortOnKeyViol Property

Used to specify whether the batch operation should be terminated immediately after key or integrity violation.

Class

[TCRBatchMove](#)

Syntax

```
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

```
property ChangedCount: Integer;
```

Remarks

Use the ChangedCount property to get the number of records changed in the destination dataset. It shows the number of records that were updated in the bmUpdate or bmAppendUpdate mode or were deleted in the bmDelete mode.

5.2.1.1.2.4 CommitCount Property

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

Class

[TCRBatchMove](#)

Syntax

```
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.

5.2.1.1.2.5 Destination Property

Used to specify the destination dataset for the batch operation.

Class

[TCRBatchMove](#)

Syntax

```
property Destination: TDataSet;
```

Remarks

Specifies the destination dataset for the batch operation.

5.2.1.1.2.6 FieldMappingMode Property

Used to specify the way fields of destination and source datasets will be mapped to each other if the [Mappings](#) list is empty.

Class

[TCRBatchMove](#)

Syntax

```
property FieldMappingMode: TCRFieldMappingMode default  
mmFieldIndex;
```

Remarks

Specifies in what way fields of destination and source datasets will be mapped to each other if the [Mappings](#) list is empty.

5.2.1.1.2.7 KeyViolCount Property

Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.

Class

[TCRBatchMove](#)

Syntax

```
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](#)

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: TString;
```

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

```
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

```
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.

5.2.1.1.2.13 Source Property

Used to specify the source dataset for the batch operation.

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

Name	Description
Execute	Performs the batch operation.

See Also

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

5.2.1.1.3.1 Execute Method

Performs the batch operation.

Class

[TCRBatchMove](#)

Syntax

```
procedure Execute;
```

Remarks

Call the Execute method to perform the batch operation.

5.2.1.1.4 Events

Events of the **TCRBatchMove** class.

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

Published

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

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

Name	Description
TCRBatchMoveProgressEvent	This type is used for the TCRBatchMove.OnBatchMoveProgress event.

5.2.2.1 TCRBatchMoveProgressEvent Procedure Reference

This type is used for the [TCRBatchMove.OnBatchMoveProgress](#) event.

Unit

[CRBatchMove](#)

Syntax

```
TCRBatchMoveProgressEvent = procedure (Sender: TObject; Percent: integer) of object;
```

Parameters

Sender

An object that raised the event.

Percent

Percentage of the batch operation progress.

5.2.3 Enumerations

Enumerations in the **CRBatchMove** unit.

Enumerations

Name	Description
TCRBatchMode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
TCRFieldMappingMode	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.

5.2.3.1 TCRBatchMode Enumeration

Used to set the type of the batch operation that will be executed after calling the [TCRBatchMove.Execute](#) method.

Unit

[CRBatchMove](#)

Syntax

```
TCRBatchMode = (bmAppend, bmUpdate, bmAppendUpdate, bmDelete);
```

Values

Value	Meaning
bmAppend	Appends the records from the source dataset to the destination dataset. The default mode.
bmAppendUpdate	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.
bmDelete	Deletes records from the destination dataset if there are matching records in the source dataset.

bmUpdate	Replaces records in the destination dataset with the matching records from the source dataset.
-----------------	--

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

Value	Meaning
mmFieldIndex	Specifies that the fields of the destination dataset will be mapped to the fields of the source dataset by field index.
mmFieldName	Mapping is performed by field names.

5.3 CREncryption

This unit contains base classes for data encryption.

Classes

Name	Description
TCREncryptor	The class that performs data encryption and decryption in a client application using various encryption algorithms .

Enumerations

Name	Description
TCREncDataHeader	Specifies whether the additional information is stored with the encrypted data.
TCREncryptionAlgorithm	Specifies the algorithm of data encryption.

TCRHashAlgorithm	Specifies the algorithm of generating hash data.
TCRInvalidHashAction	Specifies the action to perform on data fetching when hash data is invalid.

5.3.1 Classes

Classes in the **CREncryption** unit.

Classes

Name	Description
TCREncryptor	The class that performs data encryption and decryption in a client application using various encryption algorithms .

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);
```

5.3.1.1.1 Members

[TCREncryptor](#) class overview.

Properties

Name	Description
DataHeader	Specifies whether the additional information is stored with the encrypted data.
EncryptionAlgorithm	Specifies the algorithm of data encryption.

HashAlgorithm	Specifies the algorithm of generating hash data.
InvalidHashAction	Specifies the action to perform on data fetching when hash data is invalid.
Password	Used to set a password that is used to generate a key for encryption.

Methods

Name	Description
SetKey	Sets a key, using which data is encrypted.

5.3.1.1.2 Properties

Properties of the **TCREncryptor** class.

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

Published

Name	Description
DataHeader	Specifies whether the additional information is stored with the encrypted data.
EncryptionAlgorithm	Specifies the algorithm of data encryption.
HashAlgorithm	Specifies the algorithm of generating hash data.
InvalidHashAction	Specifies the action to perform on data fetching when hash data is invalid.
Password	Used to set a password that is used to generate a key for encryption.

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](#).

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](#).

5.3.1.1.2.3 HashAlgorithm Property

Specifies the algorithm of generating hash data.

Class

[TCREncryptor](#)

Syntax

```
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](#).

5.3.1.1.2.4 InvalidHashAction Property

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

Class

[TCREncryptor](#)

Syntax

```
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 exception is raised.

5.3.1.1.2.5 Password Property

Used to set a password that is used to generate a key for encryption.

Class

[TCREncryptor](#)

Syntax

```
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.

5.3.1.1.3 Methods

Methods of the **TCREncryptor** class.

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

Public

Name	Description
SetKey	Sets a key, using which data is encrypted.

See Also

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

5.3.1.1.3.1 SetKey Method

Sets a key, using which data is encrypted.

Class

[TCREncryptor](#)

Syntax

```
procedure SetKey(const Key; Count: Integer); overload;procedure  
SetKey(const Key: TBytes; Offset: Integer; Count: Integer);  
overload;
```

Parameters

Key

Holds bytes that represent a key.

Offset

Offset in bytes to the position, where the key begins.

Count

Number of bytes to use from Key.

Remarks

Use SetKey to set a key, using which data is encrypted.

Note: Calling of the SetKey method clears the Password property.

5.3.2 Enumerations

Enumerations in the **CREncryption** unit.

Enumerations

Name	Description
TCREncDataHeader	Specifies whether the additional information is stored with the encrypted data.
TCREncryptionAlgorithm	Specifies the algorithm of data encryption.
TCRHashAlgorithm	Specifies the algorithm of generating hash data.
TCRInvalidHashAction	Specifies the action to perform on data fetching when hash data is invalid.

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

Value	Meaning
ehNone	No additional information is stored.
ehTag	GUID and the random initialization vector are stored with the encrypted data.
ehTagAndHash	Hash, GUID, and the random initialization vector are stored with the encrypted data.

5.3.2.2 TCREncryptionAlgorithm Enumeration

Specifies the algorithm of data encryption.

Unit

[CREncryption](#)

Syntax

```
TCREncryptionAlgorithm = (eaTripleDES, eaBlowfish, eaAES128,  
eaAES192, eaAES256, eaCast128, eaRC4);
```

Values

Value	Meaning
eaAES128	The AES encryption algorithm with key size of 128 bits is used.
eaAES192	The AES encryption algorithm with key size of 192 bits is used.
eaAES256	The AES encryption algorithm with key size of 256 bits is used.
eaBlowfish	The Blowfish encryption algorithm is used.
eaCast128	The CAST-128 encryption algorithm with key size of 128 bits is used.
eaRC4	The RC4 encryption algorithm is used.
eaTripleDES	The Triple DES encryption algorithm is used.

5.3.2.3 TCRHashAlgorithm Enumeration

Specifies the algorithm of generating hash data.

Unit

[CREncryption](#)

Syntax

```
TCRHashAlgorithm = (haSHA1, haMD5);
```

Values

Value	Meaning
haMD5	The MD5 hash algorithm is used.
haSHA1	The SHA-1 hash algorithm is used.

5.3.2.4 TCRIInvalidHashAction Enumeration

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

Unit

[CREncryption](#)

Syntax

```
TCRIInvalidHashAction = (ihFail, ihSkipData, ihIgnoreError);
```

Values

Value	Meaning
ihFail	The EInvalidHash exception is raised and further data reading from the server is interrupted.
ihIgnoreError	In spite of the fact that the data is not valid, the value is set in the field. No exception is raised.
ihSkipData	The value of the field for this record is set to Null. No exception is raised.

5.4 CRGrid

5.4.1 Classes

Classes in the **CRGrid** unit.

Classes

Name	Description
TCRDBGrid	Addresses extended functionality commonly not found in TDBGrid component.

5.4.1.1 TCRDBGrid Class

Addresses extended functionality commonly not found in TDBGrid component.

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

Unit

CRGrid

Syntax

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

5.4.1.1.1 Members

[TCRDBGrid](#) class overview.

5.5 DAAlerter

This unit contains the base class for the TPgAlerter component.

Classes

Name	Description
TDAAlerter	A base class that defines functionality for database event notification.

Types

Name	Description
TAlerterErrorEvent	This type is used for the TDAAlerter.OnError event.

5.5.1 Classes

Classes in the **DAAlerter** unit.

Classes

Name	Description
TDAAlerter	A base class that defines functionality for database event notification.

5.5.1.1 TDAAlerter Class

A base class that defines functionality for database event notification.

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

Unit

[DAAlerter](#)

Syntax

```
TDAAlerter = class(TComponent);
```

Remarks

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

The TDAAlerter component allows you to register interest in and handle events posted by a database server. Use TDAAlerter to handle events for responding to actions and database changes made by other applications. To get events, an application must register required events. To do this, set the Events property to the required events and call the Start method. When one of the registered events occurs OnEvent handler is called.

5.5.1.1.1 Members

[TDAAlerter](#) class overview.

Properties

Name	Description
Active	Used to determine if TDAAlerter waits for messages.
AutoRegister	Used to automatically register events whenever connection opens.
Connection	Used to specify the connection for TDAAlerter.

Methods

Name	Description
SendEvent	Sends an event with Name and content Message.
Start	Starts waiting process.
Stop	Stops waiting process.

Events

Name	Description
OnError	Occurs if an exception occurs in waiting process

5.5.1.1.2 Properties

Properties of the **TDAAlerter** class.

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

Public

Name	Description
Active	Used to determine if TDAAlerter waits for messages.
AutoRegister	Used to automatically register events whenever connection opens.
Connection	Used to specify the connection for TDAAlerter.

See Also

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

5.5.1.1.2.1 Active Property

Used to determine if TDAAlerter waits for messages.

Class

[TDAAlerter](#)

Syntax

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

Remarks

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

See Also

- [Start](#)
- [Stop](#)

5.5.1.1.2.2 AutoRegister Property

Used to automatically register events whenever connection opens.

Class

[TDAAlerter](#)

Syntax

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

Remarks

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

5.5.1.1.2.3 Connection Property

Used to specify the connection for TDAAlerter.

Class

[TDAAlerter](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify the connection for TDAAlerter.

5.5.1.1.3 Methods

Methods of the **TDAAlerter** class.

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

Public

Name	Description
SendEvent	Sends an event with Name and content Message.
Start	Starts waiting process.

[Stop](#)

Stops waiting process.

See Also

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

5.5.1.1.3.1 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.

5.5.1.1.3.2 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](#)

5.5.1.1.3.3 Stop Method

Stops waiting process.

Class

[TDAAlerter](#)

Syntax

```
procedure Stop;
```

Remarks

Call Stop method to end waiting process.

See Also

- [Start](#)

5.5.1.1.4 Events

Events of the **TDAAlerter** class.

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

Public

Name	Description
OnError	Occurs if an exception occurs in waiting process

See Also

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

5.5.1.1.4.1 OnError Event

Occurs if an exception occurs in waiting process

Class

[TDAAlerter](#)

Syntax

```
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.

5.5.2 Types

Types in the **DAAlerter** unit.

Types

Name	Description
TAlerterErrorEvent	This type is used for the TDAAlerter.OnError event.

5.5.2.1 TAlerterErrorEvent Procedure Reference

This type is used for the TDAAlerter.OnError event.

Unit

[DAAlerter](#)

Syntax

```
TAlerterErrorEvent = procedure (Sender: TDAAlerter; E: Exception)  
of object;
```

Parameters

Sender

An object that raised the event.

E

Exception object.

5.6 DADump

This unit contains the base class for the TPgDump component.

Classes

Name	Description
TDADump	A base class that defines functionality for descendant classes that dump database objects to a script.
TDADumpOptions	This class allows setting up the behaviour of the TDADump class.

Types

Name	Description
TDABackupProgressEvent	This type is used for the TDADump.OnBackupProgress event.
TDARestoreProgressEvent	This type is used for the TDADump.OnRestoreProgress event.

5.6.1 Classes

Classes in the **DADump** unit.

Classes

Name	Description
TDADump	A base class that defines functionality for descendant classes that dump database objects to a script.
TDADumpOptions	This class allows setting up the behaviour of the TDADump class.

5.6.1.1 TDADump Class

A base class that defines functionality for descendant classes that dump database objects to a script.

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

Unit

[DADump](#)

Syntax

```
TDADump = class(TComponent);
```

Remarks

TDADump is a base class that defines functionality for descendant classes that dump database objects to a script. Applications never use TDADump objects directly. Instead they use descendants of TDADump.

Use TDADump descendants to dump database objects, such as tables, stored procedures, and functions for backup or for transferring the data to another SQL server. The dump contains SQL statements to create the table or other database objects and/or populate the table.

5.6.1.1.1 Members

[TDADump](#) class overview.

Properties

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
Debug	Used to display executing statement, all its parameters' values, and the type of parameters.
Options	Used to specify the behaviour of a TDADump component.
SQL	Used to set or get the dump script.
TableNames	Used to set the names of the tables to dump.

Methods

Name	Description
Backup	Dumps database objects to the TDADump.SQL property.
BackupQuery	Dumps the results of a particular query.
BackupToFile	Dumps database objects to the specified file.

BackupToStream	Dumps database objects to the stream.
Restore	Executes a script contained in the SQL property.
RestoreFromFile	Executes a script from a file.
RestoreFromStream	Executes a script received from the stream.

Events

Name	Description
OnBackupProgress	Occurs to indicate the TDADump.Backup , M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.
OnError	Occurs when PostgreSQL raises some error on TDADump.Restore .
OnRestoreProgress	Occurs to indicate the TDADump.Restore , TDADump.RestoreFromFile , or TDADump.RestoreFromStream method execution progress.

5.6.1.1.2 Properties

Properties of the **TDADump** class.

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

Public

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
Options	Used to specify the behaviour of a TDADump component.

Published

Name	Description
Debug	Used to display executing statement, all its parameters' values, and the type of parameters.
SQL	Used to set or get the dump script.
TableNames	Used to set the names of the tables to dump.

See Also

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

5.6.1.1.2.1 Connection Property

Used to specify a connection object that will be used to connect to a data store.

Class

[TDADump](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

See Also

- [TCustomDAConnection](#)

5.6.1.1.2.2 Debug Property

Used to display executing statement, all its parameters' values, and the type of parameters.

Class

[TDADump](#)

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display executing statement and all its parameters' values. Also displays the type of parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

See Also

- [TCustomDADDataSet.Debug](#)
- [TCustomDASQL.Debug](#)

5.6.1.1.2.3 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.

Option Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
GenerateHeader	Used to add a comment header to a script.

[QuoteNames](#)

Used for TDADump to quote all database object names in generated SQL statements.

5.6.1.1.2.4 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](#)

5.6.1.1.2.5 TableNames Property

Used to set the names of the tables to dump.

Class

[TDADump](#)

Syntax

```
property TableNames: string;
```

Remarks

Use the TableNames property to set the names of the tables to dump. Table names must be separated with commas. If it is empty, the [Backup](#) method will dump all available tables.

See Also

- [Backup](#)

5.6.1.1.3 Methods

Methods of the **TDADump** class.

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

Public

Name	Description
Backup	Dumps database objects to the TDADump.SQL property.
BackupQuery	Dumps the results of a particular query.
BackupToFile	Dumps database objects to the specified file.
BackupToStream	Dumps database objects to the stream.
Restore	Executes a script contained in the SQL property.
RestoreFromFile	Executes a script from a file.
RestoreFromStream	Executes a script received from the stream.

See Also

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

5.6.1.1.3.1 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.

See Also

- [SQL](#)
- [Restore](#)
- [BackupToFile](#)
- [BackupToStream](#)
- [BackupQuery](#)

5.6.1.1.3.2 BackupQuery Method

Dumps the results of a particular query.

Class

[TDADump](#)

Syntax

```
procedure BackupQuery(const Query: string);
```

Parameters

Query

Holds a query used for data selection.

Remarks

Call the BackupQuery method to dump the results of a particular query. Query must be a valid select statement. If this query selects data from several tables, only data of the first table in the from list will be dumped.

See Also

- [Restore](#)
- [Backup](#)
- [BackupToFile](#)
- [BackupToStream](#)

5.6.1.1.3.3 BackupToFile Method

Dumps database objects to the specified file.

Class

[TDADump](#)

Syntax

```
procedure BackupToFile(const FileName: string; const Query:  
string = '');;
```

Parameters

FileName

Holds the file name to dump database objects to.

Query

Your query to receive the data for dumping.

Remarks

Call the BackupToFile method to dump database objects to the specified file.

See Also

- [RestoreFromStream](#)
- [Backup](#)
- [BackupToStream](#)

5.6.1.1.3.4 BackupToStream Method

Dumps database objects to the stream.

Class

[TDADump](#)

Syntax

```
procedure BackupToStream(Stream: TStream; const Query: string =  
'');
```

Parameters

Stream

Holds the stream to dump database objects to.

Query

Your query to receive the data for dumping.

Remarks

Call the BackupToStream method to dump database objects to the stream.

See Also

- [RestoreFromStream](#)
- [Backup](#)
- [BackupToFile](#)

5.6.1.1.3.5 Restore Method

Executes a script contained in the SQL property.

Class

[TDADump](#)

Syntax

```
procedure Restore;
```

Remarks

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

See Also

- [RestoreFromFile](#)
- [RestoreFromStream](#)
- [Backup](#)
- [SQL](#)

5.6.1.1.3.6 RestoreFromFile Method

Executes a script from a file.

Class

[TDADump](#)

Syntax

```
procedure RestoreFromFile(const FileName: string);
```

Parameters

FileName

Holds the file name to execute a script from.

Remarks

Call the `RestoreFromFile` method to execute a script from the specified file.

See Also

- [Restore](#)
- [RestoreFromStream](#)
- [BackupToFile](#)

5.6.1.1.3.7 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](#)

5.6.1.1.4 Events

Events of the **TDADump** class.

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

Published

Name	Description
------	-------------

OnBackupProgress	Occurs to indicate the TDADump.Backup , M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.
OnError	Occurs when PostgreSQL raises some error on TDADump.Restore .
OnRestoreProgress	Occurs to indicate the TDADump.Restore , TDADump.RestoreFromFile , or TDADump.RestoreFromStream method execution progress.

See Also

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

5.6.1.1.4.1 OnBackupProgress Event

Occurs to indicate the [Backup](#), M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.

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](#)

5.6.1.1.4.2 OnError Event

Occurs when PostgreSQL raises some error on [Restore](#).

Class

[TDADump](#)

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

The OnError event occurs when PostgreSQL raises some error on [Restore](#).

Action indicates the action to take when the OnError handler exits. On entry into the handler, Action is always set to eaException.

Note: You should add the DAScript module to the 'uses' list to use the OnError event handler.

5.6.1.1.4.3 OnRestoreProgress Event

Occurs to indicate the [Restore](#), [RestoreFromFile](#), or [RestoreFromStream](#) method execution progress.

Class

[TDADump](#)

Syntax

```
property OnRestoreProgress: TDARestoreProgressEvent;
```

Remarks

The OnRestoreProgress event occurs several times during the dumping process of the [Restore](#), [RestoreFromFile](#), or [RestoreFromStream](#) method execution and indicates its progress. The Percent parameter of the OnRestoreProgress event handler indicates the percentage of the whole restore script execution.

See Also

- [Restore](#)
- [RestoreFromFile](#)
- [RestoreFromStream](#)

5.6.1.2 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);
```

5.6.1.2.1 Members

[TDADumpOptions](#) class overview.

Properties

Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

5.6.1.2.2 Properties

Properties of the **TDADumpOptions** class.

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

Published

Name	Description
AddDrop	Used to add drop statements to a script before creating statements.

GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

See Also

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

5.6.1.2.2.1 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.

5.6.1.2.2.2 GenerateHeader Property

Used to add a comment header to a script.

Class

[TDADumpOptions](#)

Syntax

```
property GenerateHeader: boolean default True;
```

Remarks

Use the GenerateHeader property to add a comment header to a script. It contains script generation date, DAC version, and some other information.

5.6.1.2.2.3 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.

5.6.2 Types

Types in the **DADump** unit.

Types

Name	Description
TDABackupProgressEvent	This type is used for the TDADump.OnBackupProgress event.
TDARestoreProgressEvent	This type is used for the TDADump.OnRestoreProgress event.

5.6.2.1 TDABackupProgressEvent Procedure Reference

This type is used for the [TDADump.OnBackupProgress](#) event.

Unit

[DADump](#)

Syntax

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

Parameters

Sender

An object that raised the event.

ObjectName

The name of the currently dumping database object.

ObjectNum

The number of the current database object in the backup queue starting from zero.

ObjectCount

The quantity of database objects to dump.

Percent

The current percentage of the current table data dumped.

5.6.2.2 TDARestoreProgressEvent Procedure Reference

This type is used for the [TDADump.OnRestoreProgress](#) event.

Unit

[DADump](#)

Syntax

```
TDARestoreProgressEvent = procedure (Sender: TObject; Percent: integer) of object;
```

Parameters

Sender

An object that raised the event.

Percent

The percentage of the whole restore script execution.

5.7 DALoader

This unit contains the base class for the TPgLoader component.

Classes

Name	Description
TDAColumn	Represents the attributes for column loading.
TDAColumns	Holds a collection of TDAColumn objects.
TDALoader	This class allows loading external data into database.
TDALoaderOptions	Allows loading external data into database.

Types

Name	Description
TDAPutDataEvent	This type is used for the TDALoader.OnPutData event.
TGetColumnDataEvent	This type is used for the TDALoader.OnGetColumnData event.
TLoaderProgressEvent	This type is used for the TDALoader.OnProgress event.

5.7.1 Classes

Classes in the **DALoader** unit.

Classes

Name	Description
TDAColumn	Represents the attributes for column loading.
TDAColumns	Holds a collection of TDAColumn objects.
TDALoader	This class allows loading external data into database.
TDALoaderOptions	Allows loading external data into database.

5.7.1.1 TDAColumn Class

Represents the attributes for column loading.

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

Unit

[DALoader](#)

Syntax

```
TDAColumn = class(TCollectionItem);
```

Remarks

Each [TDALoader](#) uses [TDAColumns](#) to maintain a collection of TDAColumn objects. TDAColumn object represents the attributes for column loading. Every TDAColumn object corresponds to one of the table fields with the same name as its [TDAColumn.Name](#) property. To create columns at design-time use the column editor of the [TDALoader](#) component.

See Also

- [TDALoader](#)
- [TDAColumns](#)

5.7.1.1.1 Members

[TDAColumn](#) class overview.

Properties

Name	Description
FieldType	Used to specify the types of values that will be loaded.
Name	Used to specify the field name of loading table.

5.7.1.1.2 Properties

Properties of the **TDAColumn** class.

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

Published

Name	Description
FieldType	Used to specify the types of values that will be loaded.
Name	Used to specify the field name of loading table.

See Also

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

5.7.1.1.2.1 FieldType Property

Used to specify the types of values that will be loaded.

Class

[TDAColumn](#)

Syntax

```
property FieldType: TFieldType default ftString;
```

Remarks

Use the FieldType property to specify the types of values that will be loaded. Field types for columns may not match data types for the corresponding fields in the database table.

[TDALoader](#) will cast data values to the types of their fields.

5.7.1.1.2.2 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](#)

5.7.1.2 TDAColumns Class

Holds a collection of [TDAColumn](#) objects.

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

Unit

[DALoader](#)

Syntax

```
TDAColumns = class(TownedCollection);
```

Remarks

Each TDAColumns holds a collection of [TDAColumn](#) objects. TDAColumns maintains an index of the columns in its Items array. The Count property contains the number of columns in the collection. At design-time, use the Columns editor to add, remove, or modify columns.

See Also

- [TDALoader](#)
- [TDAColumn](#)

5.7.1.2.1 Members

[TDAColumns](#) class overview.

Properties

Name	Description
Items	Used to access individual columns.

5.7.1.2.2 Properties

Properties of the **TDAColumns** class.

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

Public

Name	Description
Items	Used to access individual columns.

See Also

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

5.7.1.2.2.1 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](#)

5.7.1.3 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.

5.7.1.3.1 Members

[TDALoader](#) class overview.

Properties

Name	Description
------	-------------

Columns	Used to add a TDAColumn object for each field that will be loaded.
Connection	Used to specify TCustomDACConnection in which TDALoader will be executed.
TableName	Used to specify the name of the table to which data will be loaded.

Methods

Name	Description
CreateColumns	Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName .
Load	Starts loading data.
LoadFromDataSet	Loads data from the specified dataset.
PutColumnData	Overloaded. Puts the value of individual columns.

Events

Name	Description
OnGetColumnData	Occurs when it is needed to put column values.
OnProgress	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.
OnPutData	Occurs when putting loading data by rows is needed.

5.7.1.3.2 Properties

Properties of the **TDALoader** class.

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

Public

Name	Description
Columns	Used to add a TDAColumn object for each field that will be loaded.
Connection	Used to specify TCustomDAConnection in which TDALoader will be executed.
TableName	Used to specify the name of the table to which data will be loaded.

See Also

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

5.7.1.3.2.1 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](#)

5.7.1.3.2.2 Connection 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](#)

5.7.1.3.2.3 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](#)
- [TCustomDAConnection.GetTableNames](#)

5.7.1.3.3 Methods

Methods of the **TDALoader** class.

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

Public

Name	Description
CreateColumns	Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName .
Load	Starts loading data.

LoadFromDataSet	Loads data from the specified dataset.
PutColumnData	Overloaded. Puts the value of individual columns.

See Also

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

5.7.1.3.3.1 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](#)

5.7.1.3.3.2 Load Method

Starts loading data.

Class

[TDALoader](#)

Syntax

```
procedure Load; virtual;
```

Remarks

Call the Load method to start loading data. At first it is necessary to [create columns](#) and write one of the [OnPutData](#) or [OnGetColumnData](#) event handlers.

See Also

- [OnGetColumnData](#)
- [OnPutData](#)

5.7.1.3.3.3 LoadFromDataSet Method

Loads data from the specified dataset.

Class

[TDALoader](#)

Syntax

```
procedure LoadFromDataSet(DataSet: TDataSet);
```

Parameters

DataSet

Holds the dataset to load data from.

Remarks

Call the LoadFromDataSet method to load data from the specified dataset. There is no need to create columns and write event handlers for [OnPutData](#) and [OnGetColumnData](#) before calling this method.

5.7.1.3.3.4 PutColumnData Method

Puts the value of individual columns.

Class

[TDALoader](#)

Overload List

Name	Description
PutColumnData (Col: integer; Row: integer; const Value: variant)	Puts the value of individual columns by the column index.
PutColumnData (const ColName: string ;	Puts the value of individual columns by the

Row: integer; const Value: variant)	column name.
--	--------------

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

```
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.

5.7.1.3.4 Events

Events of the **TDALoader** class.

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

Public

Name	Description
OnGetColumnData	Occurs when it is needed to put column values.
OnProgress	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.
OnPutData	Occurs when putting loading data by rows is needed.

See Also

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

5.7.1.3.4.1 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 TfmMain.GetColumnData(Sender: TObject;
    Column: TDAColumn; Row: Integer; var Value: Variant;
    var EOF: Boolean);
begin
    if Row <= 1000 then begin
        case Column.Index of
            0: Value := Row;
            1: Value := Random(100);
            2: Value := Random*100;
            3: Value := 'abc01234567890123456789';
            4: Value := Date;
        else
            Value := Null;
        end;
    end
    else
        EOF := True;
end;
```

See Also

- [OnPutData](#)
- [Load](#)

5.7.1.3.4.2 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](#)

5.7.1.3.4.3 OnPutData Event

Occurs when putting loading data by rows is needed.

Class

[TDALoader](#)

Syntax

property OnPutData: [TDAPutDataEvent](#);

Remarks

Write the OnPutData event handler to put loading data by rows.

Note that rows should be loaded from the first in the ascending order.

To start loading, call the [Load](#) method.

Example

This handler loads 1000 rows.

```
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](#)

5.7.1.4 TDALoaderOptions Class

Allows loading external data into database.

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

Unit

[DALoader](#)

Syntax

```
TDALoaderOptions = class(TPersistent);
```

5.7.1.4.1 Members

[TDALoaderOptions](#) class overview.

Properties

Name	Description
UseBlankValues	Forces PgDAC to fill the buffer with null values after loading a row to the database.

5.7.1.4.2 Properties

Properties of the **TDALoaderOptions** class.

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

Public

Name	Description
UseBlankValues	Forces PgDAC to fill the buffer with null values after loading a row to the database.

See Also

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

5.7.1.4.2.1 UseBlankValues Property

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

Class

[TDALoaderOptions](#)

Syntax

```
property UseBlankValues: boolean default True;
```

Remarks

Used to force PgDAC to fill the buffer with null values after loading a row to the database.

5.7.2 Types

Types in the **DALoader** unit.

Types

Name	Description
TDAPutDataEvent	This type is used for the TDALoader.OnPutData event.
TGetColumnDataEvent	This type is used for the TDALoader.OnGetColumnData event.
TLoaderProgressEvent	This type is used for the TDALoader.OnProgress event.

5.7.2.1 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.

5.7.2.2 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.

5.7.2.3 TLoaderProgressEvent Procedure Reference

This type is used for the [TDALoader.OnProgress](#) event.

Unit

[DALoader](#)

Syntax

```
TLoaderProgressEvent = procedure (Sender: TObject; Percent:
integer) of object;
```

Parameters

Sender

An object that raised the event.

Percent

Percentage of the load operation progress.

5.8 DAScript

This unit contains the base class for the TPgScript component.

Classes

Name	Description
TDAScript	Makes it possible to execute several SQL statements one by one.
TDAStatement	This class has attributes and methods for controlling single SQL statement of a script.
TDAStatements	Holds a collection of TDAStatement objects.

Types

Name	Description
TAfterStatementExecuteEvent	This type is used for the TDAScript.AfterExecute event.
TBeforeStatementExecuteEvent	This type is used for the TDAScript.BeforeExecute event.
TOnErrorEvent	This type is used for the TDAScript.OnError event.

Enumerations

Name	Description
TErrorAction	Indicates the action to take when the OnError handler exits.

5.8.1 Classes

Classes in the **DAScript** unit.

Classes

Name	Description
TDAScript	Makes it possible to execute several SQL statements one by one.

TDASStatement	This class has attributes and methods for controlling single SQL statement of a script.
TDASStatements	Holds a collection of TDASStatement objects.

5.8.1.1 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](#)

5.8.1.1.1 Members

[TDAScript](#) class overview.

Properties

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

DataSet	Refers to a dataset that holds the result set of query execution.
Debug	Used to display the script execution and all its parameter values.
Delimiter	Used to set the delimiter string that separates script statements.
EndLine	Used to get the current statement last line number in a script.
EndOffset	Used to get the offset in the last line of the current statement.
EndPos	Used to get the end position of the current statement.
Macros	Used to change SQL script text in design- or run-time easily.
SQL	Used to get or set script text.
StartLine	Used to get the current statement start line number in a script.
StartOffset	Used to get the offset in the first line of the current statement.
StartPos	Used to get the start position of the current statement in a script.
Statements	Contains a list of statements obtained from the SQL property.

Methods

Name	Description
BreakExec	Stops script execution.
ErrorOffset	Used to get the offset of the statement if the Execute method raised an exception.
Execute	Executes a script.

ExecuteFile	Executes SQL statements contained in a file.
ExecuteNext	Executes the next statement in the script and then stops.
ExecuteStream	Executes SQL statements contained in a stream object.
MacroByName	Finds a Macro with the name passed in Name.

Events

Name	Description
AfterExecute	Occurs after a SQL script execution.
BeforeExecute	Occurs when taking a specific action before executing the current SQL statement is needed.
OnError	Occurs when PostgreSQL raises an error.

5.8.1.1.2 Properties

Properties of the **TDAScript** class.

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

Public

Name	Description
Connection	Used to specify the connection in which the script will be executed.
DataSet	Refers to a dataset that holds the result set of query execution.
EndLine	Used to get the current statement last line number in a script.
EndOffset	Used to get the offset in the last line of the current statement.
EndPos	Used to get the end position of the current statement.

StartLine	Used to get the current statement start line number in a script.
StartOffset	Used to get the offset in the first line of the current statement.
StartPos	Used to get the start position of the current statement in a script.
Statements	Contains a list of statements obtained from the SQL property.

Published

Name	Description
Debug	Used to display the script execution and all its parameter values.
Delimiter	Used to set the delimiter string that separates script statements.
Macros	Used to change SQL script text in design- or run-time easily.
SQL	Used to get or set script text.

See Also

- [TDA Script Class](#)
- [TDA Script Class Members](#)

5.8.1.1.2.1 Connection Property

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

Class

[TDA Script](#)

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 [TCustomDAConnection](#) objects. At run-time, set the Connection property to reference an existing TCustomDAConnection object.

See Also

- [TCustomDAConnection](#)

5.8.1.1.2.2 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

- [ExecuteNext](#)
- [Execute](#)

5.8.1.1.2.3 Debug Property

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

Class

[TDAScript](#)

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display executing statement and all its parameters' values. Also displays the type of parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the

Debug property work.

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

5.8.1.1.2.4 Delimiter Property

Used to set the delimiter string that separates script statements.

Class

[TDAScript](#)

Syntax

```
property Delimiter: string stored IsDelimiterStored;
```

Remarks

Use the Delimiter property to set the delimiter string that separates script statements. By default it is semicolon (;). You can use slash (/) to separate statements that can contain semicolon if the Delimiter property's default value is semicolon. Note that slash must be the first character in line.

5.8.1.1.2.5 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.

5.8.1.1.2.6 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.

5.8.1.1.2.7 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.

5.8.1.1.2.8 Macros Property

Used to change SQL script text in design- or run-time easily.

Class

[TDAScript](#)

Syntax

property Macros: [TMacros](#) **stored** False;

Remarks

With the help of macros you can easily change SQL script text in design- or run-time. Macros extend abilities of parameters and allow changing conditions in the WHERE clause or sort order in the ORDER BY clause. You just insert &MacroName in a SQL query text and change value of macro by the Macro property editor in design-time or the MacroByName function in run-time. In time of opening query macro is replaced by its value.

See Also

- [TMacro](#)
- [MacroByName](#)

5.8.1.1.2.9 SQL Property

Used to get or set script text.

Class

[TDAScript](#)

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to get or set script text.

5.8.1.1.2.10 StartLine Property

Used to get the current statement start line number in a script.

Class

[TDAScript](#)

Syntax

```
property StartLine: Int64;
```

Remarks

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

5.8.1.1.2.11 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.

5.8.1.1.2.12 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.

5.8.1.1.2.13 Statements Property

Contains a list of statements obtained from the SQL property.

Class

[TDAScript](#)

Syntax

```
property Statements: TDASentences;
```

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:

```
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;
    end;
end;
```

See Also

- [TDAStatements](#)

5.8.1.1.3 Methods

Methods of the **TDAScript** class.

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

Public

Name	Description
BreakExec	Stops script execution.
ErrorOffset	Used to get the offset of the statement if the Execute method raised an exception.
Execute	Executes a script.
ExecuteFile	Executes SQL statements contained in a file.
ExecuteNext	Executes the next statement in the script and then stops.
ExecuteStream	Executes SQL statements contained in a stream object.
MacroByName	Finds a Macro with the name passed in Name.

See Also

- [TDAScript Class](#)

- [TDA Script Class Members](#)

5.8.1.1.3.1 BreakExec Method

Stops script execution.

Class

[TDA Script](#)

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to stop script execution.

5.8.1.1.3.2 ErrorOffset Method

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

Class

[TDA Script](#)

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](#)

5.8.1.1.3.3 Execute Method

Executes a script.

Class

[TDA Script](#)

Syntax

```
procedure Execute; virtual;
```

Remarks

Call the Execute method to execute a script. If PostgreSQL raises an error, the OnError event occurs.

See Also

- [ExecuteNext](#)
- [OnError](#)
- [ErrorOffset](#)

5.8.1.1.3.4 ExecuteFile Method

Executes SQL statements contained in a file.

Class

[TDAScript](#)

Syntax

```
procedure ExecuteFile(const FileName: string);
```

Parameters

FileName

Holds the file name.

Remarks

Call the ExecuteFile method to execute SQL statements contained in a file. Script doesn't load full content into memory. Reading and execution is performed by blocks of 64k size. Therefore, it is optimal to use it for big files.

5.8.1.1.3.5 ExecuteNext Method

Executes the next statement in the script and then stops.

Class

[TDAScript](#)

Syntax

```
function ExecuteNext: boolean; virtual;
```

Return Value

True, if there are any statements left in the script, False otherwise.

Remarks

Use the ExecuteNext method to execute the next statement in the script statement and stop. If PostgreSQL raises an error, the OnError event occurs.

See Also

- [Execute](#)
- [OnError](#)
- [ErrorOffset](#)

5.8.1.1.3.6 ExecuteStream Method

Executes SQL statements contained in a stream object.

Class

[TDAStream](#)

Syntax

```
procedure ExecuteStream(Stream: TStream);
```

Parameters

Stream

Holds the stream object from which the statements will be executed.

Remarks

Call the ExecuteStream method to execute SQL statements contained in a stream object. Reading from the stream and execution is performed by blocks of 64k size.

5.8.1.1.3.7 MacroByName Method

Finds a Macro with the name passed in Name.

Class

[TDAStream](#)

Syntax

```
function MacroByName(Name: string): TMacro;
```


Parameters

Name

Holds the name of the Macro to search for.

Return Value

the Macro, if a match was found.

Remarks

Call the `MacroByName` method to find a Macro with the name passed in `Name`. If a match was found, `MacroByName` returns the Macro. 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 `FindMacro` method.

To assign the value of macro use the [TMacro.Value](#) property.

See Also

- [TMacro](#)
- [Macros](#)
- `M:Devart.Dac.TDAScript.FindMacro(System.String)`

5.8.1.1.4 Events

Events of the **TDAScript** class.

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

Published

Name	Description
AfterExecute	Occurs after a SQL script execution.
BeforeExecute	Occurs when taking a specific action before executing the current SQL statement is needed.
OnError	Occurs when PostgreSQL raises an error.

See Also

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

5.8.1.1.4.1 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](#)

5.8.1.1.4.2 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.

5.8.1.1.4.3 OnError Event

Occurs when PostgreSQL raises an error.

Class

[TDAScript](#)

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

Occurs when PostgreSQL raises an error.

Action indicates the action to take when the OnError handler exits. On entry into the handler, Action is always set to eaFail.

See Also

- [ErrorOffset](#)

5.8.1.2 TDASentence 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 [TDASentence](#) members.

Unit

[DAScript](#)

Syntax

```
TDASentence = class(TCollectionItem);
```

Remarks

DAScript contains SQL statements, represented as TDASentence objects. The TDASentence class has attributes and methods for controlling single SQL statement of a script.

See Also

- [DAScript](#)
- [TDASentences](#)

5.8.1.2.1 Members

[TDASentence](#) class overview.

Properties

Name	Description
EndLine	Used to determine the number of the last statement line in a script.
EndOffset	Used to get the offset in the last line of the statement.

EndPos	Used to get the end position of the statement in a script.
Omit	Used to avoid execution of a statement.
Params	Contains parameters for an SQL statement.
Script	Used to determine the TDA Script object the SQL Statement belongs to.
SQL	Used to get or set the text of an SQL statement.
StartLine	Used to determine the number of the first statement line in a script.
StartOffset	Used to get the offset in the first line of a statement.
StartPos	Used to get the start position of the statement in a script.

Methods

Name	Description
Execute	Executes a statement.

5.8.1.2.2 Properties

Properties of the **TDAStatement** class.

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

Public

Name	Description
EndLine	Used to determine the number of the last statement line in a script.
EndOffset	Used to get the offset in the last line of the statement.

EndPos	Used to get the end position of the statement in a script.
Omit	Used to avoid execution of a statement.
Params	Contains parameters for an SQL statement.
Script	Used to determine the TDA Script object the SQL Statement belongs to.
SQL	Used to get or set the text of an SQL statement.
StartLine	Used to determine the number of the first statement line in a script.
StartOffset	Used to get the offset in the first line of a statement.
StartPos	Used to get the start position of the statement in a script.

See Also

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

5.8.1.2.2.1 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.

5.8.1.2.2.2 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.

5.8.1.2.2.3 EndPos Property

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

Class

[TDASTatement](#)

Syntax

```
property EndPos: integer;
```

Remarks

Use the EndPos property to get the end position of the statement (the position of the last character in the statement) in a script.

5.8.1.2.2.4 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.

5.8.1.2.2.5 Params Property

Contains parameters for an SQL statement.

Class

[TDAStatement](#)

Syntax

```
property Params: TDAParams;
```

Remarks

Contains parameters for an SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically. Params is a zero-based array of parameter records. Index specifies the array element to access.

See Also

- [TDAParam](#)

5.8.1.2.2.6 Script Property

Used to determine the TDA Script object the SQL Statement belongs to.

Class

[TDAStatement](#)

Syntax

```
property Script: TDA Script;
```

Remarks

Use the Script property to determine the TDA Script object the SQL Statement belongs to.

5.8.1.2.2.7 SQL Property

Used to get or set the text of an SQL statement.

Class

[TDAStatement](#)

Syntax

```
property SQL: string;
```

Remarks

Use the SQL property to get or set the text of an SQL statement.

5.8.1.2.2.8 StartLine Property

Used to determine the number of the first statement line in a script.

Class

[TDASstatement](#)

Syntax

```
property StartLine: integer;
```

Remarks

Use the StartLine property to determine the number of the first statement line in a script.

5.8.1.2.2.9 StartOffset Property

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

Class

[TDASstatement](#)

Syntax

```
property startOffset: integer;
```

Remarks

Use the StartOffset property to get the offset in the first line of a statement.

5.8.1.2.2.10 StartPos Property

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

Class

[TDASstatement](#)

Syntax

```
property StartPos: integer;
```


Remarks

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

5.8.1.2.3 Methods

Methods of the **TDASstatement** class.

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

Public

Name	Description
Execute	Executes a statement.

See Also

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

5.8.1.2.3.1 Execute Method

Executes a statement.

Class

[TDASstatement](#)

Syntax

```
procedure Execute;
```

Remarks

Use the Execute method to execute a statement.

5.8.1.3 TDASstatements Class

Holds a collection of [TDASstatement](#) objects.

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

Unit

[DAScript](#)

Syntax

```
TDAStatements = class(TCollection);
```

Remarks

Each TDAStatements holds a collection of [TDASStatement](#) 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](#)
- [TDASStatement](#)

5.8.1.3.1 Members

[TDAStatements](#) class overview.

Properties

Name	Description
Items	Used to access separate script statements.

5.8.1.3.2 Properties

Properties of the **TDAStatements** class.

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

Public

Name	Description
Items	Used to access separate script statements.

See Also

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

5.8.1.3.2.1 Items Property(Indexer)

Used to access separate script statements.

Class

[TDAStatements](#)

Syntax

```
property Items[Index: Integer]: TDASatement; 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 [TDASatement](#).

See Also

- [TDASatement](#)

5.8.2 Types

Types in the **DAScript** unit.

Types

Name	Description
TAfterStatementExecuteEvent	This type is used for the TDAScript.AfterExecute event.
TBeforeStatementExecuteEvent	This type is used for the TDAScript.BeforeExecute event.
TOnErrorEvent	This type is used for the TDAScript.OnError event.

5.8.2.1 TAfterStatementExecuteEvent Procedure Reference

This type is used for the [TDAScript.AfterExecute](#) event.

Unit

[DAScript](#)

Syntax

```
TAfterStatementExecuteEvent = procedure (Sender: TObject; SQL:  
string) of object;
```

Parameters

Sender

An object that raised the event.

SQL

Holds the passed SQL statement.

5.8.2.2 TBeforeStatementExecuteEvent Procedure Reference

This type is used for the [TDAScript.BeforeExecute](#) event.

Unit

[DAScript](#)

Syntax

```
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.

5.8.2.3 TOnErrorEvent Procedure Reference

This type is used for the [TDAScript.OnError](#) event.

Unit

[DAScript](#)

Syntax

```
TOnErrorEvent = procedure (Sender: TObject; E: Exception; SQL:  
string; var Action: TErrorAction) of object;
```

Parameters

Sender

An object that raised the event.

E

The error code.

SQL

Holds the passed SQL statement.

Action

The action to take when the OnError handler exits.

5.8.3 Enumerations

Enumerations in the **DAScript** unit.

Enumerations

Name	Description
TErrorAction	Indicates the action to take when the OnError handler exits.

5.8.3.1 TErrorAction Enumeration

Indicates the action to take when the OnError handler exits.

Unit

[DAScript](#)

Syntax

```
TErrorAction = (eaAbort, eaFail, eaException, eaContinue);
```

Values

Value	Meaning
eaAbort	Abort execution without displaying an error message.
eaContinue	Continue execution.
eaException	In Delphi 6 and higher exception is handled by the Application.HandleException method.
eaFail	Abort execution and display an error message.

5.9 DASQLMonitor

This unit contains the base class for the TPgSQLMonitor component.

Classes

Name	Description
TCustomDASQLMonitor	A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.
TDBMonitorOptions	This class holds options for dbMonitor.

Types

Name	Description
TDATraceFlags	Represents the set of TDATraceFlag .
TMonitorOptions	Represents the set of TMonitorOption .
TOnSQLEvent	This type is used for the TCustomDASQLMonitor.OnSQL event.

Enumerations

Name	Description
TDATraceFlag	Use TraceFlags to specify which database operations the monitor should track in an application at runtime.
TMonitorOption	Used to define where information from SQLMonitor will be dispalyed.

5.9.1 Classes

Classes in the **DASQLMonitor** unit.

Classes

Name	Description
------	-------------

TCustomDASQLMonitor	A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.
TDBMonitorOptions	This class holds options for dbMonitor.

5.9.1.1 TCustomDASQLMonitor Class

A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.

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

Unit

[DASQLMonitor](#)

Syntax

```
TCustomDASQLMonitor = class(TComponent);
```

Remarks

TCustomDASQLMonitor is a base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively. TCustomDASQLMonitor provides two ways of displaying debug information. It monitors either by dialog window or by Borland's proprietary SQL Monitor. Furthermore to receive debug information use the [TCustomDASQLMonitor.OnSQL](#) event.

In applications use descendants of TCustomDASQLMonitor.

5.9.1.1.1 Members

[TCustomDASQLMonitor](#) class overview.

Properties

Name	Description
Active	Used to activate monitoring of SQL.
DBMonitorOptions	Used to set options for dbMonitor.
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.

Events

Name	Description
OnSQL	Occurs when tracing of SQL activity on database components is needed.

5.9.1.1.2 Properties

Properties of the **TCustomDASQLMonitor** class.

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

Public

Name	Description
Active	Used to activate monitoring of SQL.
DBMonitorOptions	Used to set options for dbMonitor.
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](#)

5.9.1.1.2.1 Active Property

Used to activate monitoring of SQL.

Class

[TCustomDASQLMonitor](#)

Syntax

```
property Active: boolean default True;
```

Remarks

Set the Active property to True to activate monitoring of SQL.

See Also

- [OnSQL](#)

5.9.1.1.2.2 DBMonitorOptions Property

Used to set options for dbMonitor.

Class

[TCustomDASQLMonitor](#)

Syntax

```
property DBMonitorOptions: TDBMonitorOptions;
```

Remarks

Use DBMonitorOptions to set options for dbMonitor.

5.9.1.1.2.3 Options Property

Used to include the desired properties for TCustomDASQLMonitor.

Class

[TCustomDASQLMonitor](#)

Syntax

```
property Options: TMonitorOptions default [moDialog,  
moSQLMonitor, moDBMonitor, moCustom];
```

Remarks

Set Options to include the desired properties for TCustomDASQLMonitor.

See Also

- [OnSQL](#)

5.9.1.1.2.4 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](#)

5.9.1.1.3 Events

Events of the **TCustomDASQLMonitor** class.

For a complete list of the **TCustomDASQLMonitor** class members, see the

[TCustomDASQLMonitor Members](#) topic.

Public

Name	Description
OnSQL	Occurs when tracing of SQL activity on database components is needed.

See Also

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

5.9.1.1.3.1 OnSQL Event

Occurs when tracing of SQL activity on database components is needed.

Class

[TCustomDASQLMonitor](#)

Syntax

```
property OnSQL: TOnSQLEvent;
```

Remarks

Write the OnSQL event handler to let an application trace SQL activity on database components. The Text parameter holds the detected SQL statement. Use the Flag parameter to make selective processing of SQL in the handler body.

See Also

- [TraceFlags](#)

5.9.1.2 TDBMonitorOptions Class

This class holds options for dbMonitor.

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

Unit

[DASQLMonitor](#)

Syntax

```
TDBMonitorOptions = class(TPersistent);
```

5.9.1.2.1 Members

[TDBMonitorOptions](#) class overview.

Properties

Name	Description
Host	Used to set the host name or IP address of the computer where dbMonitor application runs.
Port	Used to set the port number for connecting to dbMonitor.
ReconnectTimeout	Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.
SendTimeout	Used to set timeout for sending events to dbMonitor.

5.9.1.2.2 Properties

Properties of the **TDBMonitorOptions** class.

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

Published

Name	Description
Host	Used to set the host name or IP address of the computer where dbMonitor application runs.
Port	Used to set the port number for connecting to dbMonitor.
ReconnectTimeout	Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.
SendTimeout	Used to set timeout for sending events to dbMonitor.

See Also

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

5.9.1.2.2.1 Host Property

Used to set the host name or IP address of the computer where dbMonitor application runs.

Class

[TDBMonitorOptions](#)

Syntax

```
property Host: string;
```

Remarks

Use the Host property to set the host name or IP address of the computer where dbMonitor application runs.

dbMonitor supports remote monitoring. You can run dbMonitor on a different computer than monitored application runs. In this case you need to set the Host property to the corresponding computer name.

5.9.1.2.2.2 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.

5.9.1.2.2.3 ReconnectTimeout Property

Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.

Class

[TDBMonitorOptions](#)

Syntax

```
property ReconnectTimeout: integer default  
DefaultReconnectTimeout;
```

Remarks

Use the ReconnectTimeout property to set the minimum time (in milliseconds) that should be spent before allowing reconnecting to dbMonitor. If an error occurs when the component sends an event to dbMonitor (dbMonitor is not running), next events are ignored and the component does not restore the connection until ReconnectTimeout is over.

5.9.1.2.2.4 SendTimeout Property

Used to set timeout for sending events to dbMonitor.

Class

[TDBMonitorOptions](#)

Syntax

```
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.

5.9.2 Types

Types in the **DASQLMonitor** unit.

Types

Name	Description
TDATraceFlags	Represents the set of TDATraceFlag .
TMonitorOptions	Represents the set of TMonitorOption .
TOnSQLEvent	This type is used for the TCustomDASQLMonitor.OnSQL event.

5.9.2.1 TDATraceFlags Set

Represents the set of [TDATraceFlag](#).

Unit

[DASQLMonitor](#)

Syntax

```
TDATraceFlags = set of TDATraceFlag;
```

5.9.2.2 TMonitorOptions Set

Represents the set of [TMonitorOption](#).

Unit

[DASQLMonitor](#)

Syntax

```
TMonitorOptions = set of TMonitorOption;
```

5.9.2.3 TOnSQLEvent Procedure Reference

This type is used for the [TCustomDASQLMonitor.OnSQL](#) event.

Unit

[DASQLMonitor](#)

Syntax

```
TOnSQLEvent = procedure (Sender: TObject; Text: string; Flag:  
TDATraceFlag) of object;
```

Parameters

Sender

An object that raised the event.

Text

Holds the detected SQL statement.

Flag

Use the Flag parameter to make selective processing of SQL in the handler body.

5.9.3 Enumerations

Enumerations in the **DASQLMonitor** unit.

Enumerations

Name	Description
TDATraceFlag	Use TraceFlags to specify which database operations the monitor should track in an application at runtime.
TMonitorOption	Used to define where information from SQLMonitor will be dispalyed.

5.9.3.1 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

Value	Meaning
tfBlob	This option is declared for future use.
tfConnect	Establishing a connection.
tfError	Errors of query execution.
tfMisc	This option is declared for future use.
tfObjDestroy	Destroying of components.
tfParams	Representing parameter values for tfQPrepare and tfQExecute.
tfPool	Connection pool operations.
tfQExecute	Execution of the queries.
tfQFetch	This option is declared for future use.
tfQPrepare	Queries preparation.
tfService	This option is declared for future use.
tfStmt	This option is declared for future use.
tfTransact	Processing transactions.

5.9.3.2 TMonitorOption Enumeration

Used to define where information from SQLMonitor will be displayed.

Unit

[DASQLMonitor](#)

Syntax

```
TMonitorOption = (moDialog, moSQLMonitor, moDBMonitor, moCustom,
moHandled);
```

Values

Value	Meaning
moCustom	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.
moDBMonitor	Debug information is displayed in DBMonitor .

moDialog	Debug information is displayed in debug window.
moHandled	Component handle is included into the event description string.
moSQLMonitor	Debug information is displayed in Borland SQL Monitor.

5.10 DBAccess

This unit contains base classes for most of the components.

Classes

Name	Description
<u>EDAEError</u>	A base class for exceptions that are raised when an error occurs on the server side.
<u>TCRDataSource</u>	Provides an interface between a DAC dataset components and data-aware controls on a form.
<u>TCustomConnectDialog</u>	A base class for the connect dialog components.
<u>TCustomDACConnection</u>	A base class for components used to establish connections.
<u>TCustomDADataset</u>	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
<u>TCustomDASQL</u>	A base class for components executing SQL statements that do not return result sets.
<u>TCustomDAUpdateSQL</u>	A base class for components that provide DML statements for more flexible control over data modifications.
<u>TDACondition</u>	Represents a condition from the <u>TDAConditions</u> list.
<u>TDAConditions</u>	Holds a collection of <u>TDACondition</u> objects.
<u>TDACConnectionOptions</u>	This class allows setting up the behaviour of the TDACConnection class.
<u>TDADatasetOptions</u>	This class allows setting up the behaviour of the TDADataset class.

TDAEncryption	Used to specify the options of the data encryption in a dataset.
TDAMapRule	Class that forms rules for Data Type Mapping.
TDAMapRules	Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.
TDAMetaData	A class for retrieving metainformation of the specified database objects in the form of dataset.
TDAParam	A class that forms objects to 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

Name	Description
TAfterExecuteEvent	This type is used for the TCustomDADataset.AfterExecute and TCustomDASQL.AfterExecute events.
TAfterFetchEvent	This type is used for the TCustomDADataset.AfterFetch event.

TBeforeFetchEvent	This type is used for the TCustomDADataset.BeforeFetch event.
TConnectionLostEvent	This type is used for the TCustomDAConnection.OnConnecti onLost event.
TDAConnectionErrorEvent	This type is used for the TCustomDAConnection.OnError event.
TDATransactionErrorEvent	This type is used for the TDATransaction.OnError event.
TRefreshOptions	Represents the set of TRefreshOption .
TUpdateExecuteEvent	This type is used for the TCustomDADataset.AfterUpdateEx ecute and TCustomDADataset.BeforeUpdateE xecute events.

Enumerations

Name	Description
TLabelSet	Sets the language of labels in the connect dialog.
TRefreshOption	Indicates when the editing record will be refreshed.
TRetryMode	Specifies the application behavior when connection is lost.

Variables

Name	Description
BaseSQLOldBehavior	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.
ChangeCursor	When set to True allows data access components to change screen cursor for the execution time.

[SQLGeneratorCompatibility](#)

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.

5.10.1 Classes

Classes in the **DBAccess** unit.

Classes

Name	Description
EDAEError	A base class for exceptions that are raised when an error occurs on the server side.
TCRDataSource	Provides an interface between a DAC dataset components and data-aware controls on a form.
TCustomConnectDialog	A base class for the connect dialog components.
TCustomDACConnection	A base class for components used to establish connections.
TCustomDADataset	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
TCustomDASQL	A base class for components executing SQL statements that do not return result sets.
TCustomDAUpdateSQL	A base class for components that provide DML statements for more flexible control over data modifications.
TDACondition	Represents a condition from the TDAConditions list.
TDAConditions	Holds a collection of TDACondition objects.

TDAConnectionOptions	This class allows setting up the behaviour of the TDAConnection class.
TDADatasetOptions	This class allows setting up the behaviour of the TDADataset class.
TDAEncryption	Used to specify the options of the data encryption in a dataset.
TDAMapRule	Class that forms rules for Data Type Mapping.
TDAMapRules	Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.
TDAMetaData	A class for retrieving metainformation of the specified database objects in the form of dataset.
TDAParam	A class that forms objects to 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.

5.10.1.1 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

```
EDAEError = class(EDatabaseError);
```

Remarks

EDAEError is a base class for exceptions that are raised when an error occurs on the server side.

5.10.1.1.1 Members

[EDAEError](#) class overview.

Properties

Name	Description
Component	Contains the component that caused the error.
ErrorCode	Determines the error code returned by the server.

5.10.1.1.2 Properties

Properties of the **EDAEError** class.

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

Public

Name	Description
Component	Contains the component that caused the error.
ErrorCode	Determines the error code returned by the server.

See Also

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

5.10.1.1.2.1 Component Property

Contains the component that caused the error.

Class

[EDAEError](#)

Syntax

```
property Component: TObject;
```

Remarks

The Component property contains the component that caused the error.

5.10.1.1.2.2 ErrorCode Property

Determines the error code returned by the server.

Class

[EDAEError](#)

Syntax

```
property ErrorCode: integer;
```

Remarks

Use the ErrorCode property to determine the error code returned by PostgreSQL. This value is always positive.

5.10.1.2 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.

5.10.1.2.1 Members

[TCRDataSource](#) class overview.

5.10.1.3 TCustomConnectDialog Class

A base class for the connect dialog components.

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

Unit

[DBAccess](#)

Syntax

```
TCustomConnectDialog = class(TComponent);
```

Remarks

TCustomConnectDialog is a base class for the connect dialog components. It provides functionality to show a dialog box where user can edit username, password and server name before connecting to a database. You can customize captions of buttons and labels by their properties.

5.10.1.3.1 Members

[TCustomConnectDialog](#) class overview.

Properties

Name	Description
CancelButton	Used to specify the label for the Cancel button.
Caption	Used to set the caption of dialog box.
ConnectButton	Used to specify the label for the Connect button.

DialogClass	Used to specify the class of the form that will be displayed to enter login information.
LabelSet	Used to set the language of buttons and labels captions.
PasswordLabel	Used to specify a prompt for password edit.
Retries	Used to indicate the number of retries of failed connections.
SavePassword	Used for the password to be displayed in ConnectDialog in asterisks.
ServerLabel	Used to specify a prompt for the server name edit.
StoreLogInfo	Used to specify whether the login information should be kept in system registry after a connection was established.
UsernameLabel	Used to specify a prompt for username edit.

Methods

Name	Description
Execute	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.
GetServerList	Retrieves a list of available server names.

5.10.1.3.2 Properties

Properties of the **TCustomConnectDialog** class.

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

Public

Name	Description
------	-------------

CancelButton	Used to specify the label for the Cancel button.
Caption	Used to set the caption of dialog box.
ConnectButton	Used to specify the label for the Connect button.
DialogClass	Used to specify the class of the form that will be displayed to enter login information.
LabelSet	Used to set the language of buttons and labels captions.
PasswordLabel	Used to specify a prompt for password edit.
Retries	Used to indicate the number of retries of failed connections.
SavePassword	Used for the password to be displayed in ConnectDialog in asterisks.
ServerLabel	Used to specify a prompt for the server name edit.
StoreLogInfo	Used to specify whether the login information should be kept in system registry after a connection was established.
UsernameLabel	Used to specify a prompt for username edit.

See Also

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

5.10.1.3.2.1 CancelButton Property

Used to specify the label for the Cancel button.

Class

[TCustomConnectDialog](#)

Syntax

```
property CancelButton: string;
```

Remarks

Use the CancelButton property to specify the label for the Cancel button.

5.10.1.3.2.2 Caption Property

Used to set the caption of dialog box.

Class

[TCustomConnectDialog](#)

Syntax

```
property Caption: string;
```

Remarks

Use the Caption property to set the caption of dialog box.

5.10.1.3.2.3 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.

5.10.1.3.2.4 DialogClass Property

Used to specify the class of the form that will be displayed to enter login information.

Class

[TCustomConnectDialog](#)

Syntax

```
property DialogClass: string;
```

Remarks

Use the DialogClass property to specify the class of the form that will be displayed to enter login information. When this property is blank, TCustomConnectDialog uses the default form - TConnectForm. You can write your own login form to enter login information and assign its class name to the DialogClass property. Each login form must have ConnectDialog: TCustomConnectDialog published property to access connection information. For details see the implementation of the connect form which sources are in the Lib subdirectory of the PgDAC installation directory.

See Also

- [GetServerList](#)

5.10.1.3.2.5 LabelSet Property

Used to set the language of buttons and labels captions.

Class

[TCustomConnectDialog](#)

Syntax

```
property LabelSet: TLabelSet default IsEnglish;
```

Remarks

Use the LabelSet property to set the language of labels and buttons captions. The default value is IsEnglish.

5.10.1.3.2.6 PasswordLabel Property

Used to specify a prompt for password edit.

Class

[TCustomConnectDialog](#)

Syntax

```
property PasswordLabel: string;
```

Remarks

Use the PasswordLabel property to specify a prompt for password edit.

5.10.1.3.2.7 Retries Property

Used to indicate the number of retries of failed connections.

Class

[TCustomConnectDialog](#)

Syntax

```
property Retries: word default 3;
```

Remarks

Use the Retries property to determine the number of retries of failed connections.

5.10.1.3.2.8 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.

5.10.1.3.2.9 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.

5.10.1.3.2.10 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.

5.10.1.3.2.11 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.

5.10.1.3.3 Methods

Methods of the **TCustomConnectDialog** class.

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

Public

Name	Description
Execute	Displays the connect dialog and calls the connection's Connect method

	when user clicks the Connect button.
GetServerList	Retrieves a list of available server names.

See Also

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

5.10.1.3.3.1 Execute Method

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.

Class

[TCustomConnectDialog](#)

Syntax

```
function Execute: boolean; virtual;
```

Return Value

True, if connected.

Remarks

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button. Returns True if connected. If user clicks Cancel, Execute returns False. In the case of failed connection Execute offers to connect repeat [Retries](#) times.

5.10.1.3.3.2 GetServerList Method

Retrieves a list of available server names.

Class

[TCustomConnectDialog](#)

Syntax

```
procedure GetServerList(List: TStrings); virtual;
```

Parameters

List

Holds a list of available server names.

Remarks

Call the `GetServerList` method to retrieve a list of available server names. It is particularly relevant for writing custom login form.

See Also

- [DialogClass](#)

5.10.1.4 TCustomDAConnection Class

A base class for components used to establish connections.

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

Unit

[DBAccess](#)

Syntax

```
TCustomDAConnection = class(TCustomConnection);
```

Remarks

TCustomDAConnection is a base class for components that establish connection with database, provide customised login support, and perform transaction control.

Do not create instances of TCustomDAConnection. To add a component that represents a connection to a source of data, use descendants of the TCustomDAConnection class.

5.10.1.4.1 Members

[TCustomDAConnection](#) class overview.

Properties

Name	Description
ConnectDialog	Allows to link a TCustomConnectDialog component.
ConnectionString	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL	Allows customizing line breaks in string fields and parameters.
InTransaction	Indicates whether the transaction is active.

LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the connection behavior.
Password	Serves to supply a password for login.
Pooling	Enables or disables using connection pool.
PoolingOptions	Specifies the behaviour of connection pool.
Server	Serves to supply the server name for login.
Username	Used to supply a user name for login.

Methods

Name	Description
ApplyUpdates	Overloaded. Applies changes in datasets.
Commit	Commits current transaction.
Connect	Establishes a connection to the server.
CreateSQL	Creates a component for queries execution.
Disconnect	Performs disconnect.
ExecProc	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx	Allows to execute a stored procedure or function.
ExecSQL	Executes a SQL statement with parameters.

ExecSQLEx	Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames	Returns a database list from the server.
GetKeyFieldNames	Provides a list of available key field names.
GetStoredProcNames	Returns a list of stored procedures from the server.
GetTableNames	Provides a list of available tables names.
MonitorMessage	Sends a specified message through the TCustomDASQLMonitor component.
Ping	Used to check state of connection to the server.
RemoveFromPool	Marks the connection that should not be returned to the pool after disconnect.
Rollback	Discards all current data changes and ends transaction.
StartTransaction	Begins a new user transaction.

Events

Name	Description
OnConnectionLost	This event occurs when connection was lost.
OnError	This event occurs when an error has arisen in the connection.

5.10.1.4.2 Properties

Properties of the **TCustomDAConnection** class.

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

Public

Name	Description
ConnectDialog	Allows to link a TCustomConnectDialog component.
ConnectionString	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL	Allows customizing line breaks in string fields and parameters.
InTransaction	Indicates whether the transaction is active.
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the connection behavior.
Password	Serves to supply a password for login.
Pooling	Enables or disables using connection pool.
PoolingOptions	Specifies the behaviour of connection pool.
Server	Serves to supply the server name for login.
Username	Used to supply a user name for login.

See Also

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

5.10.1.4.2.1 ConnectDialog Property

Allows to link a [TCustomConnectDialog](#) component.

Class

[TCustomDAConnection](#)

Syntax

property ConnectDialog: [TCustomConnectDialog](#);

Remarks

Use the ConnectDialog property to assign to connection a [TCustomConnectDialog](#) component.

See Also

- [TCustomConnectDialog](#)

5.10.1.4.2.2 ConnectString Property

Used to specify the connection information, such as: UserName, Password, Server, etc.

Class

[TCustomDAConnection](#)

Syntax

property ConnectString: **string stored** False;

Remarks

PgDAC recognizes an ODBC-like syntax in provider string property values. Within the string, elements are delimited by using a semicolon. Each element consists of a keyword, an equal sign character, and the value passed on initialization. For example:

```
Server=London1;User ID=nancyd
```

Connection parameters

The following connection parameters can be used to customize connection:

Parameter Name	Description
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Pooling	Enables or disables using connection pool.
ConnectionLifeTime	Used to specify the maximum time during which an opened connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of

	connections that can be opened in connection pool.
Validate Connection	Used for a connection to be validated when it is returned from the pool.
Server	Serves to supply the server name for login.
Username	Used to supply a user name for login.
Password	Used to supply a user name for login.
Database	Used to set the name of the database to associate with TPgConnection component.
Charset	Used to set the character set that PgDAC uses to read and write character data.
UseUnicode	Used to enable or disable Unicode support.
Port	Used to specify the port number for the connection.
ConnectionTimeout	Used to specify the amount of time before an attempt to make a connection is considered unsuccessful.
ProtocolVersion	Used to set the version of protocol for communication with PostgreSQL server.
Schema	Used to change the search path of the connection to the specified schema, or get the first value from the search path.
IPVersion	Used to specify the version of the Internet Protocol.
CACert	Holds the pathname to the certificate authority file.
Cert	Holds the pathname to the certificate file.
CipherList	Holds the list of allowed ciphers to use for SSL encryption.
Key	Holds the pathname to the key file.
Mode	Used to determine whether or with what priority an SSL connection will be negotiated with the server.

See Also

- [Password](#)
- [Username](#)
- [Server](#)
- [Connect](#)

5.10.1.4.2.3 ConvertEOL Property

Allows customizing line breaks in string fields and parameters.

Class

[TCustomDAConnection](#)

Syntax

```
property ConvertEOL: boolean default False;
```

Remarks

Affects the line break behavior in string fields and parameters. When fetching strings (including the TEXT fields) with ConvertEOL = True, dataset converts their line breaks from the LF to CRLF form. And when posting strings to server with ConvertEOL turned on, their line breaks are converted from CRLF to LF form. By default, strings are not converted.

5.10.1.4.2.4 InTransaction Property

Indicates whether the transaction is active.

Class

[TCustomDAConnection](#)

Syntax

```
property InTransaction: boolean;
```

Remarks

Examine the InTransaction property at runtime to determine whether user transaction is currently in progress. In other words InTransaction is set to True when user explicitly calls [StartTransaction](#). Calling [Commit](#) or [Rollback](#) sets InTransaction to False. The value of the InTransaction property cannot be changed directly.

See Also

- [StartTransaction](#)
- [Commit](#)
- [Rollback](#)

5.10.1.4.2.5 LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Class

[TCustomDAConnection](#)

Syntax

```
property LoginPrompt default DefValLoginPrompt;
```

Remarks

Specifies whether a login dialog appears immediately before opening a new connection. If [ConnectDialog](#) is not specified, the default connect dialog will be shown. The connect dialog will appear only if the PgDacVcl unit appears to the uses clause.

5.10.1.4.2.6 Options Property

Specifies the connection behavior.

Class

[TCustomDAConnection](#)

Syntax

```
property Options: TDACConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of the connection.
Descriptions of all options are in the table below.

Option Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.
DefaultSortType	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.
DisconnectedMode	Used to open a connection only when needed for performing a server call and closes after performing the operation.

KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover	If True, the OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

See Also

- [Disconnected Mode](#)
- [Working in an Unstable Network](#)

5.10.1.4.2.7 Password Property

Serves to supply a password for login.

Class

[TCustomDAConnection](#)

Syntax

```
property Password: string stored False;
```

Remarks

Use the Password property to supply a password to handle server's request for a login.

Warning: Storing hard-coded user name and password entries as property values or in code for the OnLogin event handler can compromise server security.

See Also

- [Username](#)
- [Server](#)

5.10.1.4.2.8 Pooling Property

Enables or disables using connection pool.

Class

[TCustomDAConnection](#)

Syntax

```
property Pooling: boolean default DefValPooling;
```


Remarks

Normally, when TCustomDAConnection establishes connection with the server it takes server memory and time resources for allocating new server connection. For example, pooling can be very useful when using disconnect mode. If an application has wide user activity that forces many connect/disconnect operations, it may spend a lot of time on creating connection and sending requests to the server. TCustomDAConnection has software pool which stores open connections with identical parameters.

Connection pool uses separate thread that validates the pool every 30 seconds. Pool validation consists of checking each connection in the pool. If a connection is broken due to a network problem or another reason, it is deleted from the pool. The validation procedure removes also connections that are not used for a long time even if they are valid from the pool.

Set Pooling to True to enable pooling. Specify correct values for PoolingOptions. Two connections belong to the same pool if they have identical values for the parameters:

[MinPoolSize](#), [MaxPoolSize](#), [Validate](#), [ConnectionLifeTime](#)

Note: Using Pooling := True can cause errors with working with temporary tables.

See Also

- [Username](#)
- [Password](#)
- [PoolingOptions](#)
- [Connection Pooling](#)

5.10.1.4.2.9 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.

Option Name	Description
-------------	-------------

ConnectionLifetime	Used to specify the maximum time during which an opened connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of connections that can be opened in the connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

See Also

- [Pooling](#)

5.10.1.4.2.10 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.

See Also

- [Username](#)
- [Password](#)

5.10.1.4.2.11 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, PgDAC tries to connect with the user name.

Warning: Storing hard-coded user name and password entries as property values or in code for the OnLogin event handler can compromise server security.

See Also

- [Password](#)
- [Server](#)

5.10.1.4.3 Methods

Methods of the **TCustomDACConnection** class.

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

Public

Name	Description
ApplyUpdates	Overloaded. Applies changes in datasets.
Commit	Commits current transaction.
Connect	Establishes a connection to the server.
CreateSQL	Creates a component for queries execution.
Disconnect	Performs disconnect.
ExecProc	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx	Allows to execute a stored procedure or function.
ExecSQL	Executes a SQL statement with parameters.
ExecSQLEx	Executes any SQL statement outside the TQuery or TSQL components.

GetDatabaseNames	Returns a database list from the server.
GetKeyFieldNames	Provides a list of available key field names.
GetStoredProcNames	Returns a list of stored procedures from the server.
GetTableNames	Provides a list of available tables names.
MonitorMessage	Sends a specified message through the TCustomDASQLMonitor component.
Ping	Used to check state of connection to the server.
RemoveFromPool	Marks the connection that should not be returned to the pool after disconnect.
Rollback	Discards all current data changes and ends transaction.
StartTransaction	Begins a new user transaction.

See Also

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

5.10.1.4.3.1 ApplyUpdates Method

Applies changes in datasets.

Class

[TCustomDAConnection](#)

Overload List

Name	Description
ApplyUpdates	Applies changes from all active datasets.
ApplyUpdates(const DataSets: array of TCustomDADataset)	Applies changes from the specified datasets.

Applies changes from all active datasets.

Class

[TCustomDAConnection](#)

Syntax

```
procedure ApplyUpdates; overload; virtual;
```

Remarks

Call the ApplyUpdates method to write all pending cached updates from all active datasets attached to this connection to a database or from specific datasets. The ApplyUpdates method passes cached data to the database for storage, takes care of committing or rolling back transactions, and clearing the cache when the operation is successful.

Using ApplyUpdates for connection is a preferred method of updating datasets rather than calling each individual dataset's ApplyUpdates method.

See Also

- [TMemDataSet.CachedUpdates](#)
- [TMemDataSet.ApplyUpdates](#)

Applies changes from the specified datasets.

Class

[TCustomDAConnection](#)

Syntax

```
procedure ApplyUpdates(const DataSets: array of  
TCustomDADataSet); overload; virtual;
```

Parameters

DataSets

A list of datasets changes in which are to be applied.

Remarks

Call the ApplyUpdates method to write all pending cached updates from the specified datasets. The ApplyUpdates method passes cached data to the database for storage, takes care of committing or rolling back transactions and clearing the cache when operation is successful.

Using ApplyUpdates for connection is a preferred method of updating datasets rather than

calling each individual dataset's ApplyUpdates method.

5.10.1.4.3.2 Commit Method

Commits current transaction.

Class

[TCustomDAConnection](#)

Syntax

```
procedure Commit; virtual;
```

Remarks

Call the Commit method to commit current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database and then ends the transaction. The current transaction is the last transaction started by calling StartTransaction.

See Also

- [Rollback](#)
- [StartTransaction](#)
- [TCustomPgDataSet.FetchAll](#)

5.10.1.4.3.3 Connect Method

Establishes a connection to the server.

Class

[TCustomDAConnection](#)

Syntax

```
procedure Connect; overload; procedure Connect(const  
ConnectionString: 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](#)

5.10.1.4.3.4 CreateSQL Method

Creates a component for queries execution.

Class

[TCustomDAConnection](#)

Syntax

```
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.

5.10.1.4.3.5 Disconnect Method

Performs disconnect.

Class

[TCustomDAConnection](#)

Syntax

```
procedure Disconnect;
```

Remarks

Call the Disconnect method to drop a connection to database. Before the connection component is deactivated, all associated datasets are closed. Calling Disconnect is similar to setting the Connected property to False.

In most cases, closing a connection frees system resources allocated to the connection. If user transaction is active, e.g. the [InTransaction](#) flag is set, calling to Disconnect the current user transaction.

Note: If a previously active connection is closed and then reopened, any associated datasets must be individually reopened; reopening the connection does not automatically reopen associated datasets.

See Also

- [Connect](#)

5.10.1.4.3.6 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)
Label1.Caption:= MyPgConnection1.ExecProc('My_Sum', [10, 20]);
Label2.Caption:= MyPgConnection1.ParamByName('Result').AsString;
```

See Also

- [ExecProcEx](#)
- [ExecSQL](#)
- [ExecSQLEx](#)

5.10.1.4.3.7 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
```

See Also

- [ExecSQL](#)
- [ExecSQLEx](#)
- [ExecProc](#)

5.10.1.4.3.8 ExecSQL Method

Executes a SQL statement with parameters.

Class

[TCustomDAConnection](#)

Syntax

```
function ExecSQL(const Text: string): variant;  
overload; function ExecSQL(const Text: string; const Params:  
array of variant): variant; overload; virtual;
```

Parameters

Text

a SQL statement to be executed.

Params

Array of parameter values arranged in the same order as they appear in SQL statement.

Return Value

Out parameter with the name Result will hold the result of function having data type dtString. Otherwise returns Null.

Remarks

Use the ExecSQL method to execute any SQL statement outside the [TCustomDADataSet](#) or [TCustomDASQL](#) components. Supply the Params array with the values of parameters arranged in the same order as they appear in a SQL statement which itself is passed to the Text string parameter.

See Also

- [ExecSQLEx](#)
- [ExecProc](#)

5.10.1.4.3.9 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

```
PgConnection.ExecSQLEx('begin :A:= :B + :C; end;',  
    ['A', 0, 'B', 5, 'C', 3]);  
A:= PgConnection.ParamByName('A').AsInteger;
```

See Also

- [ExecSQL](#)

5.10.1.4.3.10 GetDatabaseNames Method

Returns a database list from the server.

Class

[TCustomDAConnection](#)

Syntax

```
procedure GetDatabaseNames(List: TStrings); virtual;
```

Parameters

List

A TStringList descendant that will be filled with database names.

Remarks

Populates a string list with the names of databases.

Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetDatabaseNames.

See Also

- [GetTableNames](#)
- [GetStoredProcNames](#)

5.10.1.4.3.11 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](#)

5.10.1.4.3.12 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](#)

5.10.1.4.3.13 GetTableNames Method

Provides a list of available tables names.

Class

[TCustomDAConnection](#)

Syntax

```
procedure GetTableNames(List: TStrings; AllTables: boolean =
```

```
False; OnlyTables: boolean = False); virtual;
```

Parameters

List

A TStrings descendant that will be filled with table names.

AllTables

True, if procedure returns all table names including the names of system tables to the List parameter.

OnlyTables

Remarks

Call the GetTableNames method to get the names of available tables. Populates a string list with the names of tables in the database. If AllTables = True, procedure returns all table names including the names of system tables to the List parameter, otherwise List will not contain the names of system tables. If AllTables = True, the procedure returns to the List parameter the names of the tables that belong to all schemas; otherwise, List will contain the names of the tables that belong to the current schema.

Note: Any contents already in the target string list object are eliminated and overwritten by the data produced by GetTableNames.

See Also

- [GetDatabaseNames](#)
- [GetStoredProcNames](#)

5.10.1.4.3.14 MonitorMessage Method

Sends a specified message through the [TCustomDASQLMonitor](#) component.

Class

[TCustomDAConnection](#)

Syntax

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

Parameters

Msg

Message text that will be sent.

Remarks

Call the MonitorMessage method to output specified message via the

[TCustomDASQLMonitor](#) component.

See Also

- [TCustomDASQLMonitor](#)

5.10.1.4.3.15 Ping Method

Used to check state of connection to the server.

Class

[TCustomDAConnection](#)

Syntax

```
procedure Ping;
```

Remarks

The method is used for checking server connection state.

5.10.1.4.3.16 RemoveFromPool Method

Marks the connection that should not be returned to the pool after disconnect.

Class

[TCustomDAConnection](#)

Syntax

```
procedure RemoveFromPool;
```

Remarks

Call the RemoveFromPool method to mark the connection that should be deleted after disconnect instead of returning to the connection pool.

See Also

- [Pooling](#)
- [PoolingOptions](#)

5.10.1.4.3.17 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](#)
- [TCustomPgDataSet.FetchAll](#)

5.10.1.4.3.18 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](#)

5.10.1.4.4 Events

Events of the **TCustomDAConnection** class.

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

Public

Name	Description
OnConnectionLost	This event occurs when connection was lost.
OnError	This event occurs when an error has arisen in the connection.

See Also

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

5.10.1.4.4.1 OnConnectionLost Event

This event occurs when connection was lost.

Class

[TCustomDAConnection](#)

Syntax

```
property OnConnectionLost: TConnectionLostEvent;
```

Remarks

Write the OnConnectionLost event handler to process fatal errors and perform failover.

Note: To use the OnConnectionLost event handler, you should explicitly add the MemData unit to the 'uses' list and set the TCustomDAConnection.Options.LocalFailover property to True.

5.10.1.4.4.2 OnError Event

This event occurs when an error has arisen in the connection.

Class

[TCustomDAConnection](#)

Syntax

```
property OnError: TDAConnectionErrorEvent;
```

Remarks

Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

5.10.1.5 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

5.10.1.5.1 Members

[TCustomDADataSet](#) class overview.

Properties

Name	Description
------	-------------

<u>BaseSQL</u>	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>Conditions</u>	Used to add WHERE conditions to a query
<u>Connection</u>	Used to specify a connection object to use to connect to a data store.
<u>DataTypeMap</u>	Used to set data type mapping rules
<u>Debug</u>	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u>	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u>	Used to keep dataset opened after connection is closed.
<u>FetchRows</u>	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u>	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u>	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u>	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.

KeyFields	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
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.
MacroCount	Used to get the number of macros associated with the Macros property.
Macros	Makes it possible to change SQL queries easily.
MasterFields	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.
MasterSource	Used to specify the data source component which binds current dataset to the master one.
Options	Used to specify the behaviour of TCustomDADataset object.
ParamCheck	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount	Used to indicate how many parameters are there in the Params property.
Params	Used to view and set parameter names, values, and data types dynamically.
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.
ReadOnly	Used to prevent users from updating, inserting, or deleting data in the

	dataset.
RefreshOptions	Used to indicate when the editing record is refreshed.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SQL	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.
SQLUpdate	Used to specify a SQL statement that will be used when applying an update to a dataset.
UniDirectional	Used if an application does not need bidirectional access to records in the result set.
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

Name	Description
AddWhere	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from TMemDataSet)	Applies a range to the dataset.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec	Breaks execution of the SQL statement on the server.
CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
CreateBlobStream	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
DeleteWhere	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from TMemDataSet)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from TMemDataSet)	Enables changing the starting value for an existing range.
Execute	Overloaded. Executes a SQL statement on the server.

Executing	Indicates whether SQL statement is still being executed.
Fetched	Used to learn whether TCustomDADataset has already fetched all rows.
Fetching	Used to learn whether TCustomDADataset is still fetching rows.
FetchingAll	Used to learn whether TCustomDADataset is fetching all rows to the end.
FindKey	Searches for a record which contains specified field values.
FindMacro	Description is not available at the moment.
FindNearest	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam	Determines if a parameter with the specified name exists in a dataset.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject	Returns a multireference shared object from field.
GetFieldPrecision	Retrieves the precision of a number field.
GetFieldScale	Retrieves the scale of a number field.
GetKeyFieldNames	Provides a list of available key field names.
GetOrderBy	Retrieves an ORDER BY clause from a SQL statement.

GotoCurrent	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from TMemDataSet)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from TMemDataSet)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock	Locks the current record.
MacroByName	Finds a Macro with the name passed in Name.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare	Allocates, opens, and parses cursor for a query.
RefreshRecord	Actualizes field values for the current record.
RestoreSQL	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from TMemDataSet)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL	Saves the SQL property value to BaseSQL.
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

Name	Description
AfterExecute	Occurs after a component has executed a query to database.
AfterFetch	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.

[OnUpdateRecord](#) (inherited from [TMemDataSet](#))

Occurs when a single update component can not handle the updates.

5.10.1.5.2 Properties

Properties of the **TCustomDADataset** class.

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

Public

Name	Description
BaseSQL	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
Conditions	Used to add WHERE conditions to a query
Connection	Used to specify a connection object to use to connect to a data store.
DataTypeMap	Used to set data type mapping rules
Debug	Used to display executing statement, all its parameters' values, and the type of parameters.
DetailFields	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected	Used to keep dataset opened after connection is closed.
FetchRows	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL	Used to change the WHERE clause of SELECT statement and reopen a query.

<u>FinalSQL</u>	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u>	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u>	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u>	Used to get the number of macros associated with the Macros property.
<u>Macros</u>	Makes it possible to change SQL queries easily.
<u>MasterFields</u>	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.
<u>MasterSource</u>	Used to specify the data source component which binds current dataset to the master one.
<u>Options</u>	Used to specify the behaviour of TCustomDADataset object.
<u>ParamCheck</u>	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

ParamCount	Used to indicate how many parameters are there in the Params property.
Params	Used to view and set parameter names, values, and data types dynamically.
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.
ReadOnly	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions	Used to indicate when the editing record is refreshed.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SQL	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.
SQLUpdate	Used to specify a SQL statement that will be used when applying an update to a dataset.

UniDirectional	Used if an application does not need bidirectional access to records in the result set.
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

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

5.10.1.5.2.1 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](#)

5.10.1.5.2.2 Conditions Property

Used to add WHERE conditions to a query

Class

[TCustomDADataset](#)

Syntax

```
property Conditions: TDAConditions stored False;
```

See Also

- [TDAConditions](#)

5.10.1.5.2.3 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

[TCustomDADataset](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

5.10.1.5.2.4 DataTypeMap Property

Used to set data type mapping rules

Class

[TCustomDADataset](#)

Syntax

```
property DataTypeMap: TDAMapRules stored IsMapRulesStored;
```

See Also

- [TDAMapRules](#)

5.10.1.5.2.5 Debug Property

Used to display executing statement, all its parameters' values, and the type of parameters.

Class

[TCustomDADataset](#)

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display executing statement and all its parameters' values. Also displays the type of parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

See Also

- [TCustomDASQL.Debug](#)

5.10.1.5.2.6 DetailFields Property

Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

Class

[TCustomDADataset](#)

Syntax

```
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](#)

5.10.1.5.2.7 Disconnected Property

Used to keep dataset opened after connection is closed.

Class

[TCustomDADataset](#)

Syntax

```
property Disconnected: boolean;
```

Remarks

Set the Disconnected property to True to keep dataset opened after connection is closed.

5.10.1.5.2.8 FetchRows Property

Used to define the number of rows to be transferred across the network at the same time.

Class

[TCustomDADataset](#)

Syntax

```
property FetchRows: integer 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.

5.10.1.5.2.9 FilterSQL Property

Used to change the WHERE clause of SELECT statement and reopen a query.

Class

[TCustomDADataset](#)

Syntax

```
property FilterSQL: string;
```

Remarks

The FilterSQL property is similar to the Filter property, but it changes the WHERE clause of SELECT statement and reopens query. Syntax is the same to the WHERE clause.

Note: the FilterSQL property adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

Example

```
Query1.FilterSQL := 'Dept >= 20 and DName LIKE 'M%''';
```

See Also

- [AddWhere](#)

5.10.1.5.2.10 FinalSQL Property

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Class

[TCustomDADataset](#)

Syntax

```
property FinalSQL: string;
```

Remarks

Use FinalSQL to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros. This is the exact statement that will be passed on to the database server.

See Also

- [FinalSQL](#)
- [AddWhere](#)
- [SaveSQL](#)
- [SQLSaved](#)

- [RestoreSQL](#)
- [BaseSQL](#)

5.10.1.5.2.11 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.

5.10.1.5.2.12 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 dataset, TCustomDADataset .

See Also

- [SQLDelete](#)
- [SQLInsert](#)
- [SQLRefresh](#)

- [SQLUpdate](#)

5.10.1.5.2.13 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](#)

5.10.1.5.2.14 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

```
PgQuery.SQL := 'SELECT * FROM Dept ORDER BY &Order';  
PgQuery.MacroByName('Order').Value := 'DeptNo';  
PgQuery.Open;
```

See Also

- [TMacro](#)
- [MacroByName](#)
- [Params](#)

5.10.1.5.2.15 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

```
property MasterFields: string;
```

Remarks

Use the MasterFields property after setting the [MasterSource](#) property to specify the names of one or more fields that are used as foreign keys for this dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in these fields are used to select corresponding records in this table for display.

Use Field Link Designer to set the values at design time after setting the MasterSource property.

See Also

- [DetailFields](#)
- [MasterSource](#)
- [Master/Detail Relationships](#)

5.10.1.5.2.16 MasterSource Property

Used to specify the data source component which binds current dataset to the master one.

Class

[TCustomDADataset](#)

Syntax

property MasterSource: TDataSource;

Remarks

The MasterSource property specifies the data source component which binds current dataset to the master one.

TCustomDADataset uses MasterSource to extract foreign key fields values from the master dataset when building master/detail relationship between two datasets. MasterSource must point to another dataset; it cannot point to this dataset component.

When MasterSource is not **nil** dataset fills parameter values with corresponding field values from the current record of the master dataset.

Note: Do not set the DataSource property when building master/detail relationships. Although it points to the same object as the MasterSource property, it may lead to undesirable results.

See Also

- [MasterFields](#)
- [DetailFields](#)
- [Master/Detail Relationships](#)

5.10.1.5.2.17 Options Property

Used to specify the behaviour of TCustomDADataset object.

Class

[TCustomDADataset](#)

Syntax

property Options: [TDatasetOptions](#);

Remarks

Set the properties of Options to specify the behaviour of a TCustomDADataset object. Descriptions of all options are in the table below.

Option Name	Description
AutoPrepare	Used to execute automatic Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup

	fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DetailDelay	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin	Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail	Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable	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).
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount	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.
QuoteNames	Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields	Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to

	dataset after insert or update.
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.
TrimFixedChar	Specifies whether to discard all trailing spaces in the string fields of a dataset.
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

- [Master/Detail Relationships](#)
- [TMemDataSet.CachedUpdates](#)

5.10.1.5.2.18 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](#)

5.10.1.5.2.19 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](#)

5.10.1.5.2.20 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](#)

5.10.1.5.2.21 ReadOnly Property

Used to prevent users from updating, inserting, or deleting data in the dataset.

Class

[TCustomDADataset](#)

Syntax

```
property ReadOnly: boolean default False;
```

Remarks

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is False, meaning that users can potentially alter data stored in the dataset.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to True. When ReadOnly is True, the dataset's CanModify property is False.

5.10.1.5.2.22 RefreshOptions Property

Used to indicate when the editing record is refreshed.

Class

[TCustomDADataset](#)

Syntax

```
property RefreshOptions: TRefreshOptions default [];
```

Remarks

Use the RefreshOptions property to determine when the editing record is refreshed.

Refresh is performed by the [RefreshRecord](#) method.

It queries the current record and replaces one in the dataset. Refresh record is useful when the table has triggers or the table fields have default values. Use roBeforeEdit to get actual data before editing.

The default value is [].

See Also

- [RefreshRecord](#)

5.10.1.5.2.23 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

[TCustomDADataset](#)

Syntax

```
property RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

5.10.1.5.2.24 SQL Property

Used to provide a SQL statement that a query component executes when its Open method is called.

Class

[TCustomDADataset](#)

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to provide a SQL statement that a query component executes when its Open method is called. At the design time the SQL property can be edited by invoking the String List editor in Object Inspector.

When SQL is changed, TCustomDADataset calls Close and UnPrepare.

See Also

- [SQLInsert](#)
- [SQLUpdate](#)

- [SQLDelete](#)
- [SQLRefresh](#)

5.10.1.5.2.25 SQLDelete Property

Used to specify a SQL statement that will be used when applying a deletion to a record.

Class

[TCustomDADataset](#)

Syntax

```
property SQLDelete: TStrings;
```

Remarks

Use the SQLDelete property to specify the SQL statement that will be used when applying a deletion to a record. Statements can be parameterized queries.

To create a SQLDelete statement at design-time, use the query statements editor.

Example

```
DELETE FROM Orders
WHERE
    OrderID = :old_orderID
```

See Also

- [SQL](#)
- [SQLInsert](#)
- [SQLUpdate](#)
- [SQLRefresh](#)

5.10.1.5.2.26 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.

Use `ReturnParam` to return OUT parameters back to dataset.

To create a `SQLInsert` statement at design-time, use the query statements editor.

See Also

- [SQL](#)
- [SQLUpdate](#)
- [SQLDelete](#)
- [SQLRefresh](#)

5.10.1.5.2.27 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. 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](#)

5.10.1.5.2.28 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](#)
- [SQLRefresh](#)
- [TDADatasetOptions](#)
- M:Devart.Dac.TCustomDADataset.FetchingAll

5.10.1.5.2.29 SQLRefresh Property

Used to specify a SQL statement that will be used to refresh current record by calling the [RefreshRecord](#) procedure.

Class

[TCustomDADataset](#)

Syntax

```
property SQLRefresh: TStrings;
```

Remarks

Use the SQLRefresh property to specify a SQL statement that will be used to refresh current

record by calling the [RefreshRecord](#) procedure.

Different behavior is observed when the SQLRefresh property is assigned with a single WHERE clause that holds frequently altered search condition. In this case the WHERE clause from SQLRefresh is combined with the same clause of the SELECT statement in a SQL property and this final query is then sent to the database server.

To create a SQLRefresh statement at design-time, use the query statements editor.

Example

```
SELECT Shipname FROM Orders
WHERE
    OrderID = :OrderID
```

See Also

- [RefreshRecord](#)
- [SQL](#)
- [SQLInsert](#)
- [SQLUpdate](#)
- [SQLDelete](#)

5.10.1.5.2.30 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.

Use ReturnParam to return OUT parameters back to the dataset.

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](#)

5.10.1.5.2.31 UniDirectional Property

Used if an application does not need bidirectional access to records in the result set.

Class

[TCustomDADataset](#)

Syntax

```
property UniDirectional: boolean default False;
```

Remarks

Traditionally SQL cursors are unidirectional. They can travel only forward through a dataset. TCustomDADataset, however, permits bidirectional travelling by caching records. If an application does not need bidirectional access to the records in the result set, set UniDirectional to True. When UniDirectional is True, an application requires less memory and performance is improved. However, UniDirectional datasets cannot be modified. In FetchAll=False mode data is fetched on demand. When UniDirectional is set to True, data is fetched on demand as well, but obtained rows are not cached except for the current row. In case if the Unidirectional property is True, the FetchAll property will be automatically set to False. And if the FetchAll property is True, the Unidirectional property will be automatically set to False. The default value of UniDirectional is False, enabling forward and backward navigation.

Note: Pay attention to the specificity of using the FetchAll property=False

See Also

- [TPgQuery.FetchAll](#)

5.10.1.5.3 Methods

Methods of the **TCustomDADataset** class.

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

Public

Name	Description
AddWhere	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from TMemDataSet)	Applies a range to the dataset.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec	Breaks execution of the SQL statement on the server.
CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
CreateBlobStream	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
DeleteWhere	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from TMemDataSet)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from TMemDataSet)	Enables changing the starting value for an existing range.

Execute	Overloaded. Executes a SQL statement on the server.
Executing	Indicates whether SQL statement is still being executed.
Fetched	Used to learn whether TCustomDADataset has already fetched all rows.
Fetching	Used to learn whether TCustomDADataset is still fetching rows.
FetchingAll	Used to learn whether TCustomDADataset is fetching all rows to the end.
FindKey	Searches for a record which contains specified field values.
FindMacro	Description is not available at the moment.
FindNearest	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam	Determines if a parameter with the specified name exists in a dataset.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject	Returns a multireference shared object from field.
GetFieldPrecision	Retrieves the precision of a number field.
GetFieldScale	Retrieves the scale of a number field.
GetKeyFieldNames	Provides a list of available key field names.

GetOrderBy	Retrieves an ORDER BY clause from a SQL statement.
GotoCurrent	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from TMemDataSet)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from TMemDataSet)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock	Locks the current record.
MacroByName	Finds a Macro with the name passed in Name.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare	Allocates, opens, and parses cursor for a query.
RefreshRecord	Actualizes field values for the current record.
RestoreSQL	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from TMemDataSet)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL	Saves the SQL property value to BaseSQL.
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.

See Also

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

5.10.1.5.3.1 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](#)

5.10.1.5.3.2 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. It makes sense to call BreakExec only from another thread.

5.10.1.5.3.3 CreateBlobStream Method

Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.

Class

[TCustomDADataset](#)

Syntax

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode):  
TStream; override;
```

Parameters

Field

Holds the BLOB field for reading data from or writing data to from a stream.

Mode

Holds the stream mode, for which the stream will be used.

Return Value

The BLOB Stream.

Remarks

Call the CreateBlobStream method to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter. It must be a TBlobField component. You can specify whether the stream will be used for reading, writing, or updating the contents of the field with the Mode parameter.

5.10.1.5.3.4 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](#)

5.10.1.5.3.5 Execute Method

Executes a SQL statement on the server.

Class

[TCustomDADataset](#)

Overload List

Name	Description
------	-------------

Execute	Executes a SQL statement on the server.
Execute(Iter: integer; Offset: integer)	Used to perform Batch operations .

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

```
procedure Execute(Iter: integer; Offset: integer = 0); overload;  
virtual;
```

Parameters

Iter

Specifies the number of inserted rows.

Offset

Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The Execute method executes the specified batch SQL query. See the [Batch operations](#) article for samples.

See Also

- [Batch operations](#)

5.10.1.5.3.6 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.

5.10.1.5.3.7 Fetched Method

Used to learn whether TCustomDADataset has already fetched all rows.

Class

[TCustomDADataset](#)

Syntax

```
function Fetched: boolean; virtual;
```

Return Value

True, if all rows are fetched.

Remarks

Check Fetched to learn whether TCustomDADataset has already fetched all rows.

See Also

- [Fetching](#)

5.10.1.5.3.8 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](#)

5.10.1.5.3.9 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

- [Executing](#)

5.10.1.5.3.10 FindKey Method

Searches for a record which contains specified field values.

Class

[TCustomDADataset](#)

Syntax

```
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.

5.10.1.5.3.11 FindMacro Method

Class

[TCustomDADataset](#)

Syntax

```
function FindMacro(const value: string): TMacro;
```

Parameters

Value

See Also

- [TMacro](#)
- [Macros](#)
- [MacroByName](#)

5.10.1.5.3.12 FindNearest Method

Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.

Class

[TCustomDADataset](#)

Syntax

```
procedure FindNearest(const KeyValues: array of System.TVarRec);
```

Parameters

KeyValues

Holds the values of the record key fields to which the cursor should be moved.

Remarks

Call the FindNearest method to move the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter. If there are no records that match or exceed the specified criteria, the cursor will not move.

This function is provided for BDE compatibility only. It is recommended to use functions [TMemDataSet.Locate](#) and [TMemDataSet.LocateEx](#) for the record search.

See Also

- [TMemDataSet.Locate](#)
- [TMemDataSet.LocateEx](#)
- [FindKey](#)

5.10.1.5.3.13 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](#)

5.10.1.5.3.14 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.

5.10.1.5.3.15 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; overload;function  
GetFieldObject(const FieldName: string): TSharedObject; overload;
```

Parameters

FieldName

Holds the field name.

Return Value

multireference shared object.

Remarks

Call the GetFieldObject method to return a multireference shared object from field. If field does not hold one of the TSharedObject descendants, GetFieldObject raises an exception.

5.10.1.5.3.16 GetFieldPrecision Method

Retrieves the precision of a number field.

Class

[TCustomDADataset](#)

Syntax

```
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](#)

5.10.1.5.3.17 GetFieldScale Method

Retrieves the scale of a number field.

Class

[TCustomDADataset](#)

Syntax

```
function GetFieldScale(const FieldName: string): integer;
```

Parameters

FieldName

Holds the existing field name.

Return Value

the scale of the number field.

Remarks

Call the GetFieldScale method to retrieve the scale of a number field. FieldName is the name of an existing field.

See Also

- [GetFieldPrecision](#)

5.10.1.5.3.18 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

Key field name

Remarks

Call the GetKeyFieldNames method to get the names of available key fields. Populates a

string list with the names of key fields in tables.

See Also

- [TCustomDAConnection.GetTableNames](#)
- [TCustomDAConnection.GetStoredProcNames](#)

5.10.1.5.3.19 GetOrderBy Method

Retrieves an ORDER BY clause from a SQL statement.

Class

[TCustomDADataset](#)

Syntax

```
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](#)

5.10.1.5.3.20 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](#)

5.10.1.5.3.21 Lock Method

Locks the current record.

Class

[TCustomDADataset](#)

Syntax

```
procedure Lock; virtual;
```

Remarks

Call the Lock method to lock the current record by executing the statement that is defined in the SQLLock property.

The Lock method sets the savepoint with the name LOCK_ + <component_name>.

See Also

- [UnLock](#)

5.10.1.5.3.22 MacroByName Method

Finds a Macro with the name passed in Name.

Class

[TCustomDADataset](#)

Syntax

```
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds the name of the Macro to search for.

Return Value

the Macro, if a match was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Name. If a match was found, MacroByName returns the Macro. 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 FindMacro method.

To assign the value of macro use the [TMacro.Value](#) property.

Example

```
PgQuery.SQL:= 'SELECT * FROM Scott.Dept ORDER BY &Order';  
PgQuery.MacroByName('Order').Value:= 'DeptNo';  
PgQuery.Open;
```

See Also

- [TMacro](#)
- [Macros](#)
- [FindMacro](#)

5.10.1.5.3.23 ParamByName Method

Sets or uses parameter information for a specific parameter based on its name.

Class

[TCustomDADataset](#)

Syntax

```
function ParamByName(const Value: string): TDAParam;
```

Parameters

Value

Holds the name of the parameter for which to retrieve information.

Return Value

a TDAParam object.

Remarks

Call the ParamByName method to set or use parameter information for a specific parameter

based on its name. Name is the name of the parameter for which to retrieve information. ParamByName is used to set a parameter's value at runtime and returns a [TDAParam](#) object.

Example

The following statement retrieves the current value of a parameter called "Contact" into an edit box:

```
Edit1.Text := Query1.ParamsByName('Contact').AsString;
```

See Also

- [Params](#)
- [FindParam](#)

5.10.1.5.3.24 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

[TCustomDADataset](#)

Syntax

```
procedure Prepare; override;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- [TMemDataSet.Prepared](#)
- [TMemDataSet.UnPrepare](#)
- [Options](#)

5.10.1.5.3.25 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](#)

5.10.1.5.3.26 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](#)

5.10.1.5.3.27 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](#)

5.10.1.5.3.28 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

```
Query1.SetOrderBy('DeptNo;DName');
```

See Also

- [GetOrderBy](#)

5.10.1.5.3.29 SQLSaved Method

Determines if the [SQL](#) property value was saved to the [BaseSQL](#) property.

Class

[TCustomDADataset](#)

Syntax

```
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.

5.10.1.5.3.30 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](#)

5.10.1.5.4 Events

Events of the **TCustomDADataset** class.

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

Public

Name	Description
AfterExecute	Occurs after a component has executed a query to database.
AfterFetch	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from TMemDataSet)	Occurs when a single update component can not handle the updates.

See Also

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

5.10.1.5.4.1 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](#)

5.10.1.5.4.2 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](#)

5.10.1.5.4.3 AfterUpdateExecute Event

Occurs after executing insert, delete, update, lock and refresh operations.

Class

[TCustomDADataset](#)

Syntax

```
property AfterUpdateExecute: TUpdateExecuteEvent;
```

Remarks

Occurs after executing insert, delete, update, lock, and refresh operations. You can use AfterUpdateExecute to set the parameters of corresponding statements.

5.10.1.5.4.4 BeforeFetch Event

Occurs before dataset is going to fetch block of records from the server.

Class

[TCustomDADataset](#)

Syntax

```
property BeforeFetch: TBeforeFetchEvent;
```

Remarks

The BeforeFetch event occurs every time before dataset is going to fetch a block of records from the server. Set Cancel to True to abort current fetch operation.

See Also

- [AfterFetch](#)

5.10.1.5.4.5 BeforeUpdateExecute Event

Occurs before executing insert, delete, update, lock, and refresh operations.

Class

[TCustomDADataset](#)

Syntax

```
property BeforeUpdateExecute: TUpdateExecuteEvent;
```

Remarks

Occurs before executing insert, delete, update, lock, and refresh operations. You can use BeforeUpdateExecute to set the parameters of corresponding statements.

See Also

- [AfterUpdateExecute](#)

5.10.1.6 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

```
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.

5.10.1.6.1 Members

[TCustomDASQL](#) class overview.

Properties

Name	Description
ChangeCursor	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
Connection	Used to specify a connection object to use to connect to a data store.
Debug	Used to display executing statement, all its parameters' values, and the type of parameters.
FinalSQL	Used to return a SQL statement with expanded macros.
MacroCount	Used to get the number of macros associated with the Macros property.
Macros	Makes it possible to change SQL queries easily.
ParamCheck	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
ParamCount	Indicates the number of parameters in the Params property.
Params	Used to contain parameters for a SQL statement.
ParamValues	Used to get or set the values of individual field parameters that are identified by name.
Prepared	Used to indicate whether a query is prepared for execution.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

[SQL](#)

Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

Methods

Name	Description
<u>Execute</u>	Overloaded. Executes a SQL statement on the server.
<u>Executing</u>	Checks whether TCustomDASQL still executes a SQL statement.
<u>FindMacro</u>	Searches for a macro with the specified name.
<u>FindParam</u>	Finds a parameter with the specified name.
<u>MacroByName</u>	Finds a Macro with the name passed in Name.
<u>ParamByName</u>	Finds a parameter with the specified name.
<u>Prepare</u>	Allocates, opens, and parses cursor for a query.
<u>UnPrepare</u>	Frees the resources allocated for a previously prepared query on the server and client sides.
<u>WaitExecuting</u>	Waits until TCustomDASQL executes a SQL statement.

Events

Name	Description
<u>AfterExecute</u>	Occurs after a SQL statement has been executed.

5.10.1.6.2 Properties

Properties of the **TCustomDASQL** class.

For a complete list of the **TCustomDASQL** class members, see the [TCustomDASQL](#)

[Members](#) topic.

Public

Name	Description
ChangeCursor	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
Connection	Used to specify a connection object to use to connect to a data store.
Debug	Used to display executing statement, all its parameters' values, and the type of parameters.
FinalSQL	Used to return a SQL statement with expanded macros.
MacroCount	Used to get the number of macros associated with the Macros property.
Macros	Makes it possible to change SQL queries easily.
ParamCheck	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
ParamCount	Indicates the number of parameters in the Params property.
Params	Used to contain parameters for a SQL statement.
ParamValues	Used to get or set the values of individual field parameters that are identified by name.
Prepared	Used to indicate whether a query is prepared for execution.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SQL	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

See Also

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

5.10.1.6.2.1 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.

5.10.1.6.2.2 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.

5.10.1.6.2.3 Debug Property

Used to display executing statement, all its parameters' values, and the type of parameters.

Class

[TCustomDASQL](#)

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display executing statement and all its parameters' values. Also displays the type of parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

See Also

- [TCustomDADataset.Debug](#)

5.10.1.6.2.4 FinalSQL Property

Used to return a SQL statement with expanded macros.

Class

[TCustomDASQL](#)

Syntax

```
property FinalSQL: string;
```

Remarks

Read the FinalSQL property to return a SQL statement with expanded macros. This is the exact statement that will be passed on to the database server.

5.10.1.6.2.5 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](#)

5.10.1.6.2.6 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](#)
- [MacroByName](#)
- [Params](#)

5.10.1.6.2.7 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](#)

5.10.1.6.2.8 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.

5.10.1.6.2.9 Params Property

Used to contain parameters for a SQL statement.

Class

[TCustomDASQL](#)

Syntax

property Params: [TDAParams](#) **stored** False;

Remarks

Access the Params property at runtime to view and set parameter names, values, and data types dynamically (at design-time use the Parameters editor to set parameter properties). Params is a zero-based array of parameter records. Index specifies the array element to access. An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

Example

Setting parameters at runtime:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
with PgSQL do
begin
    SQL.Clear;
    SQL.Add('INSERT INTO Temp_Table(Id, Name)');
    SQL.Add('VALUES (:id, :Name)');
    ParamByName('Id').AsInteger := 55;
    Params[1].AsString := ' Green';
    Execute;
end;
end;
```

See Also

- [TDAParam](#)
- [FindParam](#)
- [Macros](#)

5.10.1.6.2.10 ParamValues Property(Indexer)

Used to get or set the values of individual field parameters that are identified by name.

Class

[TCustomDASQL](#)

Syntax

property ParamValues[**const** ParamName: **string**]: Variant; **default**;

Parameters

ParamName

Holds parameter names separated by semicolon.

Remarks

Use the ParamValues property to get or set the values of individual field parameters that are identified by name.

Setting ParamValues sets the Value property for each parameter listed in the ParamName string. Specify the values as Variants.

Getting ParamValues retrieves an array of variants, each of which represents the value of one of the named parameters.

Note: The Params array is generated implicitly if ParamCheck property is set to True. If ParamName includes a name that does not match any of the parameters in Items, an exception is raised.

5.10.1.6.2.11 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](#)

5.10.1.6.2.12 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.

5.10.1.6.2.13 SQL Property

Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

Class

[TCustomDASQL](#)

Syntax

property SQL: TStrings;

Remarks

Use the SQL property to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called. At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

See Also

- [FinalSQL](#)
- [TCustomDASQL.Execute](#)

5.10.1.6.3 Methods

Methods of the **TCustomDASQL** class.

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

Public

Name	Description
Execute	Overloaded. Executes a SQL statement on the server.
Executing	Checks whether TCustomDASQL still executes a SQL statement.

FindMacro	Searches for a macro with the specified name.
FindParam	Finds a parameter with the specified name.
MacroByName	Finds a Macro with the name passed in Name.
ParamByName	Finds a parameter with the specified name.
Prepare	Allocates, opens, and parses cursor for a query.
UnPrepare	Frees the resources allocated for a previously prepared query on the server and client sides.
WaitExecuting	Waits until TCustomDASQL executes a SQL statement.

See Also

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

5.10.1.6.3.1 Execute Method

Executes a SQL statement on the server.

Class

[TCustomDASQL](#)

Overload List

Name	Description
Execute	Executes a SQL statement on the server.
Execute(Iters: integer; Offset: integer)	Used to perform Batch operations .

Executes a SQL statement on the server.

Class

[TCustomDASQL](#)

Syntax

procedure Execute; **overload**; **virtual**;

Remarks

Call the Execute method to execute a SQL statement on the server. If the SQL statement has OUT parameters, use the [TCustomDASQL.ParamByName](#) method or the [TCustomDASQL.Params](#) property to get their values. Iters argument specifies the number of times this statement is executed for the DML array operations.

Used to perform [Batch operations](#) .

Class

[TCustomDASQL](#)

Syntax

procedure Execute(Iters: integer; Offset: integer = 0); **overload**; **virtual**;

Parameters

Iters

Specifies the number of inserted rows.

Offset

Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The Execute method executes the specified batch SQL query. See the [Batch operations](#) article for samples.

See Also

- [Batch operations](#)

5.10.1.6.3.2 Executing Method

Checks whether TCustomDASQL still executes a SQL statement.

Class

[TCustomDASQL](#)

Syntax

function Executing: boolean;

Return Value

True, if a SQL statement is still being executed by TCustomDASQL.

Remarks

Check Executing to find out whether TCustomDASQL still executes a SQL statement.

5.10.1.6.3.3 FindMacro Method

Searches for 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

the TMacro object, if a macro with the specified name has been found. If it has not, returns nil.

Remarks

Call the FindMacro method to find a macro with the specified name in a dataset.

See Also

- [TMacro](#)
- [Macros](#)
- [MacroByName](#)

5.10.1.6.3.4 FindParam Method

Finds a parameter with the specified name.

Class

[TCustomDASQL](#)

Syntax

```
function FindParam(const value: string): TDAParam;
```

Parameters

Value

Holds the parameter name to search for.

Return Value

a TDAParm 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](#)

5.10.1.6.3.5 MacroByName Method

Finds a Macro with the name passed in Name.

Class

[TCustomDASQL](#)

Syntax

```
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds the name of the Macro to search for.

Return Value

the Macro, if a match was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Name. If a match was found, MacroByName returns the Macro. 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 FindMacro method.

To assign the value of macro use the [TMacro.Value](#) property.

See Also

- [TMacro](#)

- [Macros](#)
- [FindMacro](#)

5.10.1.6.3.6 ParamByName Method

Finds a parameter with the specified name.

Class

[TCustomDASQL](#)

Syntax

```
function ParamByName(const Value: string): TDAParam;
```

Parameters

Value

Holds the name of the parameter to search for.

Return Value

a TDAParam object, if a match was found. Otherwise, an exception is raised.

Remarks

Use the ParamByName method to find a parameter with the specified name. If no parameter with the specified name found, an exception is raised.

Example

```
PgSQL.Execute;  
Edit1.Text := PgSQL.ParamsByName('contact').AsString;
```

See Also

- [FindParam](#)

5.10.1.6.3.7 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

[TCustomDASQL](#)

Syntax

```
procedure Prepare; virtual;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- [Prepared](#)
- [UnPrepare](#)

5.10.1.6.3.8 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](#)

5.10.1.6.3.9 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](#)

5.10.1.6.4 Events

Events of the **TCustomDASQL** class.

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

Public

Name	Description
AfterExecute	Occurs after a SQL statement has been executed.

See Also

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

5.10.1.6.4.1 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](#)

5.10.1.7 TCustomDAUpdateSQL Class

A base class for components that provide DML statements for more flexible control over data modifications.

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

Unit

[DBAccess](#)

Syntax

```
TCustomDAUpdateSQL = class(TComponent);
```

Remarks

TCustomDAUpdateSQL is a base class for components that provide DML statements for more flexible control over data modifications. Besides providing BDE compatibility, this component allows to associate a separate component for each update command.

See Also

- [TCustomPgDataSet.UpdateObject](#)

5.10.1.7.1 Members

[TCustomDAUpdateSQL](#) class overview.

Properties

Name	Description
DataSet	Used to hold a reference to the TCustomDADataset object that is being updated.
DeleteObject	Provides ability to perform advanced adjustment of the delete operations.
DeleteSQL	Used when deleting a record.
InsertObject	Provides ability to perform advanced adjustment of insert operations.
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

Name	Description
Apply	Sets parameters for a SQL statement and executes it to update a record.
ExecSQL	Executes a SQL statement.

5.10.1.7.2 Properties

Properties of the **TCustomDAUpdateSQL** class.

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

Public

Name	Description
DataSet	Used to hold a reference to the TCustomDADataset object that is being updated.

[SQL](#)

Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Published

Name	Description
<u>DeleteObject</u>	Provides ability to perform advanced adjustment of the delete operations.
<u>DeleteSQL</u>	Used when deleting a record.
<u>InsertObject</u>	Provides ability to perform advanced adjustment of insert operations.
<u>InsertSQL</u>	Used when inserting a record.
<u>LockObject</u>	Provides ability to perform advanced adjustment of lock operations.
<u>LockSQL</u>	Used to lock the current record.
<u>ModifyObject</u>	Provides ability to perform advanced adjustment of modify operations.
<u>ModifySQL</u>	Used when updating a record.
<u>RefreshObject</u>	Provides ability to perform advanced adjustment of refresh operations.
<u>RefreshSQL</u>	Used to specify an SQL statement that will be used for refreshing the current record by <u>TCustomDADataset.RefreshRecord</u> procedure.

See Also

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

5.10.1.7.2.1 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.

5.10.1.7.2.2 DeleteObject Property

Provides ability to perform advanced adjustment of the delete operations.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property DeleteObject: TComponent;
```

Remarks

Assign SQL component or a TCustomPgDataSet descendant to this property to perform advanced adjustment of the delete operations. In some cases this can give some additional performance. Use the same principle to set the SQL property of an object as for setting the [DeleteSQL](#) property.

See Also

- [DeleteSQL](#)

5.10.1.7.2.3 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.

5.10.1.7.2.4 InsertObject Property

Provides ability to perform advanced adjustment of insert operations.

Class

[TCustomDAUpdateSQL](#)

Syntax

property InsertObject: TComponent;

Remarks

Assign SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of insert operations. In some cases this can give some additional performance. Set the SQL property of the object in the same way as used for the [InsertSQL](#) property.

See Also

- [InsertSQL](#)

5.10.1.7.2.5 InsertSQL Property

Used when inserting a record.

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.

5.10.1.7.2.6 LockObject Property

Provides ability to perform advanced adjustment of lock operations.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property LockObject: TComponent;
```

Remarks

Assign a SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of lock operations. In some cases that can give some additional performance. Set the SQL property of an object in the same way as used for the [LockSQL](#) property.

See Also

- [LockSQL](#)

5.10.1.7.2.7 LockSQL Property

Used to lock the current record.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property LockSQL: TStrings;
```

Remarks

Use the LockSQL property to lock the current record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

5.10.1.7.2.8 ModifyObject Property

Provides ability to perform advanced adjustment of modify operations.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property ModifyObject: TComponent;
```

Remarks

Assign a SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of modify operations. In some cases this can give some additional performance. Set the SQL property of the object in the same way as used for the [ModifySQL](#) property.

See Also

- [ModifySQL](#)

5.10.1.7.2.9 ModifySQL Property

Used when updating a record.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
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.

5.10.1.7.2.10 RefreshObject Property

Provides ability to perform advanced adjustment of refresh operations.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property RefreshObject: TComponent;
```

Remarks

Assign a SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of refresh operations. In some cases that can give some additional

performance. Set the SQL property of the object in the same way as used for the [RefreshSQL](#) property.

See Also

- [RefreshSQL](#)

5.10.1.7.2.11 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](#)

5.10.1.7.2.12 SQL Property(Indexer)

Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property SQL[UpdateKind: TUpdateKind]: TStrings;
```

Parameters

UpdateKind

Specifies which of update SQL statements to return.

Remarks

Returns a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties, depending on the value of the UpdateKind index.

5.10.1.7.3 Methods

Methods of the **TCustomDAUpdateSQL** class.

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

Public

Name	Description
Apply	Sets parameters for a SQL statement and executes it to update a record.
ExecSQL	Executes a SQL statement.

See Also

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

5.10.1.7.3.1 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](#)

5.10.1.7.3.2 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](#)

5.10.1.8 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](#)

5.10.1.8.1 Members

[TDACondition](#) class overview.

Properties

Name	Description
Enabled	Indicates whether the condition is enabled or not
Name	The name of the condition
Value	The value of the condition

Methods

Name	Description
Disable	Disables the condition
Enable	Enables the condition

5.10.1.8.2 Properties

Properties of the **TDACondition** class.

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

Published

Name	Description
Enabled	Indicates whether the condition is enabled or not
Name	The name of the condition
Value	The value of the condition

See Also

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

5.10.1.8.2.1 Enabled Property

Indicates whether the condition is enabled or not

Class

[TDACondition](#)

Syntax

```
property Enabled: Boolean default True;
```

5.10.1.8.2.2 Name Property

The name of the condition

Class

[TDACondition](#)

Syntax

```
property Name: string;
```

5.10.1.8.2.3 Value Property

The value of the condition

Class

[TDACondition](#)

Syntax

```
property value: string;
```

5.10.1.8.3 Methods

Methods of the **TDACondition** class.

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

Public

Name	Description
Disable	Disables the condition
Enable	Enables the condition

See Also

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

5.10.1.8.3.1 Disable Method

Disables the condition

Class

[TDACondition](#)

Syntax

```
procedure Disable;
```

5.10.1.8.3.2 Enable Method

Enables the condition

Class

[TDACondition](#)

Syntax

```
procedure Enable;
```

5.10.1.9 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)
```

5.10.1.9.1 Members

[TDAConditions](#) class overview.

Properties

Name	Description
Condition	Used to iterate through all the conditions.
Enabled	Indicates whether the condition is enabled
Items	Used to iterate through all conditions.
Text	The property returns condition names and values as CONDITION_NAME=CONDITION
WhereSQL	Returns the SQL WHERE condition added in the Conditions property.

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

5.10.1.9.2 Properties

Properties of the **TDAConditions** class.

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

Public

Name	Description
Condition	Used to iterate through all the conditions.
Enabled	Indicates whether the condition is enabled

Items	Used to iterate through all conditions.
Text	The property returns condition names and values as CONDITION NAME=CONDITION
WhereSQL	Returns the SQL WHERE condition added in the Conditions property.

See Also

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

5.10.1.9.2.1 Condition Property(Indexer)

Used to iterate through all the conditions.

Class

[TDAConditions](#)

Syntax

```
property Condition[Index: Integer]: TDACondition;
```

Parameters

Index

5.10.1.9.2.2 Enabled Property

Indicates whether the condition is enabled

Class

[TDAConditions](#)

Syntax

```
property Enabled: Boolean;
```

5.10.1.9.2.3 Items Property(Indexer)

Used to iterate through all conditions.

Class

[TDAConditions](#)

Syntax

```
property Items[Index: Integer]: TDACondition; default;
```

Parameters

Index

Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all conditions. Index identifies the index in the range 0..Count - 1. Items can reference a particular condition by its index, but the [Condition](#) property is preferred in order to avoid depending on the order of the conditions.

5.10.1.9.2.4 Text Property

The property returns condition names and values as CONDITION_NAME=CONDITION

Class

[TDAConditions](#)

Syntax

```
property Text: string;
```

5.10.1.9.2.5 WhereSQL Property

Returns the SQL WHERE condition added in the Conditions property.

Class

[TDAConditions](#)

Syntax

```
property whereSQL: string;
```

5.10.1.9.3 Methods

Methods of the **TDAConditions** class.

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

Public

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

See Also

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

5.10.1.9.3.1 Add Method

Adds a condition to the WHERE clause of the query.

Class

[TDAConditions](#)

Overload List

Name	Description
Add(const Value: string; Enabled: Boolean)	Adds a condition to the WHERE clause of the query.
Add(const Name: string; const Value:	Adds a condition to the WHERE clause of

string ; Enabled: Boolean)	the query.
--	------------

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:

```
SELECT * FROM EMP  
WHERE (JOB="MANAGER")  
and  
(SAL<2500)
```

5.10.1.9.3.2 Delete Method

Deletes the condition

Class

[TDAConditions](#)

Syntax

```
procedure Delete(Index: integer);
```

Parameters

Index

Index of the condition

5.10.1.9.3.3 Disable Method

Disables the condition

Class

[TDAConditions](#)

Syntax

```
procedure Disable;
```

5.10.1.9.3.4 Enable Method

Enables the condition

Class

[TDAConditions](#)

Syntax

```
procedure Enable;
```

5.10.1.9.3.5 Find Method

Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.

Class

[TDAConditions](#)

Syntax

```
function Find(const Name: string): TDACondition;
```

Parameters

Name

5.10.1.9.3.6 Get Method

Retrieving a TDACondition object by its name. If found, the TDACondition object is returned, otherwise - an exception is raised.

Class

[TDAConditions](#)

Syntax

```
function Get(const Name: string): TDACondition;
```

Parameters

Name

5.10.1.9.3.7 IndexOf Method

Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.

Class

[TDAConditions](#)

Syntax

```
function IndexOf(const Name: string): Integer;
```

Parameters

Name

5.10.1.9.3.8 Remove Method

Removes the condition

Class

[TDAConditions](#)

Syntax

```
procedure Remove(const Name: string);
```

Parameters

Name

Specifies the name of the removed condition

5.10.1.10 TDAConnectionOptions Class

This class allows setting up the behaviour of the TDAConnection class.

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

Unit

[DBAccess](#)

Syntax

```
TDAConnectionOptions = class(TPersistent);
```

5.10.1.10.1 Members

[TDAConnectionOptions](#) class overview.

Properties

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.
DefaultSortType	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.
DisconnectedMode	Used to open a connection only when needed for performing a server call and closes after performing the

	operation.
KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover	If True, the TCustomDACConnection.OnConnecti onLost event occurs and a failover operation can be performed after connection breaks.

5.10.1.10.2 Properties

Properties of the **TDACConnectionOptions** class.

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

Public

Name	Description
DefaultSortType	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.
DisconnectedMode	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover	If True, the TCustomDACConnection.OnConnecti onLost event occurs and a failover operation can be performed after connection breaks.

Published

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.

See Also

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

5.10.1.10.2.1 AllowImplicitConnect Property

Specifies whether to allow or not implicit connection opening.

Class

[TDAConnectionOptions](#)

Syntax

```
property AllowImplicitConnect: boolean default True;
```

Remarks

Use the AllowImplicitConnect property to specify whether allow or not implicit connection opening.

If a closed connection has AllowImplicitConnect set to True and a dataset that uses the connection is opened, the connection is opened implicitly to allow opening the dataset.

If a closed connection has AllowImplicitConnect set to False and a dataset that uses the connection is opened, an exception is raised.

The default value is True.

5.10.1.10.2.2 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

[TDAConnectionOptions](#)

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.

5.10.1.10.2.3 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.

5.10.1.10.2.4 KeepDesignConnected Property

Used to prevent an application from establishing a connection at the time of startup.

Class

[TDACconnectionOptions](#)

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.

5.10.1.10.2.5 LocalFailover Property

If True, the [TCustomDAConnection.OnConnectionLost](#) event occurs and a failover operation can be performed after connection breaks.

Class

[TDAConnectionOptions](#)

Syntax

```
property LocalFailover: boolean default False;
```

Remarks

If True, the [TCustomDASession.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.

5.10.1.11 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);
```

5.10.1.11.1 Members

[TDADatasetOptions](#) class overview.

Properties

Name	Description
AutoPrepare	Used to execute automatic TCustomDASession.Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the DefaultExpression property.

<u>DetailDelay</u>	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
<u>FieldsOrigin</u>	Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.
<u>FlatBuffers</u>	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
<u>InsertAllSetFields</u>	Used to include all set dataset fields in the generated INSERT statement
<u>LocalMasterDetail</u>	Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
<u>LongStrings</u>	Used to represent string fields with the length that is greater than 255 as TStringField.
<u>MasterFieldsNullable</u>	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).
<u>NumberRange</u>	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
<u>QueryRecCount</u>	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.
<u>QuoteNames</u>	Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.
<u>RemoveOnRefresh</u>	Used for a dataset to locally remove a record that can not be found on the server.

RequiredFields	Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to dataset after insert or update.
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.
TrimFixedChar	Specifies whether to discard all trailing spaces in the string fields of a dataset.
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.

5.10.1.11.2 Properties

Properties of the **TDADatasetOptions** class.

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

Public

Name	Description
AutoPrepare	Used to execute automatic TCustomDADataset.Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.

<u>DefaultValues</u>	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
<u>DetailDelay</u>	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
<u>FieldsOrigin</u>	Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.
<u>FlatBuffers</u>	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
<u>InsertAllSetFields</u>	Used to include all set dataset fields in the generated INSERT statement
<u>LocalMasterDetail</u>	Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
<u>LongStrings</u>	Used to represent string fields with the length that is greater than 255 as TStringField.
<u>MasterFieldsNullable</u>	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).
<u>NumberRange</u>	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
<u>QueryRecCount</u>	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.
<u>QuoteNames</u>	Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.

RemoveOnRefresh	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields	Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to dataset after insert or update.
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.
TrimFixedChar	Specifies whether to discard all trailing spaces in the string fields of a dataset.
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](#)

5.10.1.11.2.1 AutoPrepare Property

Used to execute automatic [TCustomDADataset.Prepare](#) on the query execution.

Class

[TDADatasetOptions](#)

Syntax

```
property AutoPrepare: boolean default False;
```

Remarks

Use the AutoPrepare property to execute automatic [TCustomDADataset.Prepare](#) on the query execution. Makes sense for cases when a query will be executed several times, for example, in Master/Detail relationships.

5.10.1.11.2.2 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.

5.10.1.11.2.3 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.

5.10.1.11.2.4 DefaultValues Property

Used to request default values/expressions from the server and assign them to the DefaultExpression property.

Class

[TDADatasetOptions](#)

Syntax

```
property DefaultValues: boolean default False;
```

Remarks

If True, the default values/expressions are requested from the server and assigned to the DefaultExpression property of TField objects replacing already existent values.

5.10.1.11.2.5 DetailDelay Property

Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.

Class

[TDADatasetOptions](#)

Syntax

```
property DetailDelay: integer default 0;
```

Remarks

Use the DetailDelay property to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset. If DetailDelay is 0 (the default value) then refreshing of detail dataset occurs immediately. The DetailDelay option should be used for detail dataset.

5.10.1.11.2.6 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 default False;
```

Remarks

If True, TCustomDADataset fills the Origin property of the TField objects by appropriate value when opening a dataset.

5.10.1.11.2.7 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.

5.10.1.11.2.8 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.

5.10.1.11.2.9 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.

5.10.1.11.2.10 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.

5.10.1.11.2.11 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;
```

5.10.1.11.2.12 NumberRange Property

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Class

[TDADatasetOptions](#)

Syntax

```
property NumberRange: boolean default False;
```

Remarks

Use the NumberRange property to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

5.10.1.11.2.13 QueryRecCount Property

Used for TCustomDADataset to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.

Class

[TDADatasetOptions](#)

Syntax

```
property QueryRecCount: boolean default False;
```

Remarks

If True, and the FetchAll property is False, TCustomDADataset performs additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records. Does not have any effect if the FetchAll property is True.

5.10.1.11.2.14 QuoteNames Property

Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.

Class

[TDADatasetOptions](#)

Syntax

```
property QuoteNames: boolean default False;
```

Remarks

If True, TCustomDADataset quotes all database object names in autogenerated SQL statements such as update SQL.

5.10.1.11.2.15 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.

5.10.1.11.2.16 RequiredFields Property

Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.

Class

[TDADatasetOptions](#)

Syntax

```
property RequiredFields: boolean default True;
```

Remarks

If True, TCustomDADataset sets the Required property of the TField objects for the NOT NULL fields. It is useful when table has a trigger which updates the NOT NULL fields.

5.10.1.11.2.17 ReturnParams Property

Used to return the new value of fields to dataset after insert or update.

Class

[TDADatasetOptions](#)

Syntax

```
property ReturnParams: boolean default False;
```

Remarks

Use the ReturnParams property to return the new value of fields to dataset after insert or update. The actual value of field after insert or update may be different from the value stored in the local memory if the table has a trigger. When ReturnParams is True, OUT parameters of the SQLInsert and SQLUpdate statements is assigned to the corresponding fields.

5.10.1.11.2.18 SetFieldsReadOnly Property

Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.

Class

[TDADatasetOptions](#)

Syntax

```
property SetFieldsReadOnly: boolean default True;
```

Remarks

If True, dataset sets the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated. Set this option for datasets that use automatic generation of the update SQL statements only.

5.10.1.11.2.19 StrictUpdate Property

Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.

Class

[TDADatasetOptions](#)

Syntax

```
property StrictUpdate: boolean default True;
```

Remarks

If True, TCustomDADataset raises an exception when the number of updated or deleted records is not equal 1. Setting this option also causes the exception if the RefreshRecord procedure returns more than one record. The exception does not occur when you execute SQL query, that doesn't return resultset.

Note: There can be problems if this option is set to True and triggers for UPDATE, DELETE, REFRESH commands that are defined for the table. So it is recommended to disable (set to False) this option with triggers.

TrimFixedChar specifies whether to discard all trailing spaces in the string fields of a dataset.

5.10.1.11.2.20 TrimFixedChar Property

Specifies whether to discard all trailing spaces in the string fields of a dataset.

Class

[TDADatasetOptions](#)

Syntax

```
property TrimFixedChar: boolean default True;
```

Remarks

Specifies whether to discard all trailing spaces in the string fields of a dataset.

5.10.1.11.2.21 UpdateAllFields Property

Used to include all dataset fields in the generated UPDATE and INSERT statements.

Class

[TDADatasetOptions](#)

Syntax

```
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.

5.10.1.11.2.22 UpdateBatchSize Property

Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Class

[TDADatasetOptions](#)

Syntax

```
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.

5.10.1.12 TDAEncryption Class

Used to specify the options of the data encryption in a dataset.
For a list of all members of this type, see [TDAEncryption](#) members.

Unit

[DBAccess](#)

Syntax

```
TDAEncryption = class(TPersistent);
```

Remarks

Set the properties of Encryption to specify the options of the data encryption in a dataset.

5.10.1.12.1 Members

[TDAEncryption](#) class overview.

Properties

Name	Description
Encryptor	Used to specify the encryptor class that will perform the data encryption.

[Fields](#)

Used to set field names for which encryption will be performed.

5.10.1.12.2 Properties

Properties of the **TDAEncryption** class.

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

Public

Name	Description
Encryptor	Used to specify the encryptor class that will perform the data encryption.

Published

Name	Description
Fields	Used to set field names for which encryption will be performed.

See Also

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

5.10.1.12.2.1 Encryptor Property

Used to specify the encryptor class that will perform the data encryption.

Class

[TDAEncryption](#)

Syntax

```
property Encryptor: TCREncryptor;
```

Remarks

Use the Encryptor property to specify the encryptor class that will perform the data encryption.

5.10.1.12.2.2 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.

5.10.1.13 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

5.10.1.13.1 Members

[TDAMapRule](#) class overview.

Properties

Name	Description
------	-------------

DBLengthMax	Maximum DB field length, until which the rule is applied.
DBLengthMin	Minimum DB field length, starting from which the rule is applied.
DBScaleMax	Maximum DB field scale, until which the rule is applied to the specified DB field.
DBScaleMin	Minimum DB field Scale, starting from which the rule is applied to the specified DB field.
DBType	DB field type, that the rule is applied to.
FieldLength	The resultant field length in Delphi.
FieldName	DataSet field name, for which the rule is applied.
FieldScale	The resultant field Scale in Delphi.
FieldType	Delphi field type, that the specified DB type or DataSet field will be mapped to.
IgnoreErrors	Ignoring errors when converting data from DB to Delphi type.

5.10.1.13.2 Properties

Properties of the **TDAMapRule** class.

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

Published

Name	Description
DBLengthMax	Maximum DB field length, until which the rule is applied.
DBLengthMin	Minimum DB field length, starting from which the rule is applied.
DBScaleMax	Maximum DB field scale, until which the rule is applied to the specified

	DB field.
DBScaleMin	Minimum DB field Scale, starting from which the rule is applied to the specified DB field.
DBType	DB field type, that the rule is applied to.
FieldLength	The resultant field length in Delphi.
FieldName	DataSet field name, for which the rule is applied.
FieldScale	The resultant field Scale in Delphi.
FieldType	Delphi field type, that the specified DB type or DataSet field will be mapped to.
IgnoreErrors	Ignoring errors when converting data from DB to Delphi type.

See Also

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

5.10.1.13.2.1 DBLengthMax Property

Maximum DB field length, until which the rule is applied.

Class

[TDAMapRule](#)

Syntax

```
property DBLengthMax: Integer default r1Any;
```

Remarks

Setting maximum DB field length, until which the rule is applied to the specified DB field.

5.10.1.13.2.2 DBLengthMin Property

Minimum DB field length, starting from which the rule is applied.

Class

[TDAMapRule](#)

Syntax

```
property DBLengthMin: Integer default r1Any;
```

Remarks

Setting minimum DB field length, starting from which the rule is applied to the specified DB field.

5.10.1.13.2.3 DBScaleMax Property

Maximum DB field scale, until which the rule is applied to the specified DB field.

Class

[TDAMapRule](#)

Syntax

```
property DBScaleMax: Integer default r1Any;
```

Remarks

Setting maximum DB field scale, until which the rule is applied to the specified DB field.

5.10.1.13.2.4 DBScaleMin Property

Minimum DB field Scale, starting from which the rule is applied to the specified DB field.

Class

[TDAMapRule](#)

Syntax

```
property DBScaleMin: Integer default r1Any;
```

Remarks

Setting minimum DB field Scale, starting from which the rule is applied to the specified DB field.

5.10.1.13.2.5 DBType Property

DB field type, that the rule is applied to.

Class

[TDAMapRule](#)

Syntax

```
property DBType: word default dtUnknown;
```

Remarks

Setting DB field type, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields of the specified type in all DataSets related to this Connection.

5.10.1.13.2.6 FieldLength Property

The resultant field length in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldLength: Integer default r1Any;
```

Remarks

Setting the Delphi field length after conversion.

5.10.1.13.2.7 FieldName Property

DataSet field name, for which the rule is applied.

Class

[TDAMapRule](#)

Syntax

```
property FieldName: string;
```

Remarks

Specifies the DataSet field name, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields with such name in DataSets related to this Connection.

5.10.1.13.2.8 FieldScale Property

The resultant field Scale in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldScale: Integer default r1Any;
```

Remarks

Setting the Delphi field Scale after conversion.

5.10.1.13.2.9 FieldType Property

Delphi field type, that the specified DB type or DataSet field will be mapped to.

Class

[TDAMapRule](#)

Syntax

```
property FieldType: TFieldType stored IsFieldTypeStored default  
ftUnknown;
```

Remarks

Setting Delphi field type, that the specified DB type or DataSet field will be mapped to.

5.10.1.13.2.10 IgnoreErrors Property

Ignoring errors when converting data from DB to Delphi type.

Class

[TDAMapRule](#)

Syntax

```
property IgnoreErrors: Boolean default False;
```

Remarks

Allows to ignore errors while data conversion in case if data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

5.10.1.14 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

5.10.1.14.1 Members

[TDAMapRules](#) class overview.

Properties

Name	Description
IgnoreInvalidRules	Used to avoid raising exception on mapping rules that can't be applied.

5.10.1.14.2 Properties

Properties of the **TDAMapRules** class.

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

Published

Name	Description
IgnoreInvalidRules	Used to avoid raising exception on mapping rules that can't be applied.

See Also

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

5.10.1.14.2.1 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](#)

5.10.1.15 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:

```
MetaData.Connection := Connection;
MetaData.MetaDataKind := 'Columns';
MetaData.Restrictions.Values['TABLE_NAME'] := 'Emp';
MetaData.Open;
```

Inheritance Hierarchy

[TMemDataSet](#)

TDAMetaData

See Also

- [TDAMetaData.MetaDataKind](#)
- [TDAMetaData.Restrictions](#)
- [TDAMetaData.GetMetaDataKinds](#)
- [TDAMetaData.GetRestrictions](#)

5.10.1.15.1 Members

[TDAMetaData](#) class overview.

Properties

Name	Description
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
Connection	Used to specify a connection object to use to connect to a data store.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
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

<u>TMemDataSet</u>)	opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MetaDataKind</u>	Used to specify which kind of metainformation to show.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.
<u>Restrictions</u>	Used to provide one or more conditions restricting the list of objects to be described.
<u>UpdateRecordTypes</u> (inherited from <u>TMemDataSet</u>)	Used to indicate the update status for the current record when cached updates are enabled.
<u>UpdatesPending</u> (inherited from <u>TMemDataSet</u>)	Used to check the status of the cached updates buffer.

Methods

Name	Description
<u>ApplyRange</u> (inherited from <u>TMemDataSet</u>)	Applies a range to the dataset.
<u>ApplyUpdates</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Writes dataset's pending cached updates to a database.
<u>CancelRange</u> (inherited from <u>TMemDataSet</u>)	Removes any ranges currently in effect for a dataset.
<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.

DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
EditRangeEnd (inherited from TMemDataSet)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from TMemDataSet)	Enables changing the starting value for an existing range.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds	Used to get values acceptable in the MetaDataKind property.
GetRestrictions	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from TMemDataSet)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from TMemDataSet)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from TMemDataSet)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from TMemDataSet)	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

Name	Description
OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from TMemDataSet)	Occurs when a single update component can not handle the updates.

5.10.1.15.2 Properties

Properties of the **TDAMetaData** class.

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

Public

Name	Description
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
Connection	Used to specify a connection object to use to connect to a data store.

IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
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.
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](#)

5.10.1.15.2.1 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

[TDAMetaData](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object to use to connect to a data store. Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, set the Connection property to reference an instantiated TCustomDAConnection object.

5.10.1.15.2.2 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:

MetadataKind	Description
Columns	show metainformation about columns of existing tables
Constraints	show metainformation about the constraints defined in the database
IndexColumns	show metainformation about indexed columns
Indexes	show metainformation about indexes in a database
MetadataKinds	show the acceptable values of this property. You will get the same result if the MetadataKind property is an empty string
ProcedureParameters	show metainformation about parameters of existing procedures
Procedures	show metainformation about existing procedures
Restrictions	generates a dataset that describes which restrictions are applicable to each MetadataKind
Tables	show metainformation about existing tables

Databases	show metainformation about existing databases
-----------	---

If you provide a value that equals neither of the values described in the table, an error will be raised.

See Also

- [Restrictions](#)

5.10.1.15.2.3 Restrictions Property

Used to provide one or more conditions restricting the list of objects to be described.

Class

[TDAMetaData](#)

Syntax

```
property Restrictions: TStrings;
```

Remarks

Use the Restriction list to provide one or more conditions restricting the list of objects to be described. To see the full list of restrictions and to which metadata kinds they are applicable, you should assign the Restrictions value to the MetadataKind property and view the result.

See Also

- [MetadataKind](#)

5.10.1.15.3 Methods

Methods of the **TDAMetaData** class.

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

Public

Name	Description
ApplyRange (inherited from TMemDataSet)	Applies a range to the dataset.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.

CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
EditRangeEnd (inherited from TMemDataSet)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from TMemDataSet)	Enables changing the starting value for an existing range.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds	Used to get values acceptable in the MetaDataKind property.
GetRestrictions	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from TMemDataSet)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from TMemDataSet)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from TMemDataSet)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.

RevertRecord (inherited from TMemDataSet)	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.

See Also

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

5.10.1.15.3.1 GetMetaDataKinds Method

Used to get values acceptable in the MetaDataKind property.

Class

[TDAMetaData](#)

Syntax

```
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](#)

5.10.1.15.3.2 GetRestrictions Method

Used to find out which restrictions are applicable to a certain `MetaDataKind`.

Class

[TDAMetaData](#)

Syntax

```
procedure GetRestrictions(List: TStrings; const MetaDataKind:  
string);
```

Parameters

List

Holds the object that will be filled with metadata kinds (restrictions).

MetaDataKind

Holds the metadata kind for which restrictions are returned.

Remarks

Call the `GetRestrictions` method to find out which restrictions are applicable to a certain `MetaDataKind`. The `List` parameter will be cleared and then filled with values.

See Also

- [Restrictions](#)
- [GetMetaDataKinds](#)

5.10.1.16 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](#)
- [TCustomDASQL](#)
- [TDAParams](#)

5.10.1.16.1 Members

[TDAParam](#) class overview.

Properties

Name	Description
AsBlob	Used to set and read the value of the BLOB parameter as string.
AsBlobRef	Used to set and read the value of the BLOB parameter as a TBlob object.
AsFloat	Used to assign the value for a float field to a parameter.
AsInteger	Used to assign the value for an integer field to the parameter.
AsLargeInt	Used to assign the value for a LargeInteger field to the parameter.

AsMemo	Used to assign the value for a memo field to the parameter.
AsMemoRef	Used to set and read the value of the memo parameter as a TBlob object.
AsSQLTimeStamp	Used to specify the value of the parameter when it represents a SQL timestamp field.
AsString	Used to assign the string value to the parameter.
AsWideString	Used to assign the Unicode string value to the parameter.
DataType	Indicates the data type of the parameter.
IsNull	Used to indicate whether the value assigned to a parameter is NULL.
ParamType	Used to indicate the type of use for a parameter.
Size	Specifies the size of a string type parameter.
Value	Used to represent the value of the parameter as Variant.

Methods

Name	Description
AssignField	Assigns field name and field value to a param.
AssignFieldValue	Assigns the specified field properties and value to a parameter.
LoadFromFile	Places the content of a specified file into a TDAParam object.
LoadFromStream	Places the content from a stream into a TDAParam object.
SetBlobData	Overloaded. Writes the data from a specified buffer to BLOB.

5.10.1.16.2 Properties

Properties of the **TDAParam** class.

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

Public

Name	Description
AsBlob	Used to set and read the value of the BLOB parameter as string.
AsBlobRef	Used to set and read the value of the BLOB parameter as a TBlob object.
AsFloat	Used to assign the value for a float field to a parameter.
AsInteger	Used to assign the value for an integer field to the parameter.
AsLargeInt	Used to assign the value for a LargeInteger field to the parameter.
AsMemo	Used to assign the value for a memo field to the parameter.
AsMemoRef	Used to set and read the value of the memo parameter as a TBlob object.
AsSQLTimeStamp	Used to specify the value of the parameter when it represents a SQL timestamp field.
AsString	Used to assign the string value to the parameter.
AsWideString	Used to assign the Unicode string value to the parameter.
IsNull	Used to indicate whether the value assigned to a parameter is NULL.

Published

Name	Description
DataType	Indicates the data type of the parameter.

ParamType	Used to indicate the type of use for a parameter.
Size	Specifies the size of a string type parameter.
Value	Used to represent the value of the parameter as Variant.

See Also

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

5.10.1.16.2.1 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.

5.10.1.16.2.2 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.

5.10.1.16.2.3 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.

5.10.1.16.2.4 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.

5.10.1.16.2.5 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.

5.10.1.16.2.6 AsMemo Property

Used to assign the value for a memo field to the parameter.

Class

[TDAParam](#)

Syntax

```
property AsMemo: string;
```

Remarks

Use the AsMemo property to assign the value for a memo field to the parameter. Setting AsMemo will set the DataType property to ftMemo.

5.10.1.16.2.7 AsMemoRef Property

Used to set and read the value of the memo parameter as a TBlob object.

Class

[TDAParam](#)

Syntax

```
property AsMemoRef: TBlob;
```

Remarks

Use the AsMemoRef property to set and read the value of the memo parameter as a TBlob object. Setting AsMemoRef will set the DataType property to ftMemo.

5.10.1.16.2.8 AsSQLTimeStamp Property

Used to specify the value of the parameter when it represents a SQL timestamp field.

Class

[TDAParam](#)

Syntax

```
property AsSQLTimeStamp: TSQLTimeStamp;
```

Remarks

Set the AsSQLTimeStamp property to assign the value for a SQL timestamp field to the parameter. Setting AsSQLTimeStamp sets the DataType property to ftTimeStamp.

5.10.1.16.2.9 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.

5.10.1.16.2.10 AsWideString Property

Used to assign the Unicode string value to the parameter.

Class

[TDAParam](#)

Syntax

```
property AswideString: string;
```

Remarks

Set `AsString` to assign the Unicode string value to the parameter. Setting `AsString` will set the `DataType` property to `ftWideString`.

Read the `AsString` 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.

5.10.1.16.2.11 `DataType` Property

Indicates the data type of the parameter.

Class

[TDAParam](#)

Syntax

```
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.

5.10.1.16.2.12 `IsNull` Property

Used to indicate whether the value assigned to a parameter is `NULL`.

Class

[TDAParam](#)

Syntax

```
property IsNull: boolean;
```

Remarks

Use the `IsNull` property to indicate whether the value assigned to a parameter is `NULL`.

5.10.1.16.2.13 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.

5.10.1.16.2.14 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.

5.10.1.16.2.15 Value Property

Used to represent the value of the parameter as Variant.

Class

[TDAParam](#)

Syntax

```
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.

5.10.1.16.3 Methods

Methods of the **TDAParam** class.

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

Public

Name	Description
AssignField	Assigns field name and field value to a param.
AssignFieldValue	Assigns the specified field properties and value to a parameter.
LoadFromFile	Places the content of a specified file into a TDAParam object.
LoadFromStream	Places the content from a stream into a TDAParam object.
SetBlobData	Overloaded. Writes the data from a specified buffer to BLOB.

See Also

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

5.10.1.16.3.1 AssignField Method

Assigns field name and field value to a param.

Class

[TDAParam](#)

Syntax

```
procedure AssignField(Field: TField);
```

Parameters

Field

Holds the field which name and value should be assigned to the param.

Remarks

Call the AssignField method to assign field name and field value to a param.

5.10.1.16.3.2 AssignFieldValue Method

Assigns the specified field properties and value to a parameter.

Class

[TDAParam](#)

Syntax

```
procedure AssignFieldValue(Field: TField; const value: Variant);  
virtual;
```

Parameters*Field*

Holds the field the properties of which will be assigned to the parameter.

Value

Holds the value for the parameter.

Remarks

Call the AssignFieldValue method to assign the specified field properties and value to a parameter.

5.10.1.16.3.3 LoadFromFile Method

Places the content of a specified file into a TDAParam object.

Class

[TDAParam](#)

Syntax

```
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](#)

5.10.1.16.3.4 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](#)

5.10.1.16.3.5 SetBlobData Method

Writes the data from a specified buffer to BLOB.

Class

[TDAParam](#)

Overload List

Name	Description
SetBlobData(Buffer: TValueBuffer)	Writes the data from a specified buffer to BLOB.
SetBlobData(Buffer: IntPtr; Size: Integer)	Writes the data from a specified buffer to BLOB.

Writes the data from a specified buffer to BLOB.

Class

[TDAParam](#)

Syntax

```
procedure SetBlobData(Buffer: TValueBuffer); overload;
```

Parameters

Buffer

Holds the pointer to the data.

Writes the data from a specified buffer to BLOB.

Class

[TDAParam](#)

Syntax

```
procedure SetBlobData(Buffer: IntPtr; Size: Integer); overload;
```

Parameters

Buffer

Holds the pointer to data.

Size

Holds the number of bytes to read from the buffer.

Remarks

Call the SetBlobData method to write data from a specified buffer to BLOB.

5.10.1.17 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](#)

5.10.1.17.1 Members

[TDAParams](#) class overview.

Properties

Name	Description
Items	Used to iterate through all parameters.

Methods

Name	Description
------	-------------

FindParam	Searches for a parameter with the specified name.
ParamByName	Searches for a parameter with the specified name.

5.10.1.17.2 Properties

Properties of the **TDAParams** class.

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

Public

Name	Description
Items	Used to iterate through all parameters.

See Also

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

5.10.1.17.2.1 Items Property(Indexer)

Used to iterate through all parameters.

Class

[TDAParams](#)

Syntax

```
property Items[Index: integer]: TDAParam; default;
```

Parameters

Index

Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all parameters. Index identifies the index in the range 0..Count - 1. Items can reference a particular parameter by its index, but the ParamByName method is preferred in order to avoid depending on the order of the parameters.

5.10.1.17.3 Methods

Methods of the **TDAParams** class.

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

Public

Name	Description
FindParam	Searches for a parameter with the specified name.
ParamByName	Searches for a parameter with the specified name.

See Also

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

5.10.1.17.3.1 FindParam Method

Searches for a parameter with the specified name.

Class

[TDAParams](#)

Syntax

```
function FindParam(const value: string): TDAParam;
```

Parameters

Value

Holds the parameter name.

Return Value

a parameter, if a match was found. Nil otherwise.

Remarks

Use the FindParam method to find a parameter with the name passed in Value. If a match is found, FindParam returns the parameter. Otherwise, it returns nil. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries. To locate more than one parameter at a time by name, use the GetParamList method instead. To get only the value of a named parameter, use the ParamValues property.

5.10.1.17.3.2 ParamByName Method

Searches for a parameter with the specified name.

Class

[TDAParams](#)

Syntax

```
function ParamByName(const Value: string): TDAParam;
```

Parameters

Value

Holds the parameter name.

Return Value

a parameter, if the match was found. otherwise an exception is raised.

Remarks

Use the ParamByName method to find a parameter with the name passed in Value. If a match was found, ParamByName returns the parameter. Otherwise, an exception is raised.

Use this method rather than a direct reference to the [Items](#) property to avoid depending on the order of the entries.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindParam method.

5.10.1.18 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.

5.10.1.18.1 Members

[TDATransaction](#) class overview.

Properties

Name	Description
Active	Used to determine if the transaction is active.
DefaultCloseAction	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.

Methods

Name	Description
Commit	Commits the current transaction.
Rollback	Discards all modifications of data associated with the current transaction and ends the transaction.
StartTransaction	Begins a new transaction.

Events

Name	Description
OnCommit	Occurs after the transaction has been successfully committed.
OnCommitRetaining	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.
OnRollbackRetaining	Occurs after RollbackRetaining has been executed.

5.10.1.18.2 Properties

Properties of the **TDATransaction** class.

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

Public

Name	Description
Active	Used to determine if the transaction is active.
DefaultCloseAction	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.

See Also

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

5.10.1.18.2.1 Active Property

Used to determine if the transaction is active.

Class

[TDATransaction](#)

Syntax

```
property Active: boolean;
```

Remarks

Indicates whether the transaction is active. This property is read-only.

5.10.1.18.2.2 DefaultCloseAction Property

Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Class

[TDATransaction](#)

Syntax

```
property DefaultCloseAction: TCRTransactionAction default  
taRollback;
```

Remarks

Use DefaultCloseAction to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

5.10.1.18.3 Methods

Methods of the **TDATransaction** class.

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

Public

Name	Description
Commit	Commits the current transaction.
Rollback	Discards all modifications of data associated with the current transaction and ends the transaction.
StartTransaction	Begins a new transaction.

See Also

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

5.10.1.18.3.1 Commit Method

Commits the current transaction.

Class

[TDATransaction](#)

Syntax

```
procedure Commit; virtual;
```

Remarks

Call the Commit method to commit the current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database, and then finishes the transaction.

See Also

- [Rollback](#)
- [StartTransaction](#)

5.10.1.18.3.2 Rollback Method

Discards all modifications of data associated with the current transaction and ends the transaction.

Class

[TDATransaction](#)

Syntax

```
procedure Rollback; virtual;
```

Remarks

Call Rollback to cancel all data modifications made within the current transaction to the database server, and finish the transaction.

See Also

- [Commit](#)
- [StartTransaction](#)

5.10.1.18.3.3 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](#)

5.10.1.18.4 Events

Events of the **TDATransaction** class.

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

Public

Name	Description
OnCommit	Occurs after the transaction has been successfully committed.
OnCommitRetaining	Occurs after <code>CommitRetaining</code> has been executed.
OnError	Used to process errors that occur during executing a transaction.
OnRollback	Occurs after the transaction has been successfully rolled back.
OnRollbackRetaining	Occurs after <code>RollbackRetaining</code> has been executed.

See Also

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

5.10.1.18.4.1 OnCommit Event

Occurs after the transaction has been successfully committed.

Class

[TDATransaction](#)

Syntax

```
property OnCommit: TNotifyEvent;
```

Remarks

The OnCommit event fires when the M:Devart.Dac.TDATransaction.Commit method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.PgDac.TPgTransaction.CommitRetaining() method execution, the [OnCommitRetaining](#) event is used. When an error occurs during commit, the [OnError](#) event fires.

See Also

- [Commit](#)
- M:Devart.PgDac.TPgTransaction.CommitRetaining()
- [OnCommitRetaining](#)
- [OnError](#)

5.10.1.18.4.2 OnCommitRetaining Event

Occurs after CommitRetaining has been executed.

Class

[TDATransaction](#)

Syntax

```
property OnCommitRetaining: TNotifyEvent;
```

Remarks

The OnCommitRetaining event fires when the M:Devart.PgDac.TPgTransaction.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

- M:Devart.PgDac.TPgTransaction.CommitRetaining()
- [Commit](#)
- [OnCommit](#)
- [OnError](#)

5.10.1.18.4.3 OnError Event

Used to process errors that occur during executing a transaction.

Class

[TDATransaction](#)

Syntax

```
property OnError: TDATransactionErrorEvent;
```

Remarks

Add a handler to the OnError event to process errors that occur during executing a transaction control statements such as [Commit](#), [Rollback](#). Check the E parameter to get the error code.

See Also

- [Commit](#)
- [Rollback](#)
- [StartTransaction](#)

5.10.1.18.4.4 OnRollback Event

Occurs after the transaction has been successfully rolled back.

Class

[TDATransaction](#)

Syntax

```
property OnRollback: TNotifyEvent;
```

Remarks

The OnRollback event fires when the M:Devart.Dac.TDATransaction.Rollback method is executed, just after the transaction is successfully rolled back. In order to respond to the

M:Devart.PgDac.TPgTransaction.RollbackRetaining() method execution, the [OnRollbackRetaining](#) event is used.

When an error occurs during rollback, the [OnError](#) event fired.

See Also

- [Rollback](#)
- M:Devart.PgDac.TPgTransaction.RollbackRetaining()
- [OnRollbackRetaining](#)
- [OnError](#)

5.10.1.18.4.5 OnRollbackRetaining Event

Occurs after RollbackRetaining has been executed.

Class

[TDATransaction](#)

Syntax

```
property OnRollbackRetaining: TNotifyEvent;
```

Remarks

The OnRollbackRetaining event fires when the M:Devart.PgDac.TPgTransaction.RollbackRetaining() method is executed, just after the transaction is successfully rolled back. In order to respond to the M:Devart.Dac.TDATransaction.Rollback method execution, the [OnRollback](#) event is used. When an error occurs during rollback, the [OnError](#) event fired.

See Also

- [Rollback](#)
- M:Devart.PgDac.TPgTransaction.RollbackRetaining()
- [OnRollback](#)
- [OnError](#)

5.10.1.19 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](#)

5.10.1.19.1 Members

[TMacro](#) class overview.

Properties

Name	Description
Active	Used to determine if the macro should be expanded.
AsDateTime	Used to set the TDateTime value to a macro.
AsFloat	Used to set the float value to a macro.
AsInteger	Used to set the integer value to a macro.
AsString	Used to assign the string value to a macro.
Name	Used to identify a particular macro.
Value	Used to set the value to a macro.

5.10.1.19.2 Properties

Properties of the **TMacro** class.

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

Public

Name	Description
AsDateTime	Used to set the TDateTime value to a macro.
AsFloat	Used to set the float value to a macro.
AsInteger	Used to set the integer value to a macro.
AsString	Used to assign the string value to a macro.

Published

Name	Description
Active	Used to determine if the macro should be expanded.
Name	Used to identify a particular macro.
Value	Used to set the value to a macro.

See Also

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

5.10.1.19.2.1 Active Property

Used to determine if the macro should be expanded.

Class

[TMacro](#)

Syntax

```
property Active: boolean default True;
```

Remarks

When set to True, the macro will be expanded, otherwise macro definition is replaced by null string. You can use the Active property to modify the SQL property.

The default value is True.

Example

```
PgQuery.SQL.Text := 'SELECT * FROM Dept WHERE DeptNo > 20 &Cond1';  
PgQuery.Macros[0].Value := 'and DName is NULL';  
PgQuery.Macros[0].Active:= False;
```

5.10.1.19.2.2 AsDateTime Property

Used to set the TDateTime value to a macro.

Class

[TMacro](#)

Syntax

```
property AsDateTime: TDateTime;
```

Remarks

Use the AsDateTime property to set the TDateTime value to a macro.

5.10.1.19.2.3 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.

5.10.1.19.2.4 AsInteger Property

Used to set the integer value to a macro.

Class

[TMacro](#)

Syntax

```
property AsInteger: integer;
```

Remarks

Use the AsInteger property to set the integer value to a macro.

5.10.1.19.2.5 AsString Property

Used to assign the string value to a macro.

Class

[TMacro](#)

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to assign the string value to a macro. Read the AsString property to determine the value of macro represented as a string.

5.10.1.19.2.6 Name Property

Used to identify a particular macro.

Class

[TMacro](#)

Syntax

```
property Name: string;
```

Remarks

Use the Name property to identify a particular macro.

5.10.1.19.2.7 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.

5.10.1.20 TMacros Class

Controls a list of TMacro objects for the [TCustomDASQL.Macros](#) or [TCustomDADataset](#) components.

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

Unit

[DBAccess](#)

Syntax

```
TMacros = class(TCollection);
```

Remarks

Use TMacros to manage a list of TMacro objects for the [TCustomDASQL](#) or [TCustomDADataset](#) components.

See Also

- [TMacro](#)

5.10.1.20.1 Members

[TMacros](#) class overview.

Properties

Name	Description
Items	Used to iterate through all the macros parameters.

Methods

Name	Description
AssignValues	Copies the macros values and properties from the specified source.
Expand	Changes the macros in the passed SQL statement to their values.
FindMacro	Searches for a TMacro object by its name.
IsEqual	Compares itself with another TMacro object.
MacroByName	Used to search for a macro with the specified name.
Scan	Creates a macros from the passed SQL statement.

5.10.1.20.2 Properties

Properties of the **TMacros** class.

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

Public

Name	Description
Items	Used to iterate through all the macros parameters.

See Also

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

5.10.1.20.2.1 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.

5.10.1.20.3 Methods

Methods of the **TMacros** class.

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

Public

Name	Description
AssignValues	Copies the macros values and properties from the specified source.
Expand	Changes the macros in the passed SQL statement to their values.
FindMacro	Searches for a TMacro object by its name.
IsEqual	Compares itself with another TMacro object.
MacroByName	Used to search for a macro with the specified name.
Scan	Creates a macros from the passed SQL statement.

See Also

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

5.10.1.20.3.1 AssignValues Method

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.

5.10.1.20.3.2 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.

5.10.1.20.3.3 FindMacro Method

Searches for a TMacro object by its 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 was found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the name passed in Value. If a match is found, FindMacro returns the macro. 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.

5.10.1.20.3.4 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.

5.10.1.20.3.5 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 rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate a macro by name without raising an exception if the parameter is not found, use the FindMacro method.

5.10.1.20.3.6 Scan Method

Creates a macros from the passed SQL statement.

Class

[TMacros](#)

Syntax

```
procedure Scan(const SQL: string);
```

Parameters*SQL*

Holds the passed SQL statement.

Remarks

Call the Scan method to create a macros from the passed SQL statement. On that all existing TMacro objects are cleared.

5.10.1.21 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);
```

5.10.1.21.1 Members

[TPoolingOptions](#) class overview.

Properties

Name	Description
ConnectionLifetime	Used to specify the maximum time during which an opened connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of connections that can be opened in the connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

5.10.1.21.2 Properties

Properties of the **TPoolingOptions** class.

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

Published

Name	Description
ConnectionLifetime	Used to specify the maximum time during which an opened connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of connections that can be opened in the connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

See Also

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

5.10.1.21.2.1 ConnectionLifetime Property

Used to specify the maximum time during which an opened connection can be used by connection pool.

Class

[TPoolingOptions](#)

Syntax

```
property ConnectionLifetime: integer default  
DefValConnectionLifetime;
```

Remarks

Use the ConnectionLifeTime property to specify the maximum time during which an opened connection can be used by connection pool. Measured in milliseconds. Pool deletes connections with exceeded connection lifetime when [TCustomDAConnection](#) is about to close. If the ConnectionLifetime property is set to 0 (by default), then the lifetime of connection is infinity. ConnectionLifetime concerns only inactive connections in the pool.

5.10.1.21.2.2 MaxPoolSize Property

Used to specify the maximum number of connections that can be opened in connection pool.

Class

[TPoolingOptions](#)

Syntax

```
property MaxPoolSize: integer default DefValMaxPoolSize;
```

Remarks

Specifies the maximum number of connections that can be opened in connection pool. Once this value is reached, no more connections are opened. The valid values are 1 and higher.

5.10.1.21.2.3 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.

5.10.1.21.2.4 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.

5.10.1.22 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);
```

5.10.1.22.1 Members

[TSmartFetchOptions](#) class overview.

Properties

Name	Description
------	-------------

Enabled	Sets SmartFetch mode enabled or not.
LiveBlock	Used to minimize memory consumption.
PrefetchedFields	List of fields additional to key fields that will be read from the database on dataset open.
SQLGetKeyValues	SQL query for the read key and prefetched fields from the database.

5.10.1.22.2 Properties

Properties of the **TSmartFetchOptions** class.

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

Published

Name	Description
Enabled	Sets SmartFetch mode enabled or not.
LiveBlock	Used to minimize memory consumption.
PrefetchedFields	List of fields additional to key fields that will be read from the database on dataset open.
SQLGetKeyValues	SQL query for the read key and prefetched fields from the database.

See Also

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

5.10.1.22.2.1 Enabled Property

Sets SmartFetch mode enabled or not.

Class

[TSmartFetchOptions](#)

Syntax

```
property Enabled: Boolean default False;
```

5.10.1.22.2.2 LiveBlock Property

Used to minimize memory consumption.

Class

[TSmartFetchOptions](#)

Syntax

```
property LiveBlock: Boolean default True;
```

Remarks

If LiveBlock is True, then on navigating through a dataset forward or backward, memory will be allocated for records count defined in the the FetchRows 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.

5.10.1.22.2.3 PrefetchedFields Property

List of fields additional to key fields that will be read from the database on dataset open.

Class

[TSmartFetchOptions](#)

Syntax

```
property PrefetchedFields: string;
```

Remarks

If you are going to use locate, filter or sort by some fields, then these fields should be added to the prefetched fields list to avoid excessive reading from the database.

5.10.1.22.2.4 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.

5.10.2 Types

Types in the **DBAccess** unit.

Types

Name	Description
TAfterExecuteEvent	This type is used for the TCustomDADataset.AfterExecute and TCustomDASQL.AfterExecute events.
TAfterFetchEvent	This type is used for the TCustomDADataset.AfterFetch event.
TBeforeFetchEvent	This type is used for the TCustomDADataset.BeforeFetch event.
TConnectionLostEvent	This type is used for the TCustomDACConnection.OnConnectionLost event.
TDAConnectionErrorEvent	This type is used for the TCustomDACConnection.OnError event.
TDATransactionErrorEvent	This type is used for the TDATransaction.OnError event.
TRefreshOptions	Represents the set of TRefreshOption .

[TUpdateExecuteEvent](#)

This type is used for the TCustomDADataset.AfterUpdateExecute and TCustomDADataset.BeforeUpdateExecute events.

5.10.2.1 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.

5.10.2.2 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.

5.10.2.3 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.

5.10.2.4 TConnectionLostEvent Procedure Reference

This type is used for the [TCustomDAConnection.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.

RetryMode

The application behavior when connection is lost.

5.10.2.5 TDAConnectionErrorEvent Procedure Reference

This type is used for the [TCustomDAConnection.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 .

5.10.2.6 TDATransactionErrorEvent Procedure Reference

This type is used for the [TDATransaction.OnError](#) event.

Unit

[DBAccess](#)

Syntax

```
TDATransactionErrorEvent = procedure (Sender: TObject; E:  
EDAError; var Fail: boolean) of object;
```

Parameters

Sender

An object that raised the event.

E

The error code.

Fail

False, if an error dialog should be prevented from being displayed and EAbort exception to cancel the current operation should be raised.

5.10.2.7 TRefreshOptions Set

Represents the set of [TRefreshOption](#).

Unit

[DBAccess](#)

Syntax

```
TRefreshOptions = set of TRefreshOption;
```

5.10.2.8 TUpdateExecuteEvent Procedure Reference

This type is used for the TCustomDADataset.AfterUpdateExecute and TCustomDADataset.BeforeUpdateExecute events.

Unit

[DBAccess](#)

Syntax

```
TUpdateExecuteEvent = procedure (Sender: TDataSet; StatementTypes: TStatementTypes; Params: TDAParams) of object;
```

Parameters

Sender

An object that raised the event.

StatementTypes

Holds the type of the SQL statement being executed.

Params

Holds the parameters with which the SQL statement will be executed.

5.10.3 Enumerations

Enumerations in the **DBAccess** unit.

Enumerations

Name	Description
TLabelSet	Sets the language of labels in the connect dialog.
TRefreshOption	Indicates when the editing record will be refreshed.
TRetryMode	Specifies the application behavior when connection is lost.

5.10.3.1 TLabelSet Enumeration

Sets the language of labels in the connect dialog.

Unit

[DBAccess](#)

Syntax

```
TLabelSet = (lsCustom, lsEnglish, lsFrench, lsGerman, lsItalian, lsPolish, lsPortuguese, lsRussian, lsSpanish);
```

Values

Value	Meaning
lsCustom	Set the language of labels in the connect dialog manually.
lsEnglish	Set English as the language of labels in the connect dialog.
lsFrench	Set French as the language of labels in the connect dialog.
lsGerman	Set German as the language of labels in the connect dialog.
lsItalian	Set Italian as the language of labels in the connect dialog.
lsPolish	Set Polish as the language of labels in the connect dialog.
lsPortuguese	Set Portuguese as the language of labels in the connect dialog.
lsRussian	Set Russian as the language of labels in the connect dialog.
lsSpanish	Set Spanish as the language of labels in the connect dialog.

5.10.3.2 TRefreshOption Enumeration

Indicates when the editing record will be refreshed.

Unit

[DBAccess](#)

Syntax

```
TRefreshOption = (roAfterInsert, roAfterUpdate, roBeforeEdit);
```

Values

Value	Meaning
roAfterInsert	Refresh is performed after inserting.
roAfterUpdate	Refresh is performed after updating.
roBeforeEdit	Refresh is performed by Edit method.

5.10.3.3 TRetryMode Enumeration

Specifies the application behavior when connection is lost.

Unit

[DBAccess](#)

Syntax

```
TRetryMode = (rmRaise, rmReconnect, rmReconnectExecute);
```

Values

Value	Meaning
rmRaise	An exception is raised.
rmReconnect	Reconnect is performed and then exception is raised.
rmReconnectExecute	Reconnect is performed and abortive operation is reexecuted. Exception is not raised.

5.10.4 Variables

Variables in the **DBAccess** unit.

Variables

Name	Description
BaseSQLOldBehavior	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.
ChangeCursor	When set to True allows data access components to change screen cursor for the execution time.
SQLGeneratorCompatibility	The value of the TCustomDADDataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause.

5.10.4.1 BaseSQLOldBehavior Variable

After assigning SQL text and modifying it by [AddWhere](#), [DeleteWhere](#), and [SetOrderBy](#), all subsequent changes of the SQL property will not be reflected in the BaseSQL property.

Unit

[DBAccess](#)

Syntax

```
BaseSQLOldBehavior: boolean = False;
```

Remarks

The [BaseSQL](#) property is similar to the SQL property, but it does not store changes made by the [AddWhere](#), [DeleteWhere](#), and [SetOrderBy](#) methods. After assigning SQL text and modifying it by one of these methods, all subsequent changes of the SQL property will not be reflected in the BaseSQL property. This behavior was changed in PgDAC . To restore old behavior, set the BaseSQLOldBehavior variable to True.

5.10.4.2 ChangeCursor Variable

When set to True allows data access components to change screen cursor for the execution time.

Unit

[DBAccess](#)

Syntax

```
ChangeCursor: boolean = True;
```

5.10.4.3 SQLGeneratorCompatibility Variable

The value of the [TCustomDADataset.BaseSQL](#) property is used to complete the refresh SQL statement, if the manually assigned [TCustomDAUpdateSQL.RefreshSQL](#) property contains only WHERE clause.

Unit

[DBAccess](#)

Syntax

```
SQLGeneratorCompatibility: boolean = False;
```

Remarks

If the manually assigned [TCustomDAUpdateSQL.RefreshSQL](#) property contains only WHERE clause, PgDAC uses the value of the [TCustomDADataset.BaseSQL](#) property to complete the refresh SQL statement. In this situation all modifications applied to the SELECT query by functions [TCustomDADataset.AddWhere](#), [TCustomDADataset.DeleteWhere](#) are not taken into account. This behavior was changed in PgDAC . To restore the old behavior, set the BaseSQLOldBehavior variable to True.

5.11 MemData

This unit contains classes for storing data in memory.

Classes

Name	Description
TAttribute	TAttribute is not used in PgDAC.
TBlob	Holds large object value for field and parameter dtBlob, dtMemo data types.
TCompressedBlob	Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.
TDBObject	A base class for classes that work with user-defined data types that have attributes.
TMemData	Implements storing data in memory.
TObjectType	This class is not used.
TSharedObject	A base class that allows to simplify memory management for object referenced by several other objects.

Types

Name	Description
TLocateExOptions	Represents the set of TLocateExOption .

TUpdateRecKinds	Represents the set of TUpdateRecKind.
---------------------------------	---------------------------------------

Enumerations

Name	Description
TCompressBlobMode	Specifies when the values should be compressed and the way they should be stored.
TConnLostCause	Specifies the cause of the connection loss.
TDANumericType	Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.
TLocateExOption	Allows to set additional search parameters which will be used by the LocateEx method.
TSortType	Specifies a sort type for string fields.
TUpdateRecKind	Indicates records for which the ApplyUpdates method will be performed.

5.11.1 Classes

Classes in the **MemData** unit.

Classes

Name	Description
TAttribute	TAttribute is not used in PgDAC.
TBlob	Holds large object value for field and parameter dtBlob, dtMemo data types.
TCompressedBlob	Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.
TDBObject	A base class for classes that work with user-defined data types that have attributes.

TMemData	Implements storing data in memory.
TObjectType	This class is not used.
TSharedObject	A base class that allows to simplify memory management for object referenced by several other objects.

5.11.1.1 TAttribute Class

TAttribute is not used in PgDAC.

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

Unit

[MemData](#)

Syntax

```
TAttribute = class(System.TObject);
```

5.11.1.1.1 Members

[TAttribute](#) class overview.

Properties

Name	Description
AttributeNo	Returns an attribute's ordinal position in object.
DataSize	Returns the size of an attribute value in internal representation.
DataType	Returns the type of data that was assigned to the Attribute.
Length	Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.
ObjectType	Returns a TObjectType object for an object attribute.
Offset	Returns an offset of the attribute value in internal representation.

Owner	Indicates TObjectType that uses the attribute to represent one of its attributes.
Scale	Returns the scale of dtFloat and dtInteger attributes.
Size	Returns the size of an attribute value in external representation.

5.11.1.1.2 Properties

Properties of the **TAttribute** class.

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

Public

Name	Description
AttributeNo	Returns an attribute's ordinal position in object.
DataSize	Returns the size of an attribute value in internal representation.
DataType	Returns the type of data that was assigned to the Attribute.
Length	Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.
ObjectType	Returns a TObjectType object for an object attribute.
Offset	Returns an offset of the attribute value in internal representation.
Owner	Indicates TObjectType that uses the attribute to represent one of its attributes.
Scale	Returns the scale of dtFloat and dtInteger attributes.
Size	Returns the size of an attribute value in external representation.

See Also

- [TAttribute Class](#)

- [TAttribute Class Members](#)

5.11.1.1.2.1 AttributeNo Property

Returns an attribute's ordinal position in object.

Class

[TAttribute](#)

Syntax

```
property AttributeNo: word;
```

Remarks

Use the AttributeNo property to learn an attribute's ordinal position in object, where 1 is the first field.

See Also

- [TObjectType.Attributes](#)

5.11.1.1.2.2 DataSize Property

Returns the size of an attribute value in internal representation.

Class

[TAttribute](#)

Syntax

```
property DataSize: Integer;
```

Remarks

Use the DataSize property to learn the size of an attribute value in internal representation.

5.11.1.1.2.3 DataType Property

Returns the type of data that was assigned to the Attribute.

Class

[TAttribute](#)

Syntax

```
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.

5.11.1.1.2.4 Length Property

Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.

Class

[TAttribute](#)

Syntax

```
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](#)

5.11.1.1.2.5 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.

5.11.1.1.2.6 Offset Property

Returns an offset of the attribute value in internal representation.

Class

[TAttribute](#)

Syntax

```
property offset: Integer;
```

Remarks

Use the DataSize property to learn an offset of the attribute value in internal representation.

5.11.1.1.2.7 Owner Property

Indicates TObjectType that uses the attribute to represent one of its attributes.

Class

[TAttribute](#)

Syntax

```
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.

5.11.1.1.2.8 Scale Property

Returns the scale of dtFloat and dtInteger attributes.

Class

[TAttribute](#)

Syntax

```
property scale: word;
```

Remarks

Use the Scale property to learn the scale of dtFloat and dtInteger attributes.

See Also

- [Length](#)

5.11.1.1.2.9 Size Property

Returns the size of an attribute value in external representation.

Class

[TAttribute](#)

Syntax

```
property Size: Integer;
```

Remarks

Read Size to learn the size of an attribute value in external representation.

For example:

dtDate	8 (sizeof(TDateTime))
dtFloat	8 (sizeof(Double))
dtInteger	4 (sizeof(Integer))

See Also

- [DataSize](#)

5.11.1.2 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

```
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](#)

5.11.1.2.1 Members

[TBlob](#) class overview.

Properties

Name	Description
AsString	Used to manipulate BLOB value as string.
AsWideString	Used to manipulate BLOB value as Unicode string.
IsUnicode	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Sets BLOB value from another TBlob object.
Clear	Deletes the current value in TBlob object.
LoadFromFile	Loads the contents of a file into a TBlob object.
LoadFromStream	Copies the contents of a stream into the TBlob object.

Read	Acquires a raw sequence of bytes from the data stored in TBlob.
Release (inherited from TSharedObject)	Decrements the reference count.
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.

5.11.1.2.2 Properties

Properties of the **TBlob** class.

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

Public

Name	Description
AsString	Used to manipulate BLOB value as string.
AsWideString	Used to manipulate BLOB value as Unicode string.
IsUnicode	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of the TBlob value in bytes.

See Also

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

5.11.1.2.2.1 AsString Property

Used to manipulate BLOB value as string.

Class

[TBlob](#)

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to manipulate BLOB value as string.

See Also

- [Assign](#)
- [AsWideString](#)

5.11.1.2.2.2 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](#)

5.11.1.2.2.3 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.

5.11.1.2.2.4 Size Property

Used to learn the size of the TBlob value in bytes.

Class

[TBlob](#)

Syntax

```
property Size: Cardinal;
```

Remarks

Use the Size property to find out the size of the TBlob value in bytes.

5.11.1.2.3 Methods

Methods of the **TBlob** class.

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

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Sets BLOB value from another TBlob object.
Clear	Deletes the current value in TBlob object.
LoadFromFile	Loads the contents of a file into a TBlob object.
LoadFromStream	Copies the contents of a stream into the TBlob object.

Read	Acquires a raw sequence of bytes from the data stored in TBlob.
Release (inherited from TSharedObject)	Decrements the reference count.
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](#)

5.11.1.2.3.1 Assign Method

Sets BLOB value from another TBlob object.

Class

[TBlob](#)

Syntax

```
procedure Assign(Source: TBlob);
```

Parameters

Source

Holds the BLOB from which the value to the current object will be assigned.

Remarks

Call the Assign method to set BLOB value from another TBlob object.

See Also

- [LoadFromStream](#)
- [AsString](#)
- [AsWideString](#)

5.11.1.2.3.2 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.

5.11.1.2.3.3 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](#)

5.11.1.2.3.4 LoadFromStream Method

Copies the contents of a stream into the TBlob object.

Class

[TBlob](#)

Syntax

```
procedure LoadFromStream(Stream: TStream); virtual;
```

Parameters

Stream

Holds the specified stream from which the field's value is copied.

Remarks

Call the LoadFromStream method to copy the contents of a stream into the TBlob object. Specify the stream from which the field's value is copied as the value of the Stream parameter.

See Also

- [SaveToStream](#)

5.11.1.2.3.5 Read Method

Acquires a raw sequence of bytes from the data stored in TBlob.

Class

[TBlob](#)

Syntax

```
function Read(Position: Cardinal; Count: Cardinal; Dest: IntPtr):  
Cardinal; virtual;
```

Parameters

Position

Holds the starting point of the byte sequence.

Count

Holds the size of the sequence in bytes.

Dest

Holds a pointer to the memory area where to store the sequence.

Return Value

Actually read byte count if the sequence crosses object size limit.

Remarks

Call the Read method to acquire a raw sequence of bytes from the data stored in TBlob. The Position parameter is the starting point of byte sequence which lasts Count number of

bytes. The Dest parameter is a pointer to the memory area where to store the sequence. If the sequence crosses object size limit, function will return actually read byte count.

See Also

- [Write](#)

5.11.1.2.3.6 SaveToFile Method

Saves the contents of the TBlob object to a file.

Class

[TBlob](#)

Syntax

```
procedure SaveToFile(const FileName: string);
```

Parameters

FileName

Holds a string that contains the name of the file.

Remarks

Call the SaveToFile method to save the contents of the TBlob object to a file. Specify the name of the file as the value of the FileName parameter.

See Also

- [LoadFromFile](#)

5.11.1.2.3.7 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](#)

5.11.1.2.3.8 Truncate Method

Sets new `TBlob` size and discards all data over it.

Class

[TBlob](#)

Syntax

```
procedure Truncate(NewSize: Cardinal); virtual;
```

Parameters

NewSize

Holds the new size of `TBlob`.

Remarks

Call the `Truncate` method to set new `TBlob` size and discard all data over it. If `NewSize` is greater or equal `TBlob.Size`, it does nothing.

5.11.1.2.3.9 Write Method

Stores a raw sequence of bytes into a `TBlob` object.

Class

[TBlob](#)

Syntax

```
procedure write(Position: Cardinal; Count: Cardinal; Source:  
IntPtr); virtual;
```

Parameters

Position

Holds the starting point of the byte sequence.

Count

Holds the size of the sequence in bytes.

Source

Holds a pointer to a source memory area.

Remarks

Call the Write method to store a raw sequence of bytes into a TBlob object.

The Position parameter is the starting point of byte sequence which lasts Count number of bytes. The Source parameter is a pointer to a source memory area.

If the value of the Position parameter crosses current size limit of TBlob object, source data will be appended to the object data.

See Also

- [Read](#)

5.11.1.3 TCompressedBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.

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

Unit

[MemData](#)

Syntax

```
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

```
type  
TCompressProc = function(dest: IntPtr; destLen: IntPtr; const source: IntPtr): IntPtr;  
TUncompressProc = function(dest: IntPtr; destLen: IntPtr; source: IntPtr): IntPtr;  
var
```

```
CompressProc: TCompressProc;
UncompressProc: TUncompressProc;
```

Inheritance Hierarchy

[TSharedObject](#)

[TBlob](#)

TCompressedBlob

See Also

- [TBlob](#)
- [TMemDataSet.GetBlob](#)
- [TCustomDADataset.Options](#)

5.11.1.3.1 Members

[TCompressedBlob](#) class overview.

Properties

Name	Description
AsString (inherited from TBlob)	Used to manipulate BLOB value as string.
AsWideString (inherited from TBlob)	Used to manipulate BLOB value as Unicode string.
Compressed	Used to indicate if the Blob is compressed.
CompressedSize	Used to indicate compressed size of the Blob data.
IsUnicode (inherited from TBlob)	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size (inherited from TBlob)	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
------	-------------

AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TBlob)	Sets BLOB value from another TBlob object.
Clear (inherited from TBlob)	Deletes the current value in TBlob object.
LoadFromFile (inherited from TBlob)	Loads the contents of a file into a TBlob object.
LoadFromStream (inherited from TBlob)	Copies the contents of a stream into the TBlob object.
Read (inherited from TBlob)	Acquires a raw sequence of bytes from the data stored in TBlob.
Release (inherited from TSharedObject)	Decrements the reference count.
SaveToFile (inherited from TBlob)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from TBlob)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from TBlob)	Sets new TBlob size and discards all data over it.
Write (inherited from TBlob)	Stores a raw sequence of bytes into a TBlob object.

5.11.1.3.2 Properties

Properties of the **TCompressedBlob** class.

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

Public

Name	Description
AsString (inherited from TBlob)	Used to manipulate BLOB value as string.
AsWideString (inherited from TBlob)	Used to manipulate BLOB value as Unicode string.

Compressed	Used to indicate if the Blob is compressed.
CompressedSize	Used to indicate compressed size of the Blob data.
IsUnicode (inherited from TBlob)	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size (inherited from TBlob)	Used to learn the size of the TBlob value in bytes.

See Also

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

5.11.1.3.2.1 Compressed Property

Used to indicate if the Blob is compressed.

Class

[TCompressedBlob](#)

Syntax

```
property Compressed: boolean;
```

Remarks

Indicates whether the Blob is compressed. Set this property to True or False to compress or decompress the Blob.

5.11.1.3.2.2 CompressedSize Property

Used to indicate compressed size of the Blob data.

Class

[TCompressedBlob](#)

Syntax

```
property CompressedSize: Cardinal;
```


Remarks

Indicates compressed size of the Blob data.

5.11.1.4 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

5.11.1.4.1 Members

[TDBObject](#) class overview.

Properties

Name	Description
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.

Release (inherited from TSharedObject)	Decrements the reference count.
---	---------------------------------

5.11.1.5 TMemData Class

Implements storing data in memory.

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

Unit

[MemData](#)

Syntax

```
TMemData = class(TData);
```

Inheritance Hierarchy

TData

TMemData

5.11.1.5.1 Members

[TMemData](#) class overview.

5.11.1.6 TObjectType Class

This class is not used.

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

Unit

[MemData](#)

Syntax

```
TObjectType = class(TSharedObject);
```

Inheritance Hierarchy

[TSharedObject](#)

TObjectType

5.11.1.6.1 Members

[TObjectType](#) class overview.

Properties

Name	Description
AttributeCount	Used to indicate the number of attributes of type.
Attributes	Used to access separate attributes.
DataType	Used to indicate the type of object dtObject, dtArray or dtTable.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of an object instance.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
FindAttribute	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.11.1.6.2 Properties

Properties of the **TObjectType** class.

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

Public

Name	Description
------	-------------

AttributeCount	Used to indicate the number of attributes of type.
Attributes	Used to access separate attributes.
DataType	Used to indicate the type of object dtObject, dtArray or dtTable.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of an object instance.

See Also

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

5.11.1.6.2.1 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.

5.11.1.6.2.2 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](#)

5.11.1.6.2.3 `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`.

See Also

- `T:Devart.Dac.Units.MemData`

5.11.1.6.2.4 `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](#)

5.11.1.6.3 Methods

Methods of the **TObjectType** class.

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

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
FindAttribute	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

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

5.11.1.6.3.1 FindAttribute Method

Indicates whether a specified Attribute component is referenced in the TAttributes object.

Class

[TObjectType](#)

Syntax

```
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](#)
- `M:Devart.Dac.TObjectType.AttributeByName(System.String)`
- [Attributes](#)

5.11.1.7 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](#)

5.11.1.7.1 Members

[TSharedObject](#) class overview.

Properties

Name	Description
------	-------------

[RefCount](#)

Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef	Increments the reference count for the number of references dependent on the TSharedObject object.
Release	Decrements the reference count.

5.11.1.7.2 Properties

Properties of the **TSharedObject** class.

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

Public

Name	Description
RefCount	Used to return the count of reference to a TSharedObject object.

See Also

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

5.11.1.7.2.1 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.

5.11.1.7.3 Methods

Methods of the **TSharedObject** class.

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

Public

Name	Description
AddRef	Increments the reference count for the number of references dependent on the TSharedObject object.
Release	Decrements the reference count.

See Also

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

5.11.1.7.3.1 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](#)

5.11.1.7.3.2 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](#)

5.11.2 Types

Types in the **MemData** unit.

Types

Name	Description
TLocateExOptions	Represents the set of TLocateExOption .
TUpdateRecKinds	Represents the set of TUpdateRecKind.

5.11.2.1 TLocateExOptions Set

Represents the set of [TLocateExOption](#).

Unit

[MemData](#)

Syntax

```
TLocateExOptions = set of TLocateExOption;
```

5.11.2.2 TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

[MemData](#)

Syntax

```
TUpdateReckinds = set of TUpdateReckind;
```

5.11.3 Enumerations

Enumerations in the **MemData** unit.

Enumerations

Name	Description
TCompressBlobMode	Specifies when the values should be compressed and the way they should be stored.
TConnLostCause	Specifies the cause of the connection loss.
TDANumericType	Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.
TLocateExOption	Allows to set additional search parameters which will be used by the LocateEx method.
TSortType	Specifies a sort type for string fields.
TUpdateReckind	Indicates records for which the ApplyUpdates method will be performed.

5.11.3.1 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

Value	Meaning
cbClient	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.
cbClientServer	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. Note: On using cbServer or cbClientServer data on the server is stored as compressed. Other applications can add records in uncompressed format but can't read and write already compressed data. If compressed BLOB is partially changed by another application (if signature was not changed), DAC will consider its value as NULL.Blob compression is not applied to Memo fields because of possible cutting.
cbNone	Values not compressed. The default value.
cbServer	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.

5.11.3.2 TConnLostCause Enumeration

Specifies the cause of the connection loss.

Unit

[MemData](#)

Syntax

```
TConnLostCause = (clUnknown, clExecute, clOpen, clRefresh, clApply, clServiceQuery, clTransStart, clConnectionApply, clConnect);
```

Values

Value	Meaning
clApply	Connection loss detected during DataSet.ApplyUpdates (Reconnect/Reexecute possible).
clConnect	Connection loss detected during connection establishing (Reconnect possible).
clConnectionApply	Connection loss detected during Connection.ApplyUpdates (Reconnect/Reexecute possible).

clExecute	Connection loss detected during SQL execution (Reconnect with exception is possible).
clOpen	Connection loss detected during execution of a SELECT statement (Reconnect with exception possible).
clRefresh	Connection loss detected during query opening (Reconnect/Reexecute possible).
clServiceQuery	Connection loss detected during service information request (Reconnect/Reexecute possible).
clTransStart	Connection loss detected during transaction start (Reconnect/Reexecute possible). clTransStart has less priority then clConnectionApply.
clUnknown	The connection loss reason is unknown.

5.11.3.3 TDANumericType Enumeration

Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.

Unit

[MemData](#)

Syntax

```
TDANumericType = (ntFloat, ntBCD, ntFmtBCD);
```

Values

Value	Meaning
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 it is slower.

5.11.3.4 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

Value	Meaning
lxCaseInsensitive	Similar to loCaseInsensitive. Key fields and key values are matched without regard to the case.
lxNearest	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.
lxNext	LocateEx searches from the current record.
lxPartialCompare	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'.
lxPartialKey	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'.
lxUp	LocateEx searches from the current record to the first record.

5.11.3.5 TSortType Enumeration

Specifies a sort type for string fields.

Unit

[MemData](#)

Syntax

```
TSortType = (stCaseSensitive, stCaseInsensitive, stBinary);
```

Values

Value	Meaning
stBinary	Sorting by character ordinal values (this comparison is also case sensitive).
stCaseInsensitive	Sorting without case sensitivity.
stCaseSensitive	Sorting with case sensitivity.

5.11.3.6 TUpdateReckind Enumeration

Indicates records for which the ApplyUpdates method will be performed.

Unit

[MemData](#)

Syntax

```
TUpdateReckind = (ukUpdate, ukInsert, ukDelete);
```

Values

Value	Meaning
ukDelete	ApplyUpdates will be performed for deleted records.
ukInsert	ApplyUpdates will be performed for inserted records.
ukUpdate	ApplyUpdates will be performed for updated records.

5.12 MemDS

This unit contains implementation of the TMemDataSet class.

Classes

Name	Description
TMemDataSet	A base class for working with data and manipulating data in memory.

Variables

Name	Description
DoNotRaiseExcetionOnUaFail	An exception will be raised if the value of the UpdateAction parameter is uaFail.
SendDataSetChangeEventAfterOpen	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.

5.12.1 Classes

Classes in the **MemDS** unit.

Classes

Name	Description
TMemDataSet	A base class for working with data and manipulating data in memory.

5.12.1.1 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.

5.12.1.1.1 Members

[TMemDataSet](#) class overview.

Properties

Name	Description
CachedUpdates	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive	Specifies the upper and lower boundaries for a range.
LocalConstraints	Used to avoid setting the Required property of a TField component for

	NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate	Used to prevent implicit 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

Name	Description
ApplyRange	Applies a range to the dataset.
ApplyUpdates	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange	Removes any ranges currently in effect for a dataset.
CancelUpdates	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates	Clears the cached updates buffer.
DeferredPost	Makes permanent changes to the database server.
EditRangeEnd	Enables changing the ending value for an existing range.
EditRangeStart	Enables changing the starting value for an existing range.
GetBlob	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Locate	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare	Allocates resources and creates field components for a dataset.
RestoreUpdates	Marks all records in the cache of updates as unapplied.
RevertRecord	Cancels changes made to the current record when cached updates are enabled.
SaveToXML	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange	Sets the starting and ending values of a range, and applies it.
SetRangeEnd	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
------	-------------

OnUpdateError	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord	Occurs when a single update component can not handle the updates.

5.12.1.1.2 Properties

Properties of the **TMemDataSet** class.

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

Public

Name	Description
CachedUpdates	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive	Specifies the upper and lower boundaries for a range.
LocalConstraints	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate	Used to prevent implicit 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.

See Also

- [TMemDataSet Class](#)

- [TMemDataSet Class Members](#)

5.12.1.1.2.1 CachedUpdates Property

Used to enable or disable the use of cached updates for a dataset.

Class

[TMemDataSet](#)

Syntax

```
property CachedUpdates: boolean default False;
```

Remarks

Use the CachedUpdates property to enable or disable the use of cached updates for a dataset. Setting CachedUpdates to True enables updates to a dataset (such as posting changes, inserting new records, or deleting records) to be stored in an internal cache on the client side instead of being written directly to the dataset's underlying database tables. When changes are completed, an application writes all cached changes to the database in the context of a single transaction.

Cached updates are especially useful for client applications working with remote database servers. Enabling cached updates brings up the following benefits:

- Fewer transactions and shorter transaction times.
- Minimized network traffic.

The potential drawbacks of enabling cached updates are:

- Other applications can access and change the actual data on the server while users are editing local copies of data, resulting in an update conflict when cached updates are applied to the database.
- Other applications cannot access data changes made by an application until its cached updates are applied to the database.

The default value is False.

Note: When establishing master/detail relationship the CachedUpdates property of detail dataset works properly only when [TDADatasetOptions.LocalMasterDetail](#) is set to True.

See Also

- [UpdatesPending](#)
- [TMemDataSet.ApplyUpdates](#)
- [RestoreUpdates](#)
- [CommitUpdates](#)

- [CancelUpdates](#)
- [UpdateStatus](#)
- [TCustomDADataset.Options](#)

5.12.1.1.2.2 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.

Example

The following procedure illustrates how to set IndexFieldNames in response to a button click:

```
DataSet1.IndexFieldNames := 'LastName ASC CIS; DateDue DESC';
```

5.12.1.1.2.3 KeyExclusive Property

Specifies the upper and lower boundaries for a range.

Class

[TMemDataSet](#)

Syntax

```
property KeyExclusive: Boolean;
```

Remarks

Use KeyExclusive to specify whether a range includes or excludes the records that match its specified starting and ending values.

By default, KeyExclusive is False, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to True.

See Also

- [SetRange](#)
- [SetRangeEnd](#)
- [SetRangeStart](#)

5.12.1.1.2.4 LocalConstraints Property

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Class

[TMemDataSet](#)

Syntax

```
property LocalConstraints: boolean default True;
```

Remarks

Use the LocalConstraints property to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet. When LocalConstraints is True, TMemDataSet ignores NOT NULL server constraints. It is useful for tables that have fields updated by triggers.

LocalConstraints is obsolete, and is only included for backward compatibility.

The default value is True.

5.12.1.1.2.5 LocalUpdate Property

Used to prevent implicit update of rows on database server.

Class

[TMemDataSet](#)

Syntax

```
property LocalUpdate: boolean default False;
```

Remarks

Set the LocalUpdate property to True to prevent implicit update of rows on database server. Data changes are cached locally in client memory.

5.12.1.1.2.6 Prepared Property

Determines whether a query is prepared for execution or not.

Class

[TMemDataSet](#)

Syntax

```
property Prepared: boolean;
```

Remarks

Determines whether a query is prepared for execution or not.

See Also

- [Prepare](#)

5.12.1.1.2.7 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](#)

5.12.1.1.2.8 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](#)

5.12.1.1.2.9 UpdatesPending Property

Used to check the status of the cached updates buffer.

Class

[TMemDataSet](#)

Syntax

```
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](#)

5.12.1.1.3 Methods

Methods of the **TMemDataSet** class.

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

Public

Name	Description
ApplyRange	Applies a range to the dataset.
ApplyUpdates	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange	Removes any ranges currently in effect for a dataset.
CancelUpdates	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates	Clears the cached updates buffer.
DeferredPost	Makes permanent changes to the database server.
EditRangeEnd	Enables changing the ending value for an existing range.
EditRangeStart	Enables changing the starting value for an existing range.
GetBlob	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Locate	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare	Allocates resources and creates field components for a dataset.
RestoreUpdates	Marks all records in the cache of updates as unapplied.
RevertRecord	Cancels changes made to the current record when cached updates are enabled.
SaveToXML	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange	Sets the starting and ending values of a range, and applies it.
SetRangeEnd	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus	Indicates the current update status for the dataset when cached updates are enabled.

See Also

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

5.12.1.1.3.1 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](#)

5.12.1.1.3.2 ApplyUpdates Method

Writes dataset's pending cached updates to a database.

Class

[TMemDataSet](#)

Overload List

Name	Description
ApplyUpdates	Writes dataset's pending cached updates to a database.
ApplyUpdates(const UpdateRecKinds: TUpdateRecKinds)	Writes dataset's pending cached updates of specified records to a database.

Writes dataset's pending cached updates to a database.

Class

[TMemDataSet](#)

Syntax

```
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:

```
procedure ApplyButtonClick(Sender: TObject);  
begin  
  with MyQuery do  
    begin  
      Session.StartTransaction;  
      try  
        ... <Modify data>  
        ApplyUpdates; <try to write the updates to the database>  
        Session.Commit; <on success, commit the changes>  
      except  
        RestoreUpdates; <restore update result for applied records>  
        Session.Rollback; <on failure, undo the changes>  
        raise; <raise the exception to prevent a call to CommitUpdates!>  
      end;  
      CommitUpdates; <on success, clear the cache>  
    end;  
end;
```

See Also

- [TMemDataSet.CachedUpdates](#)
- [TMemDataSet.CancelUpdates](#)
- [TMemDataSet.CommitUpdates](#)
- [TMemDataSet.UpdateStatus](#)

Writes dataset's pending cached updates of specified records to a database.

Class

[TMemDataSet](#)

Syntax

```
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.

5.12.1.1.3.3 CancelRange Method

Removes any ranges currently in effect for a dataset.

Class

[TMemDataSet](#)

Syntax

```
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](#)

5.12.1.1.3.4 CancelUpdates Method

Clears all pending cached updates from cache and restores dataset in its prior state.

Class

[TMemDataSet](#)

Syntax

```
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](#)

5.12.1.1.3.5 CommitUpdates Method

Clears the cached updates buffer.

Class

[TMemDataSet](#)

Syntax

```
procedure CommitUpdates;
```

Remarks

Call the CommitUpdates method to clear the cached updates buffer after both a successful call to ApplyUpdates and a database component's Commit method. Clearing the cache after applying updates ensures that the cache is empty except for records that could not be processed and were skipped by the OnUpdateRecord or OnUpdateError event handlers. An application can attempt to modify the records still in cache.

CommitUpdates also checks whether there are pending updates in 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](#)

5.12.1.1.3.6 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.

5.12.1.1.3.7 `EditRangeEnd` Method

Enables changing the ending value for an existing range.

Class

[TMemDataSet](#)

Syntax

```
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](#)

5.12.1.1.3.8 `EditRangeStart` Method

Enables changing the starting value for an existing range.

Class

[TMemDataSet](#)

Syntax

```
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](#)

5.12.1.1.3.9 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

Name	Description
GetBlob(Field: TField)	Retrieves <code>TBlob</code> object for a field or current record when the field itself is known.
GetBlob(const FieldName: string)	Retrieves <code>TBlob</code> object for a field or current record when its name is known.

Retrieves `TBlob` object for a field or current record when the field itself is known.

Class

[TMemDataSet](#)

Syntax

```
function GetBlob(Field: TField): TBlob; overload;
```

Parameters

Field

Holds an existing TField object.

Return Value

TBlob object that was retrieved.

Remarks

Call the GetBlob method to retrieve TBlob object for a field or current record when only its name or the field itself is known. FieldName is the name of an existing field. The field should have MEMO or BLOB type.

Retrieves TBlob object for a field or current record when its name is known.

Class

[TMemDataSet](#)

Syntax

```
function GetBlob(const FieldName: string): TBlob; overload;
```

Parameters*FieldName*

Holds the name of an existing field.

Return Value

TBlob object that was retrieved.

Example

```
PgQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
```

See Also

- [TBlob](#)

5.12.1.1.3.10 Locate Method

Searches a dataset for a specific record and positions the cursor on it.

Class

[TMemDataSet](#)

Overload List

Name	Description
------	-------------

Locate(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateOptions)	Searches a dataset by the specified fields for a specific record and positions cursor on it.
Locate(const KeyFields: string; const KeyValues: variant; Options: TLocateOptions)	Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

Searches a dataset by the specified fields for a specific record and positions cursor on it.

Class

[TMemDataSet](#)

Syntax

```
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:

```
with CustTable do
    Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P',
        '408-431-1000']), [loPartialKey]);
```

See Also

- [TMemDataSet.IndexFieldNames](#)
- [TMemDataSet.LocateEx](#)

5.12.1.1.3.11 LocateEx Method

Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Class

[TMemDataSet](#)

Overload List

Name	Description
LocateEx(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateExOptions)	Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.
LocateEx(const KeyFields: string; const KeyValues: variant; Options: TLocateExOptions)	Excludes features that don't need to be included to the TMemDataSet.Locate 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.

Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet by the specified field names.

Class

[TMemDataSet](#)

Syntax

```
function LocateEx(const KeyFields: string; const KeyValues:  
variant; Options: TLocateExOptions): boolean; overload;
```

Parameters

KeyFields

Holds the fields to search in.

KeyValues

Holds the values of the fields to search for.

Options

Holds additional search parameters which will be used by the LocateEx method.

Return Value

True, if a matching record was found. Otherwise returns False.

Remarks

Call the LocateEx method when you need some features not to be included to the [TMemDataSet.Locate](#) method of TDataSet.

LocateEx returns True if it finds a matching record, and makes that record the current one. Otherwise LocateEx returns False.

The LocateEx function works faster when dataset is locally sorted on the KeyFields fields.

Local dataset sorting can be set with the [TMemDataSet.IndexFieldNames](#) property.

Note: Please add the MemData unit to the "uses" list to use the TLocalExOption enumeration.

See Also

- [TMemDataSet.IndexFieldNames](#)
- [TMemDataSet.Locate](#)

5.12.1.1.3.12 Prepare Method

Allocates resources and creates field components for a dataset.

Class

[TMemDataSet](#)

Syntax

```
procedure Prepare; virtual;
```

Remarks

Call the Prepare method to allocate resources and create field components for a dataset. To learn whether dataset is prepared or not use the Prepared property.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- [Prepared](#)
- [UnPrepare](#)

5.12.1.1.3.13 RestoreUpdates Method

Marks all records in the cache of updates as unapplied.

Class

[TMemDataSet](#)

Syntax

```
procedure RestoreUpdates;
```

Remarks

Call the RestoreUpdates method to return the cache of updates to its state before calling ApplyUpdates. RestoreUpdates marks all records in the cache of updates as unapplied. It is useful when ApplyUpdates fails.

See Also

- [CachedUpdates](#)
- [TMemDataSet.ApplyUpdates](#)
- [CancelUpdates](#)
- [UpdateStatus](#)

5.12.1.1.3.14 RevertRecord Method

Cancels changes made to the current record when cached updates are enabled.

Class

[TMemDataSet](#)

Syntax

procedure RevertRecord;

Remarks

Call the RevertRecord method to undo changes made to the current record when cached updates are enabled.

See Also

- [CachedUpdates](#)
- [CancelUpdates](#)

5.12.1.1.3.15 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

Name	Description
SaveToXML(Destination: TStream)	Saves the current dataset data to a stream in the XML format compatible with ADO format.
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

- `M:Devart.Dac.TVirtualTable.LoadFromFile(System.String,System.Boolean)`
- `M:Devart.Dac.TVirtualTable.LoadFromStream(Borland.Vcl.TStream,System.Boolean)`

Saves the current dataset data to a file in the XML format compatible with ADO format.

Class

[TMemDataSet](#)

Syntax

```
procedure SaveToXML(const FileName: string); overload;
```

Parameters

FileName

Holds the name of a destination file.

5.12.1.1.3.16 SetRange Method

Sets the starting and ending values of a range, and applies it.

Class

[TMemDataSet](#)

Syntax

```
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 set to NULL.

See Also

- [ApplyRange](#)
- [CancelRange](#)
- [EditRangeEnd](#)
- [EditRangeStart](#)
- [IndexFieldNames](#)
- [KeyExclusive](#)
- [SetRangeEnd](#)
- [SetRangeStart](#)

5.12.1.1.3.17 SetRangeEnd Method

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

Class

[TMemDataSet](#)

Syntax

```
procedure SetRangeEnd;
```

Remarks

Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL.

Subsequent field assignments made with FieldByName specify the actual set of ending values for a range.

After assigning end-range values, call [ApplyRange](#) to activate the modified range.

See Also

- [ApplyRange](#)
- [CancelRange](#)
- [EditRangeStart](#)
- [IndexFieldNames](#)
- [SetRange](#)
- [SetRangeStart](#)

5.12.1.1.3.18 SetRangeStart Method

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

Class

[TMemDataSet](#)

Syntax

```
procedure SetRangeStart;
```

Remarks

Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL.

Subsequent field assignments to FieldByName specify the actual set of starting values for a range.

After assigning start-range values, call [ApplyRange](#) to activate the modified range.

See Also

- [ApplyRange](#)
- [CancelRange](#)
- [EditRangeStart](#)
- [IndexFieldNames](#)
- [SetRange](#)
- [SetRangeEnd](#)

5.12.1.1.3.19 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class

[TMemDataSet](#)

Syntax

```
procedure UnPrepare; virtual;
```

Remarks

Call the UnPrepare method to free the resources allocated for a previously prepared query on the server and client sides.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- [Prepare](#)

5.12.1.1.3.20 UpdateResult Method

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Class

[TMemDataSet](#)

Syntax

```
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](#)

5.12.1.1.3.21 UpdateStatus Method

Indicates the current update status for the dataset when cached updates are enabled.

Class

[TMemDataSet](#)

Syntax

```
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](#)

5.12.1.1.4 Events

Events of the **TMemDataSet** class.

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

Public

Name	Description
OnUpdateError	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord	Occurs when a single update component can not handle the updates.

See Also

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

5.12.1.1.4.1 OnUpdateError Event

Occurs when an exception is generated while cached updates are applied to a database.

Class

[TMemDataSet](#)

Syntax

```
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](#)

5.12.1.1.4.2 OnUpdateRecord Event

Occurs when a single update component can not handle the updates.

Class

[TMemDataSet](#)

Syntax

```
property OnUpdateRecord: TUpdateRecordEvent;
```

Remarks

Write the OnUpdateRecord event handler to process updates that cannot be handled by a single update component, such as implementation of cascading updates, insertions, or deletions. This handler is also useful for applications that require additional control over parameter substitution in update components.

UpdateKind describes the type of update to perform.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful, it should set UpdateAction to uaApplied before exiting.

See Also

- [CachedUpdates](#)

5.12.2 Variables

Variables in the **MemDS** unit.

Variables

Name	Description
DoNotRaiseExcetionOnUaFail	An exception will be raised if the value of the UpdateAction parameter is uaFail.
SendDataSetChangeEventAfterOpen	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.

5.12.2.1 DoNotRaiseExcetionOnUaFail Variable

An exception will be raised if the value of the UpdateAction parameter is uaFail.

Unit

[MemDS](#)

Syntax

```
DoNotRaiseExcetionOnUaFail: boolean = False;
```

Remarks

Starting with PgDAC , if the [OnUpdateRecord](#) event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged.

To restore the old behaviour, set DoNotRaiseExcetionOnUaFail to True.

5.12.2.2 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

```
sendDataSetChangeEventAfterOpen: boolean = True;
```

Remarks

Starting with PgDAC , the DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled.

To disable sending this event, change the value of this variable to False.

5.13 PgAccess

This unit contains main components of PgDAC.

Classes

Name	Description
------	-------------

<u>TCustomPgDataSet</u>	A base class defining functionality for the classes derived from it.
<u>TCustomPgStoredProc</u>	A component for accessing and executing stored procedures and functions.
<u>TCustomPgTable</u>	A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.
<u>TCustomPgTimeStampField</u>	A base class defining functionality for the classes derived from it.
<u>TPgConnection</u>	Represents an open connection to a PostgreSQL database.
<u>TPgConnectionOptions</u>	This class allows setting up the behaviour of the TPgConnection class.
<u>TPgConnectionSSLOptions</u>	This class allows setting up the behaviour of the TPgConnection class.
<u>TPgCursorField</u>	A class providing access to the PostgreSQL cursor fields.
<u>TPgDataSetOptions</u>	This class allows setting up the behaviour of the TCustomPgDataSet class.
<u>TPgDataSource</u>	TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.
<u>TPgDateField</u>	A class providing access to the PostgreSQL date fields.
<u>TPgEncryptor</u>	The class that performs encrypting and decrypting of data.
<u>TPgGeometricField</u>	A class providing access to the PostgreSQL geometric fields.
<u>TPgIntervalField</u>	A class providing access to the PostgreSQL interval fields.
<u>TPgLargeObject</u>	A class providing support of PostgreSQL large objects.

TPgLargeObjectField	A class providing access to the PostgreSQL large object fields.
TPgMetaData	A component for obtaining metainformation about database objects from the server.
TPgParam	A class that is used to set the values of individual parameters passed with queries or stored procedures.
TPgParams	Used to control TPgParam objects.
TPgQuery	A component for executing queries and operating record sets. It also provides flexible way to update data.
TPgSQL	A component for executing SQL statements and calling stored procedures on the database server.
TPgStoredProc	A component for accessing and executing stored procedures and functions.
TPgTable	A component for retrieving and updating data in a single table without writing SQL statements.
TPgTimeField	A class providing access to the PostgreSQL time fields.
TPgTimeStampField	A class providing access to the PostgreSQL timestamp fields.
TPgTransaction	A component for managing transactions in an application.
TPgUpdateSQL	Lets you tune update operations for the DataSet component.

Types

Name	Description
TPgNoticeEvent	This type is used for the TPgConnection.OnNotice event.
TPgNotificationEvent	This type is used for the TPgConnection.OnNotification event.

Enumerations

Name	Description
TPgIsolationLevel	Specifies the way the transactions containing database modifications are handled.

Constants

Name	Description
PgDACVersion	The version of PostgreSQL Data Access Components.

5.13.1 Classes

Classes in the **PgAccess** unit.

Classes

Name	Description
TCustomPgDataSet	A base class defining functionality for the classes derived from it.
TCustomPgStoredProc	A component for accessing and executing stored procedures and functions.
TCustomPgTable	A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.
TCustomPgTimeStampField	A base class defining functionality for the classes derived from it.
TPgConnection	Represents an open connection to a PostgreSQL database.
TPgConnectionOptions	This class allows setting up the behaviour of the TPgConnection class.
TPgConnectionSSLOptions	This class allows setting up the behaviour of the TPgConnection class.
TPgCursorField	A class providing access to the PostgreSQL cursor fields.

<u>TPgDataSetOptions</u>	This class allows setting up the behaviour of the TCustomPgDataSet class.
<u>TPgDataSource</u>	TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.
<u>TPgDateField</u>	A class providing access to the PostgreSQL date fields.
<u>TPgEncryptor</u>	The class that performs encrypting and decrypting of data.
<u>TPgGeometricField</u>	A class providing access to the PostgreSQL geometric fields.
<u>TPgIntervalField</u>	A class providing access to the PostgreSQL interval fields.
<u>TPgLargeObject</u>	A class providing support of PostgreSQL large objects.
<u>TPgLargeObjectField</u>	A class providing access to the PostgreSQL large object fields.
<u>TPgMetaData</u>	A component for obtaining metainformation about database objects from the server.
<u>TPgParam</u>	A class that is used to set the values of individual parameters passed with queries or stored procedures.
<u>TPgParams</u>	Used to control TPgParam objects.
<u>TPgQuery</u>	A component for executing queries and operating record sets. It also provides flexible way to update data.
<u>TPgSQL</u>	A component for executing SQL statements and calling stored procedures on the database server.
<u>TPgStoredProc</u>	A component for accessing and executing stored procedures and functions.
<u>TPgTable</u>	A component for retrieving and updating data in a single table without writing SQL statements.

TPgTimeField	A class providing access to the PostgreSQL time fields.
TPgTimeStampField	A class providing access to the PostgreSQL timestamp fields.
TPgTransaction	A component for managing transactions in an application.
TPgUpdateSQL	Lets you tune update operations for the DataSet component.

5.13.1.1 TCustomPgDataSet Class

A base class defining functionality for the classes derived from it.

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

Unit

[PgAccess](#)

Syntax

```
TCustomPgDataSet = class (TCustomDADataset);
```

Remarks

TCustomPgDataSet is a base dataset component that defines functionality for classes derived from it. Applications never use TCustomPgDataSet objects directly. Instead they use descendants of TCustomPgDataSet, such as TPgQuery and TPgTable that inherit its dataset-related properties and methods.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

TCustomPgDataSet

5.13.1.1.1 Members

[TCustomPgDataSet](#) class overview.

Properties

Name	Description
------	-------------

<u>BaseSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>Conditions</u> (inherited from <u>TCustomDADataset</u>)	Used to add WHERE conditions to a query
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>Cursor</u>	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
<u>DataTypeMap</u> (inherited from <u>TCustomDADataset</u>)	Used to set data type mapping rules
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u>	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchAll</u>	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.

<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u>	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u>	Returns OID of the record inserted by the last query for table with OIDs.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
MasterSource (inherited from TCustomDADataset)	Used to specify the data source component which binds current dataset to the master one.
Options	Used to specify the behaviour of TCustomPgDataSet object.
ParamCheck (inherited from TCustomDADataset)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from TCustomDADataset)	Used to indicate how many parameters are there in the Params property.
Params	Contains parameters for a query's SQL statement.
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.
ReadOnly (inherited from TCustomDADataset)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions (inherited from TCustomDADataset)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from TCustomDADataset)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SequenceMode	Used to specify the methods used internally to generate a sequenced field.
SQL (inherited from TCustomDADataset)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from	Used to specify a SQL statement that will be used when applying a deletion

<u>TCustomDADataset</u>)	to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLLock</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to perform a record lock.
<u>SQLRecCount</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.
<u>SQLUpdate</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying an update to a dataset.
<u>UniDirectional</u> (inherited from <u>TCustomDADataset</u>)	Used if an application does not need bidirectional access to records in the result set.
<u>UpdateObject</u>	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
<u>UpdateRecordTypes</u> (inherited from <u>TMemDataSet</u>)	Used to indicate the update status for the current record when cached updates are enabled.
<u>UpdatesPending</u> (inherited from <u>TMemDataSet</u>)	Used to check the status of the cached updates buffer.

Methods

Name	Description
------	-------------

AddWhere (inherited from TCustomDADataset)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from TMemDataSet)	Applies a range to the dataset.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from TCustomDADataset)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
CreateBlobStream (inherited from TCustomDADataset)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall	Generates the stored procedure call.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
DeleteWhere (inherited from TCustomDADataset)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from TMemDataSet)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from TMemDataSet)	Enables changing the starting value for an existing range.

<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Overloaded. Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Description is not available at the moment.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u>	Searches for and returns a parameter with the specified name.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetFieldObject</u> (inherited from	Returns a multireference shared object from field.

<u>TCustomDADataset</u>)	
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetKeyFieldNames</u> (inherited from <u>TCustomDADataset</u>)	Provides a list of available key field names.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetPgCursor</u>	Retrieves a TPgCursor object for a field with known name.
<u>GetPgDate</u>	Retrieves a TPgDate object for a field with known name.
<u>GetPgInterval</u>	Retrieves a TPgInterval object for a field with known name.
<u>GetPgLargeObject</u>	Retrieves a TPgLargeObject object for a field with known name.
<u>GetPgRow</u>	Retrieves a TPgRow object for a field with known name.
<u>GetPgTime</u>	Retrieves a TPgTime object for a field with known name.
<u>GetPgTimeStamp</u>	Retrieves a TPgTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.

LocateEx (inherited from TMemDataSet)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from TCustomDADataset)	Locks the current record.
MacroByName (inherited from TCustomDADataset)	Finds a Macro with the name passed in Name.
OpenNext	Opens the next REFCURSOR for stored procedure that returns more than one cursor.
ParamByName	Searches for and returns a parameter with the specified name.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
RefreshRecord (inherited from TCustomDADataset)	Actualizes field values for the current record.
RestoreSQL (inherited from TCustomDADataset)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from TMemDataSet)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from TCustomDADataset)	Saves the SQL property value to BaseSQL.
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 (inherited from TCustomDADataset)	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 (inherited from TCustomDADataset)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from TCustomDADataset)	Releases a record lock.
UnPrepare (inherited from TMemDataSet)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from TMemDataSet)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from TMemDataSet)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from TCustomDADataset)	Occurs after a component has executed a query to database.
AfterFetch (inherited from TCustomDADataset)	Occurs after dataset finishes fetching data from server.

AfterUpdateExecute (inherited from TCustomDADataset)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from TCustomDADataset)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from TCustomDADataset)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from TMemDataSet)	Occurs when a single update component can not handle the updates.

5.13.1.1.2 Properties

Properties of the **TCustomPgDataSet** class.

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

Public

Name	Description
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
Conditions (inherited from TCustomDADataset)	Used to add WHERE conditions to a query
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.

<u>TCustomDADataset</u>)	
<u>Cursor</u>	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
<u>DataTypeMap</u> (inherited from <u>TCustomDADataset</u>)	Used to set data type mapping rules
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u>	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchAll</u>	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.

<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u>	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u>	Returns OID of the record inserted by the last query for table with OIDs.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>Options</u>	Used to specify the behaviour of TCustomPgDataSet object.
<u>ParamCheck</u> (inherited from	Used to specify whether parameters for the Params property are generated automatically after the

<u>TCustomDADataset</u>)	SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u>	Contains parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>SequenceMode</u>	Used to specify the methods used internally to generate a sequenced field.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLLock</u> (inherited from <u>TCustomDADataset</u>)	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	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
UpdateRecordTypes (inherited from TMemDataSet)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from TMemDataSet)	Used to check the status of the cached updates buffer.

See Also

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

5.13.1.1.2.1 Cursor Property

Used to fetch data from the REFCURSOR parameter or REFCURSOR field.

Class

[TCustomPgDataSet](#)

Syntax

```
property Cursor: TPgRefCursor;
```

Remarks

Use the Cursor property to fetch data from the REFCURSOR parameter or REFCURSOR field. You can assign the value of TPgParam.AsCursor or TPgCursorField.AsCursor to the Cursor property. After assigning you can open the dataset once.

Example

```
PgQuery1.Cursor := PgSQL1.ParamByName('Cur').AsCursor;  
PgQuery1.Open;
```

See Also

- [TPgRefCursor](#)
- P:Devart.PgDac.TPgParam.AsCursor

5.13.1.1.2.2 DMLRefresh Property

Used to refresh record by RETURNING clause when insert or update is performed.

Class

[TCustomPgDataSet](#)

Syntax

```
property DMLRefresh: boolean;
```

Remarks

Use the DMLRefresh property to refresh record by RETURNING clause when insert or update is performed.

The default value is False.

5.13.1.1.2.3 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

[TCustomPgDataSet](#)

Syntax

```
property FetchAll: boolean default True;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

Opening a dataset in FetchAll = False mode requires an active transaction. When the FetchAll property is False, the first call to [TMemDataSet.Locate](#) and [TMemDataSet.LocateEx](#) methods may take a lot of time to retrieve additional records to the client side.

See Also

- [TCustomDADataset.FetchRows](#)

5.13.1.1.2.4 KeySequence Property

Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.

Class

[TCustomPgDataSet](#)

Syntax

```
property KeySequence: string;
```

Remarks

Use the KeySequence property to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.

Note: KeySequence is used by TCustomPgDataSet only if the KeyFields property is assigned.

See Also

- [TCustomDADataset.KeyFields](#)
- [SequenceMode](#)

5.13.1.1.2.5 LastInsertOID Property

Returns OID of the record inserted by the last query for table with OIDs.

Class

[TCustomPgDataSet](#)

Syntax

property LastInsertOID: Int64;

Remarks

Use the LastInsertOID property to get OID of the record inserted by the last query for table with OIDs.

5.13.1.1.2.6 Options Property

Used to specify the behaviour of TCustomPgDataSet object.

Class

[TCustomPgDataSet](#)

Syntax

property Options: [TPgDataSetOptions](#);

Remarks

Set the properties of Options to specify the behaviour of a TCustomPgDataSet object. Descriptions of all options are in the table below.

Option Name	Description
AutoDeleteBlob	Used to delete large objects from the database automatically when a record is deleted from the dataset.
CacheBlobs	Used to allocate local memory buffer to hold a copy of the large object content.
CursorWithHold	Used to open query in the FetchAll=False mode without transaction.
DeferredBlobRead	Used to fetch values of large objects when they are explicitly requested.
ExtendedFieldsInfo	Used to perform an additional query to get information about returned fields and the tables they belong to.
OIDAsInt	Used to read OID fields as integer values and map these fields to TIntegerField.
PrepareUpdateSQL	Used to automatically prepare update queries before execution.
UnknownAsString	Used to map fields of unknown data types to TStringField (TWideStringField).

UnpreparedExecute	Used to apply simple executing to a SQL statement.
UseParamTypes	Used to disable automatic detection of the parameters data types.

5.13.1.1.2.7 Params Property

Contains parameters for a query's SQL statement.

Class

[TCustomPgDataSet](#)

Syntax

```
property Params: TPgParams stored False;
```

Remarks

Contains parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

5.13.1.1.2.8 SequenceMode Property

Used to specify the methods used internally to generate a sequenced field.

Class

[TCustomPgDataSet](#)

Syntax

```
property SequenceMode: TSequenceMode default smPost;
```

Remarks

Set the SequenceMode property to specify which method is used internally to generate a sequenced field.

See Also

- [TCustomDADataset.KeyFields](#)
- [KeySequence](#)

5.13.1.1.2.9 UpdateObject Property

Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.

Class

[TCustomPgDataSet](#)

Syntax

```
property UpdateObject: TPgUpdateSQL;
```

Remarks

The UpdateObject property points to an update object component which provides SQL statements that perform updates of read-only datasets when cached updates are enabled.

5.13.1.1.3 Methods

Methods of the **TCustomPgDataSet** class.

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

Public

Name	Description
AddWhere (inherited from TCustomDADataset)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from TMemDataSet)	Applies a range to the dataset.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from TCustomDADataset)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.

<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.
<u>CreateBlobStream</u> (inherited from <u>TCustomDADataset</u>)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
<u>CreateProcCall</u>	Generates the stored procedure call.
<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>EditRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Enables changing the ending value for an existing range.
<u>EditRangeStart</u> (inherited from <u>TMemDataSet</u>)	Enables changing the starting value for an existing range.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Overloaded. Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.

<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Description is not available at the moment.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u>	Searches for and returns a parameter with the specified name.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetKeyFieldNames</u> (inherited from <u>TCustomDADataset</u>)	Provides a list of available key field names.
<u>GetOrderBy</u> (inherited from	Retrieves an ORDER BY clause from a SQL statement.

<u>TCustomDADataset</u>)	
<u>GetPgCursor</u>	Retrieves a TPgCursor object for a field with known name.
<u>GetPgDate</u>	Retrieves a TPgDate object for a field with known name.
<u>GetPgInterval</u>	Retrieves a TPgInterval object for a field with known name.
<u>GetPgLargeObject</u>	Retrieves a TPgLargeObject object for a field with known name.
<u>GetPgRow</u>	Retrieves a TPgRow object for a field with known name.
<u>GetPgTime</u>	Retrieves a TPgTime object for a field with known name.
<u>GetPgTimeStamp</u>	Retrieves a TPgTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADataset</u>)	Locks the current record.
<u>MacroByName</u> (inherited from <u>TCustomDADataset</u>)	Finds a Macro with the name passed in Name.
<u>OpenNext</u>	Opens the next REFCURSOR for stored procedure that returns more than one cursor.
<u>ParamByName</u>	Searches for and returns a parameter with the specified name.

Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
RefreshRecord (inherited from TCustomDADataset)	Actualizes field values for the current record.
RestoreSQL (inherited from TCustomDADataset)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from TMemDataSet)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from TCustomDADataset)	Saves the SQL property value to BaseSQL.
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 (inherited from TCustomDADataset)	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 (inherited from TCustomDADataset)	Determines if the SQL property value was saved to the BaseSQL property.

<u>TCustomDADataset</u>)	
<u>UnLock</u> (inherited from <u>TCustomDADataset</u>)	Releases a record lock.
<u>UnPrepare</u> (inherited from <u>TMemDataSet</u>)	Frees the resources allocated for a previously prepared query on the server and client sides.
<u>UpdateResult</u> (inherited from <u>TMemDataSet</u>)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
<u>UpdateStatus</u> (inherited from <u>TMemDataSet</u>)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

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

5.13.1.1.3.1 CreateProcCall Method

Generates the stored procedure call.

Class

[TCustomPgDataSet](#)

Syntax

```
procedure CreateProcCall(Name: string; Overload: integer = 0);
```

Parameters

Name

the name of the stored procedure.

Overload

the number of the overloaded procedure.

Remarks

Call the CreateProcCall method to assign SQL statement that calls the stored procedure to the SQL property and fill the Params property. The Overload parameter contains the number of the overloaded procedure. Retrieves information about the procedure parameters from PostgreSQL server. After calling CreateProcCall you can execute stored procedure by the Execute method.

5.13.1.1.3.2 FindParam Method

Searches for and returns a parameter with the specified name.

Class

[TCustomPgDataSet](#)

Syntax

```
function FindParam(const value: string): TPgParam;
```

Parameters

Value

Holds the stored procedure name.

Return Value

the parameter, if a match was found. Nil otherwise.

Remarks

Call the FindParam method to find a parameter with the name passed in the Name argument. If a match was found, FindParam returns the parameter. Otherwise, it returns nil.

See Also

- [TPgParam](#)
- [TPgSQL.ParamByName](#)

5.13.1.1.3.3 GetPgCursor Method

Retrieves a TPgCursor object for a field with known name.

Class

[TCustomPgDataSet](#)

Syntax

```
function GetPgCursor(const FieldName: string): TPgRefCursor;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TPgCursor object for a field with known name.

Remarks

Call the `GetPgCursor` method to retrieve a `TPgCursor` object for a field when only its name is known. `FieldName` is the name of an existing field.

5.13.1.1.3.4 GetPgDate Method

Retrieves a `TPgDate` object for a field with known name.

Class

[TCustomPgDataSet](#)

Syntax

```
function GetPgDate(const FieldName: string): TPgDate;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a `TPgDate` object for a field with known name.

Remarks

Call the `GetPgDate` method to retrieve a `TPgDate` object for a field when only its name is known. `FieldName` is the name of an existing field.

5.13.1.1.3.5 GetPgInterval Method

Retrieves a `TPgInterval` object for a field with known name.

Class

[TCustomPgDataSet](#)

Syntax

```
function GetPgInterval(const FieldName: string): TPgInterval;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a `TPgInterval` object for a field with known name.

Remarks

Call the `GetPgInterval` method to retrieve a `TPgInterval` object for a field when only its name is

known. `FieldName` is the name of an existing field.

See Also

- [TPgInterval](#)
- [TPgParam.AsPgInterval](#)

5.13.1.1.3.6 GetPgLargeObject Method

Retrieves a `TPgLargeObject` object for a field with known name.

Class

[TCustomPgDataSet](#)

Syntax

```
function GetPgLargeObject(const FieldName: string):  
TPgLargeObject;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a `TPgLargeObject` object for a field with known name.

Remarks

Call the `GetPgLargeObject` method to retrieve a `TPgLargeObject` object for a field when only its name is known. `FieldName` is the name of an existing field.

5.13.1.1.3.7 GetPgRow Method

Retrieves a `TPgRow` object for a field with known name.

Class

[TCustomPgDataSet](#)

Syntax

```
function GetPgRow(const FieldName: string): TPgRow;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TPgRow object for a field with known name.

Remarks

Call the GetPgRow method to retrieve a TPgRow object for a field when only its name is known. FieldName is the name of an existing field.

5.13.1.1.3.8 GetPgTime Method

Retrieves a TPgTime object for a field with known name.

Class

[TCustomPgDataSet](#)

Syntax

```
function GetPgTime(const FieldName: string): TPgTime;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TPgTime object for a field with known name.

Remarks

Call the GetPgTime method to retrieve a TPgTime object for a field when only its name is known. FieldName is the name of an existing field.

5.13.1.1.3.9 GetPgTimeStamp Method

Retrieves a TPgTimeStamp object for a field with known name.

Class

[TCustomPgDataSet](#)

Syntax

```
function GetPgTimeStamp(const FieldName: string): TPgTimeStamp;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TPgTimeStamp object for a field with known name.

Remarks

Call the GetPgTimeStamp method to retrieve a TPgTimeStamp object for a field when only its name is known. FieldName is the name of an existing field.

5.13.1.1.3.10 OpenNext Method

Opens the next REFCURSOR for stored procedure that returns more than one cursor.

Class

[TCustomPgDataSet](#)

Syntax

```
function openNext: boolean;
```

Return Value

True, if the next cursor is opened. False, if there are no more cursors.

Remarks

Call the OpenNext method to open next REFCURSOR for stored procedure that returns more than one cursor. When you execute a stored procedure that has REFCURSOR parameters, PgDAC opens a cursor from the first parameter automatically. The OpenNext function closes current cursor and opens a cursor from the next REFCURSOR parameter.

5.13.1.1.3.11 ParamByName Method

Searches for and returns a parameter with the specified name.

Class

[TCustomPgDataSet](#)

Syntax

```
function ParamByName(const value: string): TPgParam;
```

Parameters

Value

Holds the parameter name.

Return Value

the parameter, if a match was found. Otherwise an exception is raised.

Remarks

Call the ParamByName method to find a parameter with the name passed in the Name argument.

If a match is found, ParamByName returns the parameter. Otherwise, an exception is raised.

See Also

- [TPgParam](#)
- [TPgSQL.FindParam](#)

5.13.1.2 TCustomPgStoredProc Class

A component for accessing and executing stored procedures and functions.

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

Unit

[PgAccess](#)

Syntax

```
TCustomPgStoredProc = class(TCustomPgDataSet);
```

Remarks

Use TPgStoredProc to access stored procedures on the database server.

You need only to define the StoredProcName property, and the SQL statement to call the stored procedure will be generated automatically.

Use the Execute method at runtime to generate request that instructs server to execute procedure and PrepareSQL to describe parameters at run time

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

[TCustomPgDataSet](#)

TCustomPgStoredProc

5.13.1.2.1 Members

[TCustomPgStoredProc](#) class overview.

Properties

Name	Description
------	-------------

<u>BaseSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>Conditions</u> (inherited from <u>TCustomDADataset</u>)	Used to add WHERE conditions to a query
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>Cursor</u> (inherited from <u>TCustomPgDataSet</u>)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
<u>DataTypeMap</u> (inherited from <u>TCustomDADataset</u>)	Used to set data type mapping rules
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TCustomPgDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchAll</u> (inherited from <u>TCustomPgDataSet</u>)	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from	Used to define the number of rows to be transferred across the network at the same time.

<u>TCustomDADataset</u>)	
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u> (inherited from <u>TCustomPgDataSet</u>)	Returns OID of the record inserted by the last query for table with OIDs.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.

<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>Options</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the behaviour of TCustomPgDataSet object.
<u>Overload</u>	Used to specify the overloading number.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomPgDataSet</u>)	Contains parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

<u>SequenceMode</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLLock</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to perform a record lock.
<u>SQLRecCount</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.
<u>SQLUpdate</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying an update to a dataset.
<u>UniDirectional</u> (inherited from <u>TCustomDADataset</u>)	Used if an application does not need bidirectional access to records in the result set.
<u>UpdateObject</u> (inherited from <u>TCustomPgDataSet</u>)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
<u>UpdateRecordTypes</u> (inherited from <u>TMemDataSet</u>)	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

Name	Description
AddWhere (inherited from TCustomDADataset)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from TMemDataSet)	Applies a range to the dataset.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from TCustomDADataset)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
CreateBlobStream (inherited from TCustomDADataset)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from TCustomPgDataSet)	Generates the stored procedure call.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.

<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>EditRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Enables changing the ending value for an existing range.
<u>EditRangeStart</u> (inherited from <u>TMemDataSet</u>)	Enables changing the starting value for an existing range.
<u>ExecProc</u>	Executes a SQL statement on the server.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Overloaded. Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Description is not available at the moment.
<u>FindNearest</u> (inherited from	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the

<u>TCustomDADataset</u>)	values specified in the KeyValues parameter.
<u>FindParam</u> (inherited from <u>TCustomPgDataset</u>)	Searches for and returns a parameter with the specified name.
<u>GetBlob</u> (inherited from <u>TMemDataset</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetKeyFieldNames</u> (inherited from <u>TCustomDADataset</u>)	Provides a list of available key field names.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetPgCursor</u> (inherited from <u>TCustomPgDataset</u>)	Retrieves a TPgCursor object for a field with known name.
<u>GetPgDate</u> (inherited from <u>TCustomPgDataset</u>)	Retrieves a TPgDate object for a field with known name.
<u>GetPgInterval</u> (inherited from <u>TCustomPgDataset</u>)	Retrieves a TPgInterval object for a field with known name.

<u>GetPgLargeObject</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgLargeObject object for a field with known name.
<u>GetPgRow</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgRow object for a field with known name.
<u>GetPgTime</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTime object for a field with known name.
<u>GetPgTimeStamp</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADDataSet</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADDataSet</u>)	Locks the current record.
<u>MacroByName</u> (inherited from <u>TCustomDADDataSet</u>)	Finds a Macro with the name passed in Name.
<u>OpenNext</u> (inherited from <u>TCustomPgDataSet</u>)	Opens the next REFCURSOR for stored procedure that returns more than one cursor.
<u>ParamByName</u> (inherited from <u>TCustomPgDataSet</u>)	Searches for and returns a parameter with the specified name.
<u>Prepare</u> (inherited from <u>TCustomPgDataSet</u>)	Allocates, opens, and parses cursor for a query.

<u>TCustomDADataset</u>)	
<u>PrepareSQL</u>	Describes the parameters of a stored procedure.
<u>RefreshRecord</u> (inherited from <u>TCustomDADataset</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADataset</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>SaveSQL</u> (inherited from <u>TCustomDADataset</u>)	Saves the SQL property value to BaseSQL.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Builds an ORDER BY clause of a SELECT statement.
<u>SetRange</u> (inherited from <u>TMemDataSet</u>)	Sets the starting and ending values of a range, and applies it.
<u>SetRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
<u>SetRangeStart</u> (inherited from <u>TMemDataSet</u>)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
<u>SQLSaved</u> (inherited from	Determines if the <u>SQL</u> property value was saved to the <u>BaseSQL</u> property.

<u>TCustomDADataset</u>)	
<u>UnLock</u> (inherited from <u>TCustomDADataset</u>)	Releases a record lock.
<u>UnPrepare</u> (inherited from <u>TMemDataSet</u>)	Frees the resources allocated for a previously prepared query on the server and client sides.
<u>UpdateResult</u> (inherited from <u>TMemDataSet</u>)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
<u>UpdateStatus</u> (inherited from <u>TMemDataSet</u>)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
<u>AfterExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs after a component has executed a query to database.
<u>AfterFetch</u> (inherited from <u>TCustomDADataset</u>)	Occurs after dataset finishes fetching data from server.
<u>AfterUpdateExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs after executing insert, delete, update, lock and refresh operations.
<u>BeforeFetch</u> (inherited from <u>TCustomDADataset</u>)	Occurs before dataset is going to fetch block of records from the server.
<u>BeforeUpdateExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs before executing insert, delete, update, lock, and refresh operations.
<u>OnUpdateError</u> (inherited from <u>TMemDataSet</u>)	Occurs when an exception is generated while cached updates are applied to a database.
<u>OnUpdateRecord</u> (inherited from	Occurs when a single update component can not handle the

[TMemDataSet](#))

updates.

5.13.1.2.2 Properties

Properties of the **TCustomPgStoredProc** class.

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

Public

Name	Description
<u>BaseSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>Conditions</u> (inherited from <u>TCustomDADataset</u>)	Used to add WHERE conditions to a query
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>Cursor</u> (inherited from <u>TCustomPgDataSet</u>)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
<u>DataTypeMap</u> (inherited from <u>TCustomDADataset</u>)	Used to set data type mapping rules
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TCustomPgDataset</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchAll</u> (inherited from <u>TCustomPgDataset</u>)	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataset</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataset</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataset</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u> (inherited from <u>TCustomPgDataset</u>)	Returns OID of the record inserted by the last query for table with OIDs.

<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>Options</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the behaviour of TCustomPgDataSet object.
<u>Overload</u>	Used to specify the overloading number.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomPgDataSet</u>)	Contains parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.

<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>SequenceMode</u> (inherited from <u>TCustomPgDataset</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLLock</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to perform a record lock.
<u>SQLRecCount</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.
<u>SQLUpdate</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying an update to a dataset.

UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomPgDataSet)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
UpdateRecordTypes (inherited from TMemDataSet)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from TMemDataSet)	Used to check the status of the cached updates buffer.

See Also

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

5.13.1.2.2.1 Overload Property

Used to specify the overloading number.

Class

[TCustomPgStoredProc](#)

Syntax

```
property overload: integer default 0;
```

Remarks

Set the Overload property to specify the overloading number in case the stored procedure is overloaded.

5.13.1.2.3 Methods

Methods of the **TCustomPgStoredProc** class.

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

Public

Name	Description
<u>AddWhere</u> (inherited from <u>TCustomDADataset</u>)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
<u>ApplyRange</u> (inherited from <u>TMemDataSet</u>)	Applies a range to the dataset.
<u>ApplyUpdates</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Writes dataset's pending cached updates to a database.
<u>BreakExec</u> (inherited from <u>TCustomDADataset</u>)	Breaks execution of the SQL statement on the server.
<u>CancelRange</u> (inherited from <u>TMemDataSet</u>)	Removes any ranges currently in effect for a dataset.
<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.
<u>CreateBlobStream</u> (inherited from <u>TCustomDADataset</u>)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
<u>CreateProcCall</u> (inherited from <u>TCustomPgDataSet</u>)	Generates the stored procedure call.
<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>EditRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Enables changing the ending value for an existing range.

<u>EditRangeStart</u> (inherited from <u>TMemDataSet</u>)	Enables changing the starting value for an existing range.
<u>ExecProc</u>	Executes a SQL statement on the server.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Overloaded. Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Description is not available at the moment.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u> (inherited from <u>TCustomPgDataSet</u>)	Searches for and returns a parameter with the specified name.

<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetKeyFieldNames</u> (inherited from <u>TCustomDADataset</u>)	Provides a list of available key field names.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetPgCursor</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgCursor object for a field with known name.
<u>GetPgDate</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgDate object for a field with known name.
<u>GetPgInterval</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgInterval object for a field with known name.
<u>GetPgLargeObject</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgLargeObject object for a field with known name.
<u>GetPgRow</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgRow object for a field with known name.

<u>TCustomPgDataSet</u>)	
<u>GetPgTime</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTime object for a field with known name.
<u>GetPgTimeStamp</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADataset</u>)	Locks the current record.
<u>MacroByName</u> (inherited from <u>TCustomDADataset</u>)	Finds a Macro with the name passed in Name.
<u>OpenNext</u> (inherited from <u>TCustomPgDataSet</u>)	Opens the next REFCURSOR for stored procedure that returns more than one cursor.
<u>ParamByName</u> (inherited from <u>TCustomPgDataSet</u>)	Searches for and returns a parameter with the specified name.
<u>Prepare</u> (inherited from <u>TCustomDADataset</u>)	Allocates, opens, and parses cursor for a query.
<u>PrepareSQL</u>	Describes the parameters of a stored procedure.
<u>RefreshRecord</u> (inherited from	Actualizes field values for the current record.

<u>TCustomDADataset</u>)	
<u>RestoreSQL</u> (inherited from <u>TCustomDADataset</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>SaveSQL</u> (inherited from <u>TCustomDADataset</u>)	Saves the SQL property value to BaseSQL.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Builds an ORDER BY clause of a SELECT statement.
<u>SetRange</u> (inherited from <u>TMemDataSet</u>)	Sets the starting and ending values of a range, and applies it.
<u>SetRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
<u>SetRangeStart</u> (inherited from <u>TMemDataSet</u>)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
<u>SQLSaved</u> (inherited from <u>TCustomDADataset</u>)	Determines if the <u>SQL</u> property value was saved to the <u>BaseSQL</u> property.
<u>UnLock</u> (inherited from <u>TCustomDADataset</u>)	Releases a record lock.

UnPrepare (inherited from TMemDataSet)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from TMemDataSet)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from TMemDataSet)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

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

5.13.1.2.3.1 ExecProc Method

Executes a SQL statement on the server.

Class

[TCustomPgStoredProc](#)

Syntax

```
procedure ExecProc;
```

Remarks

Call the ExecProc method to execute a SQL statement on the server. The ExecProc method is similar to the [TCustomDADDataSet.Execute](#) method. It is included for compatibility with TStoredProc.

5.13.1.2.3.2 PrepareSQL Method

Describes the parameters of a stored procedure.

Class

[TCustomPgStoredProc](#)

Syntax

```
procedure PrepareSQL;
```

Remarks

Call the PrepareSQL method to describe parameters of a stored procedure and assign SQL statement that calls the stored procedure to the SQL property. If necessary, the Execute method calls it automatically. You can define parameters at design time if Parameters Editor is opened.

5.13.1.3 TCustomPgTable Class

A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.

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

Unit

[PgAccess](#)

Syntax

```
TCustomPgTable = class(TCustomPgDataSet);
```

Remarks

TCustomPgTable implements functionality to access data in a table. Use TCustomPgTable properties and methods to gain direct access to records and fields in an underlying server database without writing SQL statements.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

[TCustomPgDataSet](#)

TCustomPgTable

See Also

- [TPgTable](#)
- [TPgQuery](#)
- [Master/Detail Relationships](#)

5.13.1.3.1 Members

[TCustomPgTable](#) class overview.

Properties

Name	Description
------	-------------

<u>BaseSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>Conditions</u> (inherited from <u>TCustomDADataset</u>)	Used to add WHERE conditions to a query
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>Cursor</u> (inherited from <u>TCustomPgDataSet</u>)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
<u>DataTypeMap</u> (inherited from <u>TCustomDADataset</u>)	Used to set data type mapping rules
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TCustomPgDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchAll</u> (inherited from <u>TCustomPgDataSet</u>)	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from	Used to define the number of rows to be transferred across the network at the same time.

<u>TCustomDADataset</u>)	
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u> (inherited from <u>TCustomPgDataSet</u>)	Returns OID of the record inserted by the last query for table with OIDs.
<u>Limit</u>	Used to set the number of rows retrieved from the query.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.

Macros (inherited from TCustomDADataset)	Makes it possible to change SQL queries easily.
MasterFields (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.
MasterSource (inherited from TCustomDADataset)	Used to specify the data source component which binds current dataset to the master one.
Offset	Used to allow retrieving data from the server starting from the specified row.
Options (inherited from TCustomPgDataSet)	Used to specify the behaviour of TCustomPgDataSet object.
ParamCheck (inherited from TCustomDADataset)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from TCustomDADataset)	Used to indicate how many parameters are there in the Params property.
Params (inherited from TCustomPgDataSet)	Contains parameters for a query's SQL statement.
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.
ReadOnly (inherited from TCustomDADataset)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions (inherited from TCustomDADataset)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from	Used to indicate the number of rows which were inserted, updated, or deleted during the last query

<u>TCustomDADataset</u>)	operation.
<u>SequenceMode</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLLock</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to perform a record lock.
<u>SQLRecCount</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.
<u>SQLUpdate</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying an update to a dataset.
<u>UniDirectional</u> (inherited from <u>TCustomDADataset</u>)	Used if an application does not need bidirectional access to records in the result set.
<u>UpdateObject</u> (inherited from <u>TCustomPgDataSet</u>)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
<u>UpdateRecordTypes</u> (inherited from	Used to indicate the update status for the current record when cached

<u>TMemDataSet</u>)	updates are enabled.
<u>UpdatesPending</u> (inherited from <u>TMemDataSet</u>)	Used to check the status of the cached updates buffer.

Methods

Name	Description
<u>AddWhere</u> (inherited from <u>TCustomDADDataSet</u>)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
<u>ApplyRange</u> (inherited from <u>TMemDataSet</u>)	Applies a range to the dataset.
<u>ApplyUpdates</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Writes dataset's pending cached updates to a database.
<u>BreakExec</u> (inherited from <u>TCustomDADDataSet</u>)	Breaks execution of the SQL statement on the server.
<u>CancelRange</u> (inherited from <u>TMemDataSet</u>)	Removes any ranges currently in effect for a dataset.
<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.
<u>CreateBlobStream</u> (inherited from <u>TCustomDADDataSet</u>)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
<u>CreateProcCall</u> (inherited from <u>TCustomPgDataSet</u>)	Generates the stored procedure call.

DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
DeleteWhere (inherited from TCustomDADataset)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from TMemDataSet)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from TMemDataSet)	Enables changing the starting value for an existing range.
Execute (inherited from TCustomDADataset)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from TCustomDADataset)	Indicates whether SQL statement is still being executed.
Fetched (inherited from TCustomDADataset)	Used to learn whether TCustomDADataset has already fetched all rows.
Fetching (inherited from TCustomDADataset)	Used to learn whether TCustomDADataset is still fetching rows.
FetchingAll (inherited from TCustomDADataset)	Used to learn whether TCustomDADataset is fetching all rows to the end.
FindKey (inherited from TCustomDADataset)	Searches for a record which contains specified field values.
FindMacro (inherited from TCustomDADataset)	Description is not available at the moment.
FindNearest (inherited from TCustomDADataset)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the

<u>TCustomDADataset</u>)	values specified in the KeyValues parameter.
<u>FindParam</u> (inherited from <u>TCustomPgDataset</u>)	Searches for and returns a parameter with the specified name.
<u>GetBlob</u> (inherited from <u>TMemDataset</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetKeyFieldNames</u> (inherited from <u>TCustomDADataset</u>)	Provides a list of available key field names.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetPgCursor</u> (inherited from <u>TCustomPgDataset</u>)	Retrieves a TPgCursor object for a field with known name.
<u>GetPgDate</u> (inherited from <u>TCustomPgDataset</u>)	Retrieves a TPgDate object for a field with known name.
<u>GetPgInterval</u> (inherited from <u>TCustomPgDataset</u>)	Retrieves a TPgInterval object for a field with known name.

GetPgLargeObject (inherited from TCustomPgDataSet)	Retrieves a TPgLargeObject object for a field with known name.
GetPgRow (inherited from TCustomPgDataSet)	Retrieves a TPgRow object for a field with known name.
GetPgTime (inherited from TCustomPgDataSet)	Retrieves a TPgTime object for a field with known name.
GetPgTimeStamp (inherited from TCustomPgDataSet)	Retrieves a TPgTimeStamp object for a field with known name.
GotoCurrent (inherited from TCustomDADDataSet)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from TMemDataSet)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from TMemDataSet)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from TCustomDADDataSet)	Locks the current record.
MacroByName (inherited from TCustomDADDataSet)	Finds a Macro with the name passed in Name.
OpenNext (inherited from TCustomPgDataSet)	Opens the next REFCURSOR for stored procedure that returns more than one cursor.
ParamByName (inherited from TCustomPgDataSet)	Searches for and returns a parameter with the specified name.
Prepare (inherited from TCustomPgDataSet)	Allocates, opens, and parses cursor for a query.

<u>TCustomDADataset</u>)	
<u>RefreshRecord</u> (inherited from <u>TCustomDADataset</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADataset</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>SaveSQL</u> (inherited from <u>TCustomDADataset</u>)	Saves the SQL property value to BaseSQL.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Builds an ORDER BY clause of a SELECT statement.
<u>SetRange</u> (inherited from <u>TMemDataSet</u>)	Sets the starting and ending values of a range, and applies it.
<u>SetRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
<u>SetRangeStart</u> (inherited from <u>TMemDataSet</u>)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
<u>SQLSaved</u> (inherited from <u>TCustomDADataset</u>)	Determines if the <u>SQL</u> property value was saved to the <u>BaseSQL</u> property.

UnLock (inherited from TCustomDADataset)	Releases a record lock.
UnPrepare (inherited from TMemDataSet)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from TMemDataSet)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from TMemDataSet)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from TCustomDADataset)	Occurs after a component has executed a query to database.
AfterFetch (inherited from TCustomDADataset)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from TCustomDADataset)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from TCustomDADataset)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from TCustomDADataset)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from TMemDataSet)	Occurs when a single update component can not handle the updates.

5.13.1.3.2 Properties

Properties of the **TCustomPgTable** class.

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

Public

Name	Description
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
Conditions (inherited from TCustomDADataset)	Used to add WHERE conditions to a query
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from TCustomPgDataSet)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
DataTypeMap (inherited from TCustomDADataset)	Used to set data type mapping rules
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of parameters.
DetailFields (inherited from TCustomDADataset)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from TCustomDADataset)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from	Used to refresh record by RETURNING clause when insert or

<u>TCustomPgDataSet</u>)	update is performed.
<u>FetchAll</u> (inherited from <u>TCustomPgDataSet</u>)	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u> (inherited from <u>TCustomPgDataSet</u>)	Returns OID of the record inserted by the last query for table with OIDs.
<u>Limit</u>	Used to set the number of rows retrieved from the query.
<u>LocalConstraints</u> (inherited from	Used to avoid setting the Required property of a TField component for

<u>TMemDataSet</u>)	NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>Offset</u>	Used to allow retrieving data from the server starting from the specified row.
<u>Options</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the behaviour of TCustomPgDataSet object.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomPgDataSet</u>)	Contains parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.

<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>SequenceMode</u> (inherited from <u>TCustomPgDataset</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLLock</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to perform a record lock.
<u>SQLRecCount</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.
<u>SQLUpdate</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying an update to a dataset.

UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomPgDataSet)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
UpdateRecordTypes (inherited from TMemDataSet)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from TMemDataSet)	Used to check the status of the cached updates buffer.

See Also

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

5.13.1.3.2.1 Limit Property

Used to set the number of rows retrieved from the query.

Class

[TCustomPgTable](#)

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](#)

5.13.1.3.2.2 Offset Property

Used to allow retrieving data from the server starting from the specified row.

Class

[TCustomPgTable](#)

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](#)

5.13.1.4 TCustomPgTimeStampField Class

A base class defining functionality for the classes derived from it.

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

Unit

[PgAccess](#)

Syntax

```
TCustomPgTimeStampField = class(TField);
```

Remarks

TCustomPgTimeStampField is a base dataset component that defines functionality for classes derived from it. Applications never use TCustomPgTimeStampField objects directly. Instead they use descendants of TCustomPgTimeStampField

5.13.1.4.1 Members

[TCustomPgTimeStampField](#) class overview.

Properties

Name	Description
------	-------------

[AsPgTimeStamp](#)

Used to provide access to a TPgTimeStamp object.

5.13.1.4.2 Properties

Properties of the **TCustomPgTimeStampField** class.

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

Public

Name	Description
AsPgTimeStamp	Used to provide access to a TPgTimeStamp object.

See Also

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

5.13.1.4.2.1 AsPgTimeStamp Property

Used to provide access to a TPgTimeStamp object.

Class

[TCustomPgTimeStampField](#)

Syntax

```
property ASPgTimeStamp: TCustomPgTimeStamp;
```

Remarks

Use the AsTimeStamp property to get access to a TPgTimeStamp object which you can use for manipulations with timestamp value.

5.13.1.5 TPgConnection Class

Represents an open connection to a PostgreSQL database.

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

Unit

[PgAccess](#)

Syntax

```
TPgConnection = class(TCustomDAConnection);
```

Remarks

The TPgConnection component is used to maintain connection to a PostgreSQL database. After setting the Username, Password, Server, and Database properties, you can establish a connection to the database by calling the Connect method or setting the Connected property to True. There are also many properties at the connection level that affect the default behavior of the queries executed within this connection.

Use this component in conjunction with TPgQuery, TPqTable, TPgStoredProc or other components for convenient interoperation with PostgreSQL database.

TPgConnection object represents a unique connection to PostgreSQL database.

Note: If the port differs from the default one (5432) please use the Port property to set it.

Inheritance Hierarchy

[TCustomDAConnection](#)

TPgConnection

5.13.1.5.1 Members

[TPgConnection](#) class overview.

Properties

Name	Description
ConnectDialog (inherited from TCustomDAConnection)	Allows to link a TCustomConnectDialog component.
ConnectionTimeout	Used to set the time to wait while trying to establish a connection before terminating the attempt and generating an error.
ConnectionString (inherited from TCustomDAConnection)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from TCustomDAConnection)	Allows customizing line breaks in string fields and parameters.

Database	Used to specify the name of the database to be used once a connection is open.
InTransaction (inherited from TCustomDAConnection)	Indicates whether the transaction is active.
LoginPrompt (inherited from TCustomDAConnection)	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the behaviour of the TPgConnectionOptions object.
Password	Used to specify a password for a connection.
Pooling (inherited from TCustomDAConnection)	Enables or disables using connection pool.
PoolingOptions (inherited from TCustomDAConnection)	Specifies the behaviour of connection pool.
Port	Used to specify the port for connection.
ProcessID	Returns the process ID (PID) of the backend server process handling this connection.
ProtocolVersion	Used to set the version of protocol for communication with PostgreSQL server.
Schema	Used to change the search path of the connection to the specified schema, or get the first value from the search path.
Server	Contains the server name.
ServerVersion	Holds the PostgreSQL server version number.
ServerVersionFull	Holds full PostgreSQL server version including version number and some additional information.

SSLOptions	Used to set the properties required for protected SSL connection with the server.
Username	Contains username.

Methods

Name	Description
ApplyUpdates (inherited from TCustomDACConnection)	Overloaded. Applies changes in datasets.
BreakExec	Used to cancel a query that is currently being executed on the connection.
Commit (inherited from TCustomDACConnection)	Commits current transaction.
Connect (inherited from TCustomDACConnection)	Establishes a connection to the server.
CreateDataSet	Returns a new instance of TPgQuery class and associates it with the connection object.
CreateMetaData	Returns a new instance of the TPgMetaData class.
CreateSQL	Returns a new instance of TPgSQL class and associates it with the connection object.
CreateTransaction	Returns a new instance of the TPgTransaction class.
Disconnect (inherited from TCustomDACConnection)	Performs disconnect.
ExecProc (inherited from TCustomDACConnection)	Allows to execute stored procedure or function providing its name and parameters.

<u>ExecProcEx</u> (inherited from <u>TCustomDAConnection</u>)	Allows to execute a stored procedure or function.
<u>ExecSQL</u> (inherited from <u>TCustomDAConnection</u>)	Executes a SQL statement with parameters.
<u>ExecSQLEx</u> (inherited from <u>TCustomDAConnection</u>)	Executes any SQL statement outside the TQuery or TSQL components.
<u>GetDatabaseNames</u> (inherited from <u>TCustomDAConnection</u>)	Returns a database list from the server.
<u>GetKeyFieldNames</u> (inherited from <u>TCustomDAConnection</u>)	Provides a list of available key field names.
<u>GetRowType</u>	Overloaded. Used to get a TPgRowType object for composite type.
<u>GetStoredProcNames</u> (inherited from <u>TCustomDAConnection</u>)	Returns a list of stored procedures from the server.
<u>GetTableNames</u> (inherited from <u>TCustomDAConnection</u>)	Provides a list of available tables names.
<u>MonitorMessage</u> (inherited from <u>TCustomDAConnection</u>)	Sends a specified message through the <u>TCustomDASQLMonitor</u> component.
<u>Ping</u> (inherited from <u>TCustomDAConnection</u>)	Used to check state of connection to the server.
<u>ReleaseSavepoint</u>	Releases the specified savepoint without affecting any work that has been performed after its creation.
<u>RemoveFromPool</u> (inherited from <u>TCustomDAConnection</u>)	Marks the connection that should not be returned to the pool after disconnect.

TCustomDAConnection)	
Rollback (inherited from TCustomDAConnection)	Discards all current data changes and ends transaction.
RollbackToSavepoint	Cancels all updates for the current transaction.
Savepoint	Defines a point in the transaction to which you can roll back later.
StartTransaction	Overloaded. Starts a new user transaction against the database server.

Events

Name	Description
OnConnectionLost (inherited from TCustomDAConnection)	This event occurs when connection was lost.
OnError (inherited from TCustomDAConnection)	This event occurs when an error has arisen in the connection.
OnNotice	Occurs when a message or notice is received from PostgreSQL server.
OnNotification	Occurs when an asynchronous notification is received from PostgreSQL server.

5.13.1.5.2 Properties

Properties of the **TPgConnection** class.

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

Public

Name	Description
------	-------------

<u>ConnectDialog</u> (inherited from <u>TCustomDAConnection</u>)	Allows to link a <u>TCustomConnectDialog</u> component.
<u>ConnectionString</u> (inherited from <u>TCustomDAConnection</u>)	Used to specify the connection information, such as: UserName, Password, Server, etc.
<u>ConvertEOL</u> (inherited from <u>TCustomDAConnection</u>)	Allows customizing line breaks in string fields and parameters.
<u>InTransaction</u> (inherited from <u>TCustomDAConnection</u>)	Indicates whether the transaction is active.
<u>LoginPrompt</u> (inherited from <u>TCustomDAConnection</u>)	Specifies whether a login dialog appears immediately before opening a new connection.
<u>Pooling</u> (inherited from <u>TCustomDAConnection</u>)	Enables or disables using connection pool.
<u>PoolingOptions</u> (inherited from <u>TCustomDAConnection</u>)	Specifies the behaviour of connection pool.
<u>ProcessID</u>	Returns the process ID (PID) of the backend server process handling this connection.
<u>ServerVersion</u>	Holds the PostgreSQL server version number.
<u>ServerVersionFull</u>	Holds full PostgreSQL server version including version number and some additional information.

Published

Name	Description
<u>ConnectionTimeout</u>	Used to set the time to wait while trying to establish a connection before terminating the attempt and generating an error.

Database	Used to specify the name of the database to be used once a connection is open.
Options	Specifies the behaviour of the TPgConnectionOptions object.
Password	Used to specify a password for a connection.
Port	Used to specify the port for connection.
ProtocolVersion	Used to set the version of protocol for communication with PostgreSQL server.
Schema	Used to change the search path of the connection to the specified schema, or get the first value from the search path.
Server	Contains the server name.
SSLOptions	Used to set the properties required for protected SSL connection with the server.
Username	Contains username.

See Also

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

5.13.1.5.2.1 ConnectionTimeout Property

Used to set the time to wait while trying to establish a connection before terminating the attempt and generating an error.

Class

[TPgConnection](#)

Syntax

```
property ConnectionTimeout: integer default
    DefValConnectionTimeout;
```

Remarks

Use the ConnectionTimeout property to set the time to wait while trying to establish a connection before terminating the attempt and generating an error. The 0 value indicates no limit.

5.13.1.5.2.2 Database Property

Used to specify the name of the database to be used once a connection is open.

Class

[TPgConnection](#)

Syntax

```
property Database: string;
```

Remarks

Use the Database property to specify the name of the database to be used once a connection is open. If the Database property is empty, it is assumed that the name of a database is identical to the user name.

See Also

- [TCustomDAConnection.Server](#)
- [Port](#)
- [TCustomDAConnection.Username](#)
- [TCustomDAConnection.Password](#)
- [TCustomDAConnection.GetDatabaseNames](#)

5.13.1.5.2.3 Options Property

Specifies the behaviour of the TPgConnectionOptions object.

Class

[TPgConnection](#)

Syntax

```
property Options: TPgConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TPgConnectionOptions object. Descriptions of all options are in the table below.

Option Name	Description
ApplicationName	Defines the application name for connecting to the server. This name will be displayed in the pg_stat_activity view and included in CSV log entries. Only printable ASCII characters may be used in the ApplicationName value. Other characters will be replaced with question marks (?).
Charset	Used to set the character set that PgDAC uses to read and write character data.
EnableComposites	Used to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.
EnableDomains	Used to map fields of domain types to TField with the base domain type.
EnableGeometrics	Used to map fields of geometric types to TPgGeometricField.
EnablePgTimeStamps	Used to map DATE, TIME, and TIMESTAMP fields to TPgDateField, TPgTimeField, and TPgTimeStampField accordingly.
IPVersion	Used to specify the version of the Internet Protocol.
UseUnicode	Used to specify a value indicating whether the UTF8 charset will be used.

5.13.1.5.2.4 Password Property

Used to specify a password for a connection.

Class

[TPgConnection](#)

Syntax

```
property Password: string;
```

Remarks

Use the Password property to specify a password for a connection. When property is being changed TPgConnection calls Disconnect method.

See Also

- [Username](#)
- [Server](#)
- [TCustomDAConnection.Connect](#)

5.13.1.5.2.5 Port Property

Used to specify the port for connection.

Class

[TPgConnection](#)

Syntax

```
property Port: integer default PgDefValPort;
```

Remarks

Use the Port property to specify the port number of PostgreSQL database for connection.

See Also

- [TCustomDAConnection.Server](#)
- [Database](#)

5.13.1.5.2.6 ProcessID Property

Returns the process ID (PID) of the backend server process handling this connection.

Class

[TPgConnection](#)

Syntax

```
property ProcessID: integer;
```

Remarks

Used to return the process ID (PID) of the backend server process handling this connection. The backend PID is useful for debugging purposes and for comparison to NOTIFY messages (which include the PID of the notifying backend process).

5.13.1.5.2.7 ProtocolVersion Property

Used to set the version of protocol for communication with PostgreSQL server.

Class

[TPgConnection](#)

Syntax

```
property ProtocolVersion: TProtocolVersion default  
DefValProtocol;
```

Remarks

Set the ProtocolVersion to pv20 to work with PostgreSQL servers 7.3 or lower that don't support the 3.0 protocol.

5.13.1.5.2.8 Schema Property

Used to change the search path of the connection to the specified schema, or get the first value from the search path.

Class

[TPgConnection](#)

Syntax

```
property Schema: string stored IsSchemaStored;
```

Remarks

Use the Schema property to change the search path of the connection to the specified schema, or get the first value from the search path.

Set the Schema property to change the search path of the connection to the specified schema. When connection is open, read the value of the property to get the name of the current schema (the first value from the current search path).

The TPgQuery, TPgStoredProc and other PgDAC components can implicitly use this property. Therefore you may have problems in case your search path contains several schemas and you use objects not from the current one.

5.13.1.5.2.9 Server Property

Contains the server name.

Class

[TPgConnection](#)

Syntax

```
property Server: string;
```

Remarks

Use the Server property to supply server name to handle server's request for a login. When the property is being changed TPgConnection calls the Disconnect method.

See Also

- [Username](#)
- [Password](#)
- [TCustomDAConnection.Connect](#)

5.13.1.5.2.10 ServerVersion Property

Holds the PostgreSQL server version number.

Class

[TPgConnection](#)

Syntax

```
property ServerVersion: string;
```

Remarks

Read the ServerVersion property to get the PostgreSQL server version number in a string, for a example '8.2.5'

Works only when a TPgConnection instance is connected.

5.13.1.5.2.11 ServerVersionFull Property

Holds full PostgreSQL server version including version number and some additional information.

Class

[TPgConnection](#)

Syntax

```
property ServerVersionFull: string;
```

Remarks

Read the `ServerVersionFull` property to get full PostgreSQL server version including version number and some additional information.

Works only when a `TPgConnection` instance is connected.

5.13.1.5.2.12 SSLOptions Property

Used to set the properties required for protected SSL connection with the server.

Class

[TPgConnection](#)

Syntax

```
property SSLOptions: TPgConnectionSSLOptions;
```

Remarks

Use the `SSLOptions` property to set the properties required for protected SSL connection with the server.

Descriptions of all options are in the table below.

Option Name	Description
CACert	Holds the pathname to the certificate authority file.
Cert	Holds the pathname to the certificate file.
CipherList	Holds the list of allowed ciphers to use for SSL encryption.
Key	Holds the pathname to the key file.
Mode	Used to determine whether or with what priority an SSL connection will be negotiated with the server.

5.13.1.5.2.13 Username Property

Contains username.

Class

[TPgConnection](#)

Syntax

```
property Username: string;
```

Remarks

Use the Username property to supply a user name to handle server's request for a login. When the property is being changed TPgConnection calls Disconnect method.

Note: The UserName property will be used as a default value for the Database and Schema parameters if they are empty.

See Also

- [Password](#)
- [Server](#)
- [TCustomDAConnection.Connect](#)

5.13.1.5.3 Methods

Methods of the **TPgConnection** class.

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

Public

Name	Description
ApplyUpdates (inherited from TCustomDAConnection)	Overloaded. Applies changes in datasets.
BreakExec	Used to cancel a query that is currently being executed on the connection.
Commit (inherited from TCustomDAConnection)	Commits current transaction.
Connect (inherited from TCustomDAConnection)	Establishes a connection to the server.
CreateDataSet	Returns a new instance of TPgQuery class and associates it with the connection object.
CreateMetaData	Returns a new instance of the TPgMetaData class.

CreateSQL	Returns a new instance of TPgSQL class and associates it with the connection object.
CreateTransaction	Returns a new instance of the TPgTransaction class.
Disconnect (inherited from TCustomDAConnection)	Performs disconnect.
ExecProc (inherited from TCustomDAConnection)	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx (inherited from TCustomDAConnection)	Allows to execute a stored procedure or function.
ExecSQL (inherited from TCustomDAConnection)	Executes a SQL statement with parameters.
ExecSQLEx (inherited from TCustomDAConnection)	Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames (inherited from TCustomDAConnection)	Returns a database list from the server.
GetKeyFieldNames (inherited from TCustomDAConnection)	Provides a list of available key field names.
GetRowType	Overloaded. Used to get a TPgRowType object for composite type.
GetStoredProcNames (inherited from TCustomDAConnection)	Returns a list of stored procedures from the server.
GetTableNames (inherited from TCustomDAConnection)	Provides a list of available tables names.

MonitorMessage (inherited from TCustomDAConnection)	Sends a specified message through the TCustomDASQLMonitor component.
Ping (inherited from TCustomDAConnection)	Used to check state of connection to the server.
ReleaseSavepoint	Releases the specified savepoint without affecting any work that has been performed after its creation.
RemoveFromPool (inherited from TCustomDAConnection)	Marks the connection that should not be returned to the pool after disconnect.
Rollback (inherited from TCustomDAConnection)	Discards all current data changes and ends transaction.
RollbackToSavepoint	Cancels all updates for the current transaction.
Savepoint	Defines a point in the transaction to which you can roll back later.
StartTransaction	Overloaded. Starts a new user transaction against the database server.

See Also

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

5.13.1.5.3.1 BreakExec Method

Used to cancel a query that is currently being executed on the connection.

Class

[TPgConnection](#)

Syntax

```
procedure BreakExec;
```

Remarks

Call the BreakExec method to cancel a query that is currently being executed on the connection. If you call the Execute method of TPgQuery, it will not return until the query is executed. So you need to call the BreakExec method from another thread to cancel the query.

5.13.1.5.3.2 CreateDataSet Method

Returns a new instance of TPgQuery class and associates it with the connection object.

Class

[TPgConnection](#)

Syntax

```
function CreateDataSet(AOwner: TComponent = nil):  
TCustomDADataset; override;
```

Return Value

a new instance of TPgQuery class.

Remarks

The CreateDataSet method returns a new instance of TPgQuery class and associates it with the connection object.

5.13.1.5.3.3 CreateMetaData Method

Returns a new instance of the TPgMetaData class.

Class

[TPgConnection](#)

Syntax

```
function CreateMetaData: TDAMetaData; override;
```

Return Value

a new instance of the TPgMetaData class.

Remarks

The CreateMetaData method returns a new instance of the TPgMetaData class and associates it with the connection object.

5.13.1.5.3.4 CreateSQL Method

Returns a new instance of TPgSQL class and associates it with the connection object.

Class

[TPgConnection](#)

Syntax

```
function CreateSQL: TCustomDASQL; override;
```

Return Value

a new instance of TPgSQL class.

Remarks

Call the CreateSQL to return a new instance of TPgSQL class and associates it with the connection object.

5.13.1.5.3.5 CreateTransaction Method

Returns a new instance of the TPgTransaction class.

Class

[TPgConnection](#)

Syntax

```
function CreateTransaction: TDATransaction; override;
```

Return Value

a new instance of the TPgTransaction class.

Remarks

The createTransaction method returns a new instance of the TPgTransaction class.

5.13.1.5.3.6 GetRow Type Method

Used to get a TPgRowType object for composite type.

Class

[TPgConnection](#)

Overload List

Name	Description
GetRowType(TypeOID: integer)	Used to get a TPgRowType object for composite type by its OID.
GetRowType(const TypeName: string)	Used to get a TPgRowType object for composite type by its name.

Used to get a TPgRowType object for composite type by its OID.

Class

[TPgConnection](#)

Syntax

```
function GetRowType(TypeOID: integer): TPgRowType; overload;
```

Parameters

TypeOID

OID of the composite type.

Return Value

a TPgRowType object.

Remarks

Use the GetRowType function to set the RowType property of TPgRow when you know the OID of the composite type. You can also use a TPgRowType object to get a list of composite type attributes.

Used to get a TPgRowType object for composite type by its name.

Class

[TPgConnection](#)

Syntax

```
function GetRowType(const TypeName: string): TPgRowType;  
overload;
```

Parameters

TypeName

Name of the composite type.

Return Value

a TPgRowType object.

Remarks

Use the `GetRowType` function to set the `RowType` property of `TPgRow` when you know the name of the composite type. You can also use a `TPgRowType` object to get a list of composite type attributes.

5.13.1.5.3.7 ReleaseSavepoint Method

Releases the specified savepoint without affecting any work that has been performed after its creation.

Class

[TPgConnection](#)

Syntax

```
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](#)

5.13.1.5.3.8 RollbackToSavepoint Method

Cancels all updates for the current transaction.

Class

[TPgConnection](#)

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

- [ReleaseSavepoint](#)
- [Savepoint](#)

5.13.1.5.3.9 Savepoint Method

Defines a point in the transaction to which you can roll back later.

Class

[TPgConnection](#)

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

- [ReleaseSavepoint](#)
- [RollbackToSavepoint](#)

5.13.1.5.3.10 StartTransaction Method

Starts a new user transaction against the database server.

Class

[TPgConnection](#)

Overload List

Name	Description
StartTransaction	Starts a new user transaction against the database server.
StartTransaction(IsolationLevel: TPgIsolationLevel; ReadOnly: boolean)	Starts a new user transaction against the database server.

Starts a new user transaction against the database server.

Class

[TPgConnection](#)

Syntax

```
procedure StartTransaction; overload; override;
```

Remarks

StartTransaction is an overload method for [TCustomDAConnection.StartTransaction](#). Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction property. If InTransaction is True, it indicates that a transaction is already in progress, a subsequent call to StartTransaction without first calling [TCustomDAConnection.Commit](#) or [TCustomDAConnection.Rollback](#) to end the current transaction raises EDatabaseError.

Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes or Rollback to cancel them.

Starts a new user transaction against the database server.

Class

[TPgConnection](#)

Syntax

```
procedure StartTransaction(IsolationLevel: TPgIsolationLevel;  
ReadOnly: boolean = False); reintroduce; overload;
```

Parameters

IsolationLevel

Specifies how the transactions containing database modifications are handled.

ReadOnly

If True, read-only transaction is started that cannot modify data in the database.

Remarks

Call the `StartTransaction` method to begin a new user transaction against the database server. Before calling `StartTransaction`, an application should check the status of the `InTransaction` property. If `InTransaction` is `True`, it indicates that a transaction is already in progress, a subsequent call to `StartTransaction` without first calling [TCustomDACConnection.Commit](#) or [TCustomDACConnection.Rollback](#) to end the current transaction raises `EDatabaseError`. Calling `StartTransaction` when connection is closed also raises `EDatabaseError`.

Updates, insertions, and deletions that take place after a call to `StartTransaction` are held by the server until an application calls `Commit` to save the changes or `Rollback` to cancel them.

5.13.1.5.4 Events

Events of the **TPgConnection** class.

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

Public

Name	Description
OnConnectionLost (inherited from TCustomDACConnection)	This event occurs when connection was lost.
OnError (inherited from TCustomDACConnection)	This event occurs when an error has arisen in the connection.

Published

Name	Description
OnNotice	Occurs when a message or notice is received from PostgreSQL server.
OnNotification	Occurs when an asynchronous notification is received from PostgreSQL server.

See Also

- [TPgConnection Class](#)

- [TPgConnection Class Members](#)

5.13.1.5.4.1 OnNotice Event

Occurs when a message or notice is received from PostgreSQL server.

Class

[TPgConnection](#)

Syntax

```
property OnNotice: TPgNoticeEvent;
```

Remarks

The OnNotice event occurs when PostgreSQL server sends a message or multiple messages with Severity lower than ERROR.

To set the message levels to be sent to the client, use the PostgreSQL client_min_messages configuration parameter. For more information refer to the PostgreSQL documentation.

5.13.1.5.4.2 OnNotification Event

Occurs when an asynchronous notification is received from PostgreSQL server.

Class

[TPgConnection](#)

Syntax

```
property OnNotification: TPgNotificationEvent;
```

Remarks

The OnNotification event occurs when an asynchronous notification is received from PostgreSQL server. Notification can be sent using NOTIFY command. To receive notifications use LISTEN command. To stop receiving notifications use UNLISTEN command. You can read more information about these commands in the PostgreSQL documentation.

Note: Notifications can be received only when a call to the server is performed (for example, on executing SQL statement). While the connection is idle, notifications are not received. To receive notifications while the connection is idle, use the TPgAlerter component.

5.13.1.6 TPgConnectionOptions Class

This class allows setting up the behaviour of the TPgConnection class.

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

Unit

[PgAccess](#)

Syntax

```
TPgConnectionOptions = class(TDACConnectionOptions);
```

Inheritance Hierarchy

[TDACConnectionOptions](#)

TPgConnectionOptions

5.13.1.6.1 Members

[TPgConnectionOptions](#) class overview.

Properties

Name	Description
AllowImplicitConnect (inherited from TDACConnectionOptions)	Specifies whether to allow or not implicit connection opening.
ApplicationName	Defines the application name for connecting to the server. This name will be displayed in the pg_stat_activity view and included in CSV log entries. Only printable ASCII characters may be used in the ApplicationName value. Other characters will be replaced with question marks (?).
Charset	Used to set the character set that PgDAC uses to read and write character data.
DefaultSortType (inherited from TDACConnectionOptions)	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.
DisconnectedMode (inherited from TDACConnectionOptions)	Used to open a connection only when needed for performing a server call and closes after performing the operation.

EnableComposites	Used to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.
EnableDomains	Used to map fields of domain types to TField with the base domain type.
EnableGeometrics	Used to map fields of geometric types to TPgGeometricField.
EnablePgTimeStamps	Used to map DATE, TIME, and TIMESTAMP fields to TPgDateField, TPgTimeField, and TPgTimeStampField accordingly.
IPVersion	Used to specify the version of the Internet Protocol.
KeepDesignConnected (inherited from TDAConnectionOptions)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from TDAConnectionOptions)	If True, the TCustomDAConnection.OnConnecti onLost event occurs and a failover operation can be performed after connection breaks.
UseUnicode	Used to specify a value indicating whether the UTF8 charset will be used.

5.13.1.6.2 Properties

Properties of the **TPgConnectionOptions** class.

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

Public

Name	Description
DefaultSortType (inherited from TDAConnectionOptions)	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.

DisconnectedMode (inherited from TDACConnectionOptions)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected (inherited from TDACConnectionOptions)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from TDACConnectionOptions)	If True, the TCustomDACConnection.OnConnecti onLost event occurs and a failover operation can be performed after connection breaks.

Published

Name	Description
AllowImplicitConnect (inherited from TDACConnectionOptions)	Specifies whether to allow or not implicit connection opening.
ApplicationName	Defines the application name for connecting to the server. This name will be displayed in the pg_stat_activity view and included in CSV log entries. Only printable ASCII characters may be used in the ApplicationName value. Other characters will be replaced with question marks (?).
Charset	Used to set the character set that PgDAC uses to read and write character data.
EnableComposites	Used to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.
EnableDomains	Used to map fields of domain types to TField with the base domain type.
EnableGeometrics	Used to map fields of geometric types to TPgGeometricField.
EnablePgTimeStamps	Used to map DATE, TIME, and TIMESTAMP fields to TPgDateField,

	TPgTimeField, and TPgTimeStampField accordingly.
IPVersion	Used to specify the version of the Internet Protocol.
UseUnicode	Used to specify a value indicating whether the UTF8 charset will be used.

See Also

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

5.13.1.6.2.1 ApplicationName Property

Defines the application name for connecting to the server. This name will be displayed in the pg_stat_activity view and included in CSV log entries. Only printable ASCII characters may be used in the ApplicationName value. Other characters will be replaced with question marks (?).

Class

[TPgConnectionOptions](#)

Syntax

```
property ApplicationName: string;
```

Remarks

Note: required PostgreSQL 9.0 or higher.

5.13.1.6.2.2 Charset Property

Used to set the character set that PgDAC uses to read and write character data.

Class

[TPgConnectionOptions](#)

Syntax

```
property Charset: string;
```

Remarks

Use the Charset property to set the character set that PgDAC uses to read and write

character data.

5.13.1.6.2.3 EnableComposites Property

Used to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.

Class

[TPgConnectionOptions](#)

Syntax

```
property EnableComposites: boolean default True;
```

Remarks

Use the EnableComposites property to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.

When EnableComposites is set to True, PgDAC detects fields of composite type and creates separate fields for each attribute of composite type. If the ObjectView property is set to True, PgDAC also creates an instance of TADTField that is parent for all attributes fields.

When EnableComposites is False, fields of composite types are not detected and mapped to TStringField, TMemoField, TWideStringField, or TWideMemoField depending on the UnknownAsString and UseUnicode options.

5.13.1.6.2.4 EnableDomains Property

Used to map fields of domain types to TField with the base domain type.

Class

[TPgConnectionOptions](#)

Syntax

```
property EnableDomains: boolean default True;
```

Remarks

Use the EnableDomains property to map fields of domain types to TField with base domain type.

When EnableDomains is set to True, the base data type of the domain is detected and the field with the domain data type is mapped to TField with the corresponding type. When EnableDomains is False, fields of domain types are mapped to TStringField, TMemoField,

TWideStringField, or TWideMemoField depending on the [UnknownAsString](#) and [UseUnicode](#) options.

5.13.1.6.2.5 EnableGeometrics Property

Used to map fields of geometric types to TPgGeometricField.

Class

[TPgConnectionOptions](#)

Syntax

```
property EnableGeometrics: boolean default True;
```

Remarks

Use the EnableGeometrics property to map fields of geometric types to TPgGeometricField. When EnableGeometrics is set to True, fields with POINT, LSEG, BOX, PATH, POLYGON, and CIRCLE data type are mapped to TPgGeometricField. When EnableGeometrics is False, fields of geometric types are mapped to TStringField, TMemoField, TWideStringField, or TWideMemoField depending on the UnknownAsString and UseUnicode options.

5.13.1.6.2.6 EnablePgTimeStamps Property

Used to map DATE, TIME, and TIMESTAMP fields to TPgDateField, TPgTimeField, and TPgTimeStampField accordingly.

Class

[TPgConnectionOptions](#)

Syntax

```
property EnablePgTimeStamps: boolean default False;
```

Remarks

Use the EnablePgTimeStamp property to map DATE, TIME, and TIMESTAMP fields to TPgDateField, TPgTimeField, and TPgTimeStampField accordingly.

By default this option is set to False, and DATE, TIME, and TIMESTAMP fields are mapped to standard TDateField, TTimeField, and TDateTimeField. Set this option to True to work with values than can be represented by TDateTime variable. These values include dates before 01-01-0100 or after 12-31-9999 and special values INFINITY and -INFINITY that can hold TIMESTAMP field. You can work with such values using special PgDAC field types: TPgDateField, TPgTimeField, and TPgTimeStampField. TPgTimeField and

TPgTimeStampField allows also getting timezone value for fields WITH TIMEZONE.

5.13.1.6.2.7 IPVersion Property

Used to specify the version of the Internet Protocol.

Class

[TPgConnectionOptions](#)

Syntax

```
property IPVersion: TIPVersion default DefValIPVersion;
```

Remarks

Use the IPVersion property to specify the Internet Protocol version. The default value is T:Devart.Dac.TIPVersion.

See Also

- T:Devart.Dac.TIPVersion

5.13.1.6.2.8 UseUnicode Property

Used to specify a value indicating whether the UTF8 charset will be used.

Class

[TPgConnectionOptions](#)

Syntax

```
property UseUnicode: boolean default DefValUseUnicode;
```

Remarks

Use the UseUnicode property to set PostgreSQL client charset to UTF8 and converts client data according to this charset.

When a value is assigned to this property, TPgConnection is closed.

5.13.1.7 TPgConnectionSSLOptions Class

This class allows setting up the behaviour of the TPgConnection class.

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

Unit

[PgAccess](#)

Syntax

```
TPgConnectionSSLOptions = class(TPersistent);
```

5.13.1.7.1 Members

[TPgConnectionSSLOptions](#) class overview.

Properties

Name	Description
CACert	Holds the pathname to the certificate authority file.
Cert	Holds the pathname to the certificate file.
CipherList	Holds the list of allowed ciphers to use for SSL encryption.
Key	Holds the pathname to the key file.
Mode	Used to determine whether or with what priority an SSL connection will be negotiated with the server.

5.13.1.7.2 Properties

Properties of the **TPgConnectionSSLOptions** class.

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

Published

Name	Description
CACert	Holds the pathname to the certificate authority file.
Cert	Holds the pathname to the certificate file.
CipherList	Holds the list of allowed ciphers to use for SSL encryption.
Key	Holds the pathname to the key file.

[Mode](#)

Used to determine whether or with what priority an SSL connection will be negotiated with the server.

See Also

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

5.13.1.7.2.1 CACert Property

Holds the pathname to the certificate authority file.

Class

[TPgConnectionSSLOptions](#)

Syntax

```
property CACert: string;
```

Remarks

CACert is the pathname to the certificate authority file.

5.13.1.7.2.2 Cert Property

Holds the pathname to the certificate file.

Class

[TPgConnectionSSLOptions](#)

Syntax

```
property Cert: string;
```

Remarks

Cert is the pathname to the certificate file.

5.13.1.7.2.3 CipherList Property

Holds the list of allowed ciphers to use for SSL encryption.

Class

[TPgConnectionSSLOptions](#)

Syntax

```
property CipherList: string;
```

Remarks

CipherList is the list of allowed ciphers to use for SSL encryption.

5.13.1.7.2.4 Key Property

Holds the pathname to the key file.

Class

[TPgConnectionSSLOptions](#)

Syntax

```
property Key: string;
```

Remarks

Key is the pathname to the key file.

5.13.1.7.2.5 Mode Property

Used to determine whether or with what priority an SSL connection will be negotiated with the server.

Class

[TPgConnectionSSLOptions](#)

Syntax

```
property Mode: TSSLMode default DefValSSLMode;
```

Remarks

Use the Mode property to determine whether or with what priority an SSL connection will be negotiated with the server.

5.13.1.8 TPgCursorField Class

A class providing access to the PostgreSQL cursor fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgCursorField = class(TDACursorField);
```

Remarks

The TPgCursorField class provides access to the PostgreSQL cursor fields.

Inheritance Hierarchy

TDACursorField

TPgCursorField

5.13.1.8.1 Members

[TPgCursorField](#) class overview.

Properties

Name	Description
AsCursor	Used to provide access to a TPgCursor object.

5.13.1.8.2 Properties

Properties of the **TPgCursorField** class.

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

Public

Name	Description
AsCursor	Used to provide access to a TPgCursor object.

See Also

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

5.13.1.8.2.1 AsCursor Property

Used to provide access to a TPgCursor object.

Class

[TPgCursorField](#)

Syntax

```
property AsCursor: TPgRefCursor;
```

Remarks

Use the AsCursor property to provide access to a TPgCursor object you can use for manipulations with the cursor value.

5.13.1.9 TPgDataSetOptions Class

This class allows setting up the behaviour of the TCustomPgDataSet class.

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

Unit

[PgAccess](#)

Syntax

```
TPgDataSetOptions = class(TDADatasetOptions);
```

Inheritance Hierarchy

[TDADatasetOptions](#)

TPgDataSetOptions

5.13.1.9.1 Members

[TPgDataSetOptions](#) class overview.

Properties

Name	Description
AutoDeleteBlob	Used to delete large objects from the database automatically when a record is deleted from the dataset.
AutoPrepare (inherited from	Used to execute automatic TCustomDADataset.Prepare on the query execution.

<u>TDADatasetOptions</u>)	
<u>CacheBlobs</u>	Used to allocate local memory buffer to hold a copy of the large object content.
<u>CacheCalcFields</u> (inherited from <u>TDADatasetOptions</u>)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
<u>CompressBlobMode</u> (inherited from <u>TDADatasetOptions</u>)	Used to store values of the BLOB fields in compressed form.
<u>CursorWithHold</u>	Used to open query in the FetchAll=False mode without transaction.
<u>DefaultValues</u> (inherited from <u>TDADatasetOptions</u>)	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
<u>DeferredBlobRead</u>	Used to fetch values of large objects when they are explicitly requested.
<u>DetailDelay</u> (inherited from <u>TDADatasetOptions</u>)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
<u>ExtendedFieldsInfo</u>	Used to perform an additional query to get information about returned fields and the tables they belong to.
<u>FieldsOrigin</u> (inherited from <u>TDADatasetOptions</u>)	Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.
<u>FlatBuffers</u> (inherited from <u>TDADatasetOptions</u>)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
<u>InsertAllSetFields</u> (inherited from <u>TDADatasetOptions</u>)	Used to include all set dataset fields in the generated INSERT statement
<u>LocalMasterDetail</u> (inherited from	Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset

<u>TDDataSetOptions</u>)	and does not refer to the server.
<u>LongStrings</u> (inherited from <u>TDDataSetOptions</u>)	Used to represent string fields with the length that is greater than 255 as TStringField.
<u>MasterFieldsNullable</u> (inherited from <u>TDDataSetOptions</u>)	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).
<u>NumberRange</u> (inherited from <u>TDDataSetOptions</u>)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
<u>OIDAsInt</u>	Used to read OID fields as integer values and map these fields to TIntegerField.
<u>PrepareUpdateSQL</u>	Used to automatically prepare update queries before execution.
<u>QueryRecCount</u> (inherited from <u>TDDataSetOptions</u>)	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.
<u>QuoteNames</u> (inherited from <u>TDDataSetOptions</u>)	Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.
<u>RemoveOnRefresh</u> (inherited from <u>TDDataSetOptions</u>)	Used for a dataset to locally remove a record that can not be found on the server.
<u>RequiredFields</u> (inherited from <u>TDDataSetOptions</u>)	Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.
<u>ReturnParams</u> (inherited from <u>TDDataSetOptions</u>)	Used to return the new value of fields to dataset after insert or update.

SetFieldsReadOnly (inherited from TDADatasetOptions)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StrictUpdate (inherited from TDADatasetOptions)	Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar (inherited from TDADatasetOptions)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UnknownAsString	Used to map fields of unknown data types to TStringField (TWideStringField).
UnpreparedExecute	Used to apply simple executing to a SQL statement.
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.
UseParamTypes	Used to disable automatic detection of the parameters data types.

5.13.1.9.2 Properties

Properties of the **TPgDataSetOptions** class.

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

Public

Name	Description
AutoPrepare (inherited from TDADatasetOptions)	Used to execute automatic TCustomDADataset.Prepare on the query execution.

<u>CacheCalcFields</u> (inherited from <u>TDADatasetOptions</u>)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
<u>CompressBlobMode</u> (inherited from <u>TDADatasetOptions</u>)	Used to store values of the BLOB fields in compressed form.
<u>DefaultValues</u> (inherited from <u>TDADatasetOptions</u>)	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
<u>DetailDelay</u> (inherited from <u>TDADatasetOptions</u>)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
<u>FieldsOrigin</u> (inherited from <u>TDADatasetOptions</u>)	Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.
<u>FlatBuffers</u> (inherited from <u>TDADatasetOptions</u>)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
<u>InsertAllSetFields</u> (inherited from <u>TDADatasetOptions</u>)	Used to include all set dataset fields in the generated INSERT statement
<u>LocalMasterDetail</u> (inherited from <u>TDADatasetOptions</u>)	Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
<u>LongStrings</u> (inherited from <u>TDADatasetOptions</u>)	Used to represent string fields with the length that is greater than 255 as TStringField.
<u>MasterFieldsNullable</u> (inherited from <u>TDADatasetOptions</u>)	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).
<u>NumberRange</u> (inherited from	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate

<u>TDDataSetOptions</u>)	values.
<u>QueryRecCount</u> (inherited from <u>TDDataSetOptions</u>)	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.
<u>QuoteNames</u> (inherited from <u>TDDataSetOptions</u>)	Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.
<u>RemoveOnRefresh</u> (inherited from <u>TDDataSetOptions</u>)	Used for a dataset to locally remove a record that can not be found on the server.
<u>RequiredFields</u> (inherited from <u>TDDataSetOptions</u>)	Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.
<u>ReturnParams</u> (inherited from <u>TDDataSetOptions</u>)	Used to return the new value of fields to dataset after insert or update.
<u>SetFieldsReadOnly</u> (inherited from <u>TDDataSetOptions</u>)	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.
<u>StrictUpdate</u> (inherited from <u>TDDataSetOptions</u>)	Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.
<u>TrimFixedChar</u> (inherited from <u>TDDataSetOptions</u>)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
<u>UpdateAllFields</u> (inherited from <u>TDDataSetOptions</u>)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
<u>UpdateBatchSize</u> (inherited from <u>TDDataSetOptions</u>)	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

Name	Description
AutoDeleteBlob	Used to delete large objects from the database automatically when a record is deleted from the dataset.
CacheBlobs	Used to allocate local memory buffer to hold a copy of the large object content.
CursorWithHold	Used to open query in the FetchAll=False mode without transaction.
DeferredBlobRead	Used to fetch values of large objects when they are explicitly requested.
ExtendedFieldsInfo	Used to perform an additional query to get information about returned fields and the tables they belong to.
OIDAsInt	Used to read OID fields as integer values and map these fields to TIntegerField.
PrepareUpdateSQL	Used to automatically prepare update queries before execution.
UnknownAsString	Used to map fields of unknown data types to TStringField (TWideStringField).
UnpreparedExecute	Used to apply simple executing to a SQL statement.
UseParamTypes	Used to disable automatic detection of the parameters data types.

See Also

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

5.13.1.9.2.1 AutoDeleteBlob Property

Used to delete large objects from the database automatically when a record is deleted from the dataset.

Class

[TPgDataSetOptions](#)

Syntax

```
property AutoDeleteBlob: boolean default True;
```

Remarks

Use the AutoDeleteBlob property to delete large objects from the database automatically when a record is deleted from the dataset.

If True, the large objects are deleted from the database automatically when a record that holds OIDs of these large objects is deleted from the dataset.

5.13.1.9.2.2 CacheBlobs Property

Used to allocate local memory buffer to hold a copy of the large object content.

Class

[TPgDataSetOptions](#)

Syntax

```
property CacheBlobs: boolean default True;
```

Remarks

Use the CacheBlobs property to allocate local memory buffer to hold a copy of the large object content.

If True, local memory buffer is allocated to hold a copy of the large object content.

If False, value of large object is being read and written directly from/to database without memory buffer on the client. This can save memory on the client for very large values.

5.13.1.9.2.3 CursorWithHold Property

Used to open query in the FetchAll=False mode without transaction.

Class

[TPgDataSetOptions](#)

Syntax

```
property CursorWithHold: boolean default False;
```

Remarks

Use the CursorWithHold option to open query in the FetchAll=False mode without transaction. When this option is False (default), an active transaction is required to open a query in the FetchAll=False mode. If there is no active transaction, PgDAC opens additional internal connection and starts transaction on this connection.

When this option is True, PgDAC uses DECLARE CURSOR ... WITH HOLD statement to open the query. In this case no active transaction is required but this may take additional server resources.

5.13.1.9.2.4 DeferredBlobRead Property

Used to fetch values of large objects when they are explicitly requested.

Class

[TPgDataSetOptions](#)

Syntax

```
property DeferredBlobRead: boolean default False;
```

Remarks

Use the DeferredBlobRead property to fetch values of large objects when they are explicitly requested.

If True, values of large objects are only fetched when they are explicitly requested.

Otherwise all values of large objects are fetched when dataset is opened. This option has no sense when the CacheBlobs option is set to False.

5.13.1.9.2.5 ExtendedFieldsInfo Property

Used to perform an additional query to get information about returned fields and the tables they belong to.

Class

[TPgDataSetOptions](#)

Syntax

```
property ExtendedFieldsInfo: boolean default True;
```

Remarks

Use the ExtendedFieldsInfo property to perform an additional query to get information about

returned fields and the tables they belong to.

If True, an additional query is performed to get information about returned fields and the tables they belong to. This information includes NOT NULL attribute of the field, SEQUENCE linked to the field, and table name corresponding to the field. Table name information is needed to detect fields that belong to updating table, and set read-only attribute for all other fields returned by the query.

5.13.1.9.2.6 OIDAsInt Property

Used to read OID fields as integer values and map these fields to TIntegerField.

Class

[TPgDataSetOptions](#)

Syntax

```
property OIDAsInt: boolean default False;
```

Remarks

Use the OIDAsInt property to read OID fields as integer values and map these fields to TIntegerField.

By default dataset treats all fields with OID data type in the table (except main OID field of table created WITH OIDS option) as descriptors of large objects. PgDAC will try to read large objects with OIDs from query, and will fail if these OIDs are not large object descriptors.

Set these option to False to read OIDs as integer values and map OID fields to TIntegerField. Set this option to True to read OIDs as large objects descriptors, and then read the values of the corresponding large objects. When OIDAsInt is True, OID fields are mapped on TBlobField.

5.13.1.9.2.7 PrepareUpdateSQL Property

Used to automatically prepare update queries before execution.

Class

[TPgDataSetOptions](#)

Syntax

```
property PrepareUpdateSQL: boolean;
```

Remarks

If True, update queries are automatically prepared before executing.

5.13.1.9.2.8 UnknownAsString Property

Used to map fields of unknown data types to TStringField (TWideStringField).

Class

[TPgDataSetOptions](#)

Syntax

```
property UnknownAsString: boolean default False;
```

Remarks

Use the UnknownAsString to map fields of unknown data types to TStringField (TWideStringField).

If False, fields of unknown data types (including geometric types and composite type when the EnableGeometrics and EnableComposites options of TPgConnection are set to False) are mapped to TMemoField or TWideMemoField depending on the value of the UseUnicode option.

Memo is used because maximum length of values from such fields is unknown.

If True, fields of unknown data types are mapped to TStringField or TWideStringField depending on the value of the UseUnicode option. Size of fields is set to 8192. Values larger than this size are truncated.

5.13.1.9.2.9 UnpreparedExecute Property

Used to apply simple executing to a SQL statement.

Class

[TPgDataSetOptions](#)

Syntax

```
property UnpreparedExecute: boolean default False;
```

Remarks

If the UnpreparedExecute property is set to True, the simple execute is used for SQL statement. Statement is not prepared before execute. It allows to add multiple statements separated by semicolon to the SQL property.

5.13.1.9.2.10 UseParamTypes Property

Used to disable automatic detection of the parameters data types.

Class

[TPgDataSetOptions](#)

Syntax

```
property UseParamTypes: boolean default False;
```

Remarks

Set the UseParamTypes option to True to disable automatic detection of parameter types.

When this option is True, data types of parameters are set basing on the DataType property.

When this option is False, data types of the parameters are detected by server automatically.

5.13.1.10 TPgDataSource Class

TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.

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

Unit

[PgAccess](#)

Syntax

```
TPgDataSource = class(TCRDataSource);
```

Remarks

TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.

TPgDataSource inherits its functionality directly from the TDataSource component.

At design-time assign individual data-aware components' DataSource properties from their drop-down listboxes.

Inheritance Hierarchy

[TCRDataSource](#)

TPgDataSource

5.13.1.10.1 Members

[TPgDataSource](#) class overview.

5.13.1.11 TPgDateField Class

A class providing access to the PostgreSQL date fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgDateField = class(TCustomPgTimeStampField);
```

Remarks

The TPgDateField class provides access to the PostgreSQL date fields.

Inheritance Hierarchy

[TCustomPgTimeStampField](#)

TPgDateField

5.13.1.11.1 Members

[TPgDateField](#) class overview.

Properties

Name	Description
AsPgDate	Used to provide access to a TPgDate object.
AsPgTimeStamp (inherited from TCustomPgTimeStampField)	Used to provide access to a TPgTimeStamp object.

5.13.1.11.2 Properties

Properties of the **TPgDateField** class.

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

Public

Name	Description
AsPgDate	Used to provide access to a TPgDate object.
AsPgTimeStamp (inherited from TCustomPgTimeStampField)	Used to provide access to a TPgTimeStamp object.

See Also

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

5.13.1.11.2.1 AsPgDate Property

Used to provide access to a TPgDate object.

Class

[TPgDateField](#)

Syntax

```
property AsPgDate: TPgDate;
```

Remarks

Use the AsPgDate property to provide access to a TPgDate object you can use for manipulations with the date value.

5.13.1.12 TPgEncryptor Class

The class that performs encrypting and decrypting of data.

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

Unit

[PgAccess](#)

Syntax

```
TPgEncryptor = class(TCREncryptor);
```

Inheritance Hierarchy

[TCREncryptor](#)

TPgEncryptor

5.13.1.12.1 Members

[TPgEncryptor](#) class overview.

Properties

Name	Description
DataHeader (inherited from TCREncryptor)	Specifies whether the additional information is stored with the encrypted data.
EncryptionAlgorithm (inherited from TCREncryptor)	Specifies the algorithm of data encryption.
HashAlgorithm (inherited from TCREncryptor)	Specifies the algorithm of generating hash data.
InvalidHashAction (inherited from TCREncryptor)	Specifies the action to perform on data fetching when hash data is invalid.
Password (inherited from TCREncryptor)	Used to set a password that is used to generate a key for encryption.

Methods

Name	Description
SetKey (inherited from TCREncryptor)	Sets a key, using which data is encrypted.

5.13.1.13 TPgGeometricField Class

A class providing access to the PostgreSQL geometric fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgGeometricField = class(TField);
```

Remarks

The TPgGeometricField class provides access to the PostgreSQL geometric fields.

5.13.1.13.1 Members

[TPgGeometricField](#) class overview.

Properties

Name	Description
AsPgGeometric	Used to provide access to a TPgGeometric object.

5.13.1.13.2 Properties

Properties of the **TPgGeometricField** class.

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

Public

Name	Description
AsPgGeometric	Used to provide access to a TPgGeometric object.

See Also

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

5.13.1.13.2.1 AsPgGeometric Property

Used to provide access to a TPgGeometric object.

Class

[TPgGeometricField](#)

Syntax

```
property AsPgGeometric: TPgGeometric;
```

Remarks

Use the `AsPgGeometric` property to provide access to a `TPgGeometric` object you can use for manipulations with the date value.

5.13.1.14 `TPgIntervalField` Class

A class providing access to the PostgreSQL interval fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgIntervalField = class(TField);
```

Remarks

The `TPgIntervalFields` class provides access to the PostgreSQL interval fields.

5.13.1.14.1 Members

[TPgIntervalField](#) class overview.

Properties

Name	Description
AsPgInterval	Used to provide access to a <code>TPgInterval</code> object.

5.13.1.14.2 Properties

Properties of the `TPgIntervalField` class.

For a complete list of the `TPgIntervalField` class members, see the [TPgIntervalField Members](#) topic.

Public

Name	Description
AsPgInterval	Used to provide access to a <code>TPgInterval</code> object.

See Also

- [TPgIntervalField Class](#)

- [TPgIntervalField Class Members](#)

5.13.1.14.2.1 AsPgInterval Property

Used to provide access to a TPgInterval object.

Class

[TPgIntervalField](#)

Syntax

```
property AsPgInterval: TPgInterval;
```

Remarks

Use the AsPgInterval property to provide access to a TPgInterval object you can use for manipulations with the interval value.

5.13.1.15 TPgLargeObject Class

A class providing support of PostgreSQL large objects.

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

Unit

[PgAccess](#)

Syntax

```
TPgLargeObject = class(TPgSQLLargeObject);
```

Remarks

The TPgLargeObject class provides support of PostgreSQL large objects.

Inheritance Hierarchy

[TSharedObject](#)

[TBlob](#)

[TCompressedBlob](#)

[TPgSQLLargeObject](#)

TPgLargeObject

5.13.1.15.1 Members

[TPgLargeObject](#) class overview.

Properties

Name	Description
AsString (inherited from TBlob)	Used to manipulate BLOB value as string.
AsWideString (inherited from TBlob)	Used to manipulate BLOB value as Unicode string.
Cached (inherited from TPgSQLLargeObject)	Used to specify whether the content of large object should be stored in memory buffer on the client.
Compressed (inherited from TCompressedBlob)	Used to indicate if the Blob is compressed.
CompressedSize (inherited from TCompressedBlob)	Used to indicate compressed size of the Blob data.
Connection	Connection used to read and write value of large object.
IsUnicode (inherited from TBlob)	Gives choice of making TBlob store and process data in Unicode format or not.
OID (inherited from TPgSQLLargeObject)	Holds the OID of a large object.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size (inherited from TBlob)	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.

Assign (inherited from TBlob)	Sets BLOB value from another TBlob object.
Clear (inherited from TBlob)	Deletes the current value in TBlob object.
CloseObject (inherited from TPgSQLLargeObject)	Closes the large object that was previously opened by the OpenObject method.
CreateObject (inherited from TPgSQLLargeObject)	Creates a new large object database.
LoadFromFile (inherited from TBlob)	Loads the contents of a file into a TBlob object.
LoadFromStream (inherited from TBlob)	Copies the contents of a stream into the TBlob object.
OpenObject (inherited from TPgSQLLargeObject)	Opens the large object specified by the OID property.
Read (inherited from TBlob)	Acquires a raw sequence of bytes from the data stored in TBlob.
ReadBlob (inherited from TPgSQLLargeObject)	Reads the content of the large object from the database.
Release (inherited from TSharedObject)	Decrements the reference count.
SaveToFile (inherited from TBlob)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from TBlob)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from TBlob)	Sets new TBlob size and discards all data over it.
UnlinkObject (inherited from TPgSQLLargeObject)	Deletes the large object specified by the OID property from the database.

Write (inherited from TBlob)	Stores a raw sequence of bytes into a TBlob object.
WriteBlob (inherited from TPgSQLLargeObject)	Writes the content of the large object to the database.

5.13.1.15.2 Properties

Properties of the **TPgLargeObject** class.

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

Public

Name	Description
AsString (inherited from TBlob)	Used to manipulate BLOB value as string.
AsWideString (inherited from TBlob)	Used to manipulate BLOB value as Unicode string.
Cached (inherited from TPgSQLLargeObject)	Used to specify whether the content of large object should be stored in memory buffer on the client.
Compressed (inherited from TCompressedBlob)	Used to indicate if the Blob is compressed.
CompressedSize (inherited from TCompressedBlob)	Used to indicate compressed size of the Blob data.
Connection	Connection used to read and write value of large object.
IsUnicode (inherited from TBlob)	Gives choice of making TBlob store and process data in Unicode format or not.
OID (inherited from TPgSQLLargeObject)	Holds the OID of a large object.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

[Size](#) (inherited from [TBlob](#))

Used to learn the size of the TBlob value in bytes.

See Also

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

5.13.1.15.2.1 Connection Property

Connection used to read and write value of large object.

Class

[TPgLargeObject](#)

Syntax

```
property Connection: TPgConnection;
```

Remarks

If you call methods that read or write large object value from/to database, set the Connection property to an appropriate TPgConnection objects. Otherwise these methods will fail.

5.13.1.16 TPgLargeObjectField Class

A class providing access to the PostgreSQL large object fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgLargeObjectField = class(TBlobField);
```

Remarks

The TPgLargeObjectFields class provides access to the PostgreSQL large object fields.

5.13.1.16.1 Members

[TPgLargeObjectField](#) class overview.

Properties

Name	Description
AsLargeObject	Used to provide access to a TPgLargeObject object.

5.13.1.16.2 Properties

Properties of the **TPgLargeObjectField** class.

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

Public

Name	Description
AsLargeObject	Used to provide access to a TPgLargeObject object.

See Also

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

5.13.1.16.2.1 AsLargeObject Property

Used to provide access to a TPgLargeObject object.

Class

[TPgLargeObjectField](#)

Syntax

```
property AsLargeObject: TPgLargeObject;
```

Remarks

Use the AsLargeObject property to provide access to a TPgLargeObject object you can use for manipulations with the large object value.

5.13.1.17 TPgMetaData Class

A component for obtaining metainformation about database objects from the server. For a list of all members of this type, see [TPgMetaData](#) members.

Unit

[PgAccess](#)

Syntax

```
TPgMetaData = class(TDAMetaData);
```

Remarks

The TPgMetaData component is used to obtain metainformation from the server about objects in the database, such as tables, table columns, stored procedures, etc.

Inheritance Hierarchy

[TMemDataSet](#)

[TDAMetaData](#)

TPgMetaData

See Also

- [TCustomDADataset.Debug](#)
- [TCustomDASQL.Debug](#)
- [DBMonitor](#)

5.13.1.17.1 Members

[TPgMetaData](#) class overview.

Properties

Name	Description
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
Connection (inherited from TDAMetaData)	Used to specify a connection object to use to connect to a data store.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
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

<u>TMemDataSet</u>)	NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MetaDataKind</u> (inherited from <u>TDAMetaData</u>)	Used to specify which kind of metainformation to show.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.
<u>Restrictions</u> (inherited from <u>TDAMetaData</u>)	Used to provide one or more conditions restricting the list of objects to be described.
<u>UpdateRecordTypes</u> (inherited from <u>TMemDataSet</u>)	Used to indicate the update status for the current record when cached updates are enabled.
<u>UpdatesPending</u> (inherited from <u>TMemDataSet</u>)	Used to check the status of the cached updates buffer.

Methods

Name	Description
<u>ApplyRange</u> (inherited from <u>TMemDataSet</u>)	Applies a range to the dataset.
<u>ApplyUpdates</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Writes dataset's pending cached updates to a database.
<u>CancelRange</u> (inherited from <u>TMemDataSet</u>)	Removes any ranges currently in effect for a dataset.
<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.

DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
EditRangeEnd (inherited from TMemDataSet)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from TMemDataSet)	Enables changing the starting value for an existing range.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds (inherited from TDAMetaData)	Used to get values acceptable in the MetaDataKind property.
GetRestrictions (inherited from TDAMetaData)	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from TMemDataSet)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from TMemDataSet)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from TMemDataSet)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from TMemDataSet)	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

Name	Description
OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from TMemDataSet)	Occurs when a single update component can not handle the updates.

5.13.1.18 TPgParam Class

A class that is used to set the values of individual parameters passed with queries or stored procedures.

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

Unit

[PgAccess](#)

Syntax

```
TPgParam = class (TDAParam);
```

Remarks

Use the properties of `TPgParam` to set the value of a parameter. Objects that use parameters create `TPgParam` objects to represent these parameters. For example, `TPgParam` objects are used by `TPgSQL`, `TCustomPgDataSet`.

`TPgParam` shares many properties with `TField`, as both describe the value of a field in a dataset. However, a `TField` object has several properties to describe the field binding, and how the field is displayed, edited, or calculated that are not needed in a `TPgParam` object. Conversely, `TPgParam` includes properties that indicate how the field value is passed as a parameter.

Inheritance Hierarchy

[TDAParam](#)

TPgParam

See Also

- [TCustomPgDataSet](#)
- [TPgSQL](#)
- [TPgParams](#)

5.13.1.18.1 Members

[TPgParam](#) class overview.

Properties

Name	Description
AsBlob (inherited from TDAParam)	Used to set and read the value of the BLOB parameter as string.
AsBlobRef (inherited from TDAParam)	Used to set and read the value of the BLOB parameter as a <code>TBlob</code> object.
AsFloat (inherited from TDAParam)	Used to assign the value for a float field to a parameter.
AsInteger (inherited from TDAParam)	Used to assign the value for an integer field to the parameter.
AsLargeInt (inherited from TDAParam)	Used to assign the value for a <code>LargeInteger</code> field to the parameter.
AsMemo (inherited from TDAParam)	Used to assign the value for a memo field to the parameter.

AsMemoRef (inherited from TDAParam)	Used to set and read the value of the memo parameter as a TBlob object.
AsPgDate	Used to specify the parameter value when it represents the date type.
AsPgInterval	Used to specify the parameter value when it represents the INTERVAL type.
AsPgTime	Used to specify the parameter value when it represents the time type.
AsPgTimeStamp	Used to specify the parameter value when it represents the TimeStamp type.
AsSQLTimeStamp (inherited from TDAParam)	Used to specify the value of the parameter when it represents a SQL timestamp field.
AsString (inherited from TDAParam)	Used to assign the string value to the parameter.
AsWideString (inherited from TDAParam)	Used to assign the Unicode string value to the parameter.
DataType (inherited from TDAParam)	Indicates the data type of the parameter.
IsNull (inherited from TDAParam)	Used to indicate whether the value assigned to a parameter is NULL.
ParamType (inherited from TDAParam)	Used to indicate the type of use for a parameter.
Size (inherited from TDAParam)	Specifies the size of a string type parameter.
Value (inherited from TDAParam)	Used to represent the value of the parameter as Variant.

Methods

Name	Description
AssignField (inherited from TDAParam)	Assigns field name and field value to a param.

AssignFieldValue (inherited from TDAParam)	Assigns the specified field properties and value to a parameter.
LoadFromFile (inherited from TDAParam)	Places the content of a specified file into a TDAParam object.
LoadFromStream (inherited from TDAParam)	Places the content from a stream into a TDAParam object.
SetBlobData (inherited from TDAParam)	Overloaded. Writes the data from a specified buffer to BLOB.

5.13.1.18.2 Properties

Properties of the **TPgParam** class.

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

Public

Name	Description
AsBlob (inherited from TDAParam)	Used to set and read the value of the BLOB parameter as string.
AsBlobRef (inherited from TDAParam)	Used to set and read the value of the BLOB parameter as a TBlob object.
AsFloat (inherited from TDAParam)	Used to assign the value for a float field to a parameter.
AsInteger (inherited from TDAParam)	Used to assign the value for an integer field to the parameter.
AsLargeInt (inherited from TDAParam)	Used to assign the value for a LargeInteger field to the parameter.
AsMemo (inherited from TDAParam)	Used to assign the value for a memo field to the parameter.
AsMemoRef (inherited from TDAParam)	Used to set and read the value of the memo parameter as a TBlob object.
AsPgDate	Used to specify the parameter value when it represents the date type.
AsPgInterval	Used to specify the parameter value when it represents the INTERVAL type.

AsPgTime	Used to specify the parameter value when it represents the time type.
AsPgTimeStamp	Used to specify the parameter value when it represents the TimeStamp type.
AsSQLTimeStamp (inherited from TDAParam)	Used to specify the value of the parameter when it represents a SQL timestamp field.
AsString (inherited from TDAParam)	Used to assign the string value to the parameter.
AsWideString (inherited from TDAParam)	Used to assign the Unicode string value to the parameter.
IsNull (inherited from TDAParam)	Used to indicate whether the value assigned to a parameter is NULL.

Published

Name	Description
DataType (inherited from TDAParam)	Indicates the data type of the parameter.
ParamType (inherited from TDAParam)	Used to indicate the type of use for a parameter.
Size (inherited from TDAParam)	Specifies the size of a string type parameter.
Value (inherited from TDAParam)	Used to represent the value of the parameter as Variant.

See Also

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

5.13.1.18.2.1 AsPgDate Property

Used to specify the parameter value when it represents the date type.

Class

[TPgParam](#)

Syntax

```
property AsPgDate: TPgDate;
```

Remarks

Use the AsPgDate property to specify the parameter value when it represents the date type.

5.13.1.18.2.2 AsPgInterval Property

Used to specify the parameter value when it represents the INTERVAL type.

Class

[TPgParam](#)

Syntax

```
property AsPgInterval: TPgInterval;
```

Remarks

Use the AsPgInterval property to specify the parameter value when it represents the INTERVAL type.

5.13.1.18.2.3 AsPgTime Property

Used to specify the parameter value when it represents the time type.

Class

[TPgParam](#)

Syntax

```
property AsPgTime: TPgTime;
```

Remarks

Use the AsPgTime property to specify the parameter value when it represents the time type.

5.13.1.18.2.4 AsPgTimeStamp Property

Used to specify the parameter value when it represents the TimeStamp type.

Class

[TPgParam](#)

Syntax

```
property AsPgTimeStamp: TPgTimeStamp;
```

Remarks

Use the AsPgTimeStamp property to specify the parameter value when it represents the TimeStamp type.

5.13.1.19 TPgParams Class

Used to control TPgParam objects.

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

Unit

[PgAccess](#)

Syntax

```
TPgParams = class(TDAParams);
```

Remarks

Use TPgParams to manage a list of TPgParam objects for an object that uses field parameters. For example, TPgStoredProc objects and TPgQuery objects use TPgParams objects to create and access their parameters.

Inheritance Hierarchy

[TDAParams](#)

TPgParams

See Also

- [TPgParam](#)
- [TCustomDASQL.Params](#)
- [TCustomDADataset.Params](#)

5.13.1.19.1 Members

[TPgParams](#) class overview.

Properties

Name	Description
------	-------------

Items	Used to iterate through all field parameters.
-----------------------	---

Methods

Name	Description
FindParam (inherited from TDAParams)	Searches for a parameter with the specified name.
ParamByName (inherited from TDAParams)	Searches for a parameter with the specified name.

5.13.1.19.2 Properties

Properties of the **TPgParams** class.

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

Public

Name	Description
Items	Used to iterate through all field parameters.

See Also

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

5.13.1.19.2.1 Items Property(Indexer)

Used to iterate through all field parameters.

Class

[TPgParams](#)

Syntax

```
property Items[Index: integer]: TPgParam; default;
```

Parameters

Index

Holds the index in the range 0..Count - 1.

Remarks

Use the `Items` property to iterate through all field parameters. `Index` identifies the index in the range `0..Count - 1`. Items can refer to a particular parameter by its index, but the [TDAParams.ParamByName](#) method is preferred to avoid depending on the order of the parameters.

5.13.1.20 TPgQuery Class

A component for executing queries and operating record sets. It also provides flexible way to update data.

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

Unit

[PgAccess](#)

Syntax

```
TPgQuery = class (TCustomPgDataSet);
```

Remarks

TPgQuery is a direct descendant of the [TCustomPgDataSet](#) component. It publishes most of its inherited properties and events so that they can be manipulated at design-time.

Use TPgQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. TPgQuery provides automatic blocking of records, their checking before edit and refreshing after post. Set `SQL`, `SQLInsert`, `SQLDelete`, `SQLRefresh`, and `SQLUpdate` properties to define SQL statements for subsequent accesses to the database server. There is no restriction to their syntax, so any SQL statement is allowed. Usually you need to use `INSERT`, `DELETE`, and `UPDATE` statements but you also may use stored procedures in more diverse cases.

To modify records, you can specify `KeyFields`. If they are not specified, TPgQuery will retrieve primary keys for `UpdatingTable` from metadata. TPgQuery can automatically update only one table. Updating table is defined by the `UpdatingTable` property if this property is set.

Otherwise, the table a field of which is the first field in the field list in the `SELECT` clause is used as an updating table.

The `SQLInsert`, `SQLDelete`, `SQLUpdate`, `SQLRefresh` properties support automatic binding of parameters which have identical names to fields captions. To retrieve the value of a field as it was before the operation use the field name with the `'OLD_'` prefix. This is especially useful when doing field comparisons in the `WHERE` clause of the statement. Use the [TCustomDADataset.BeforeUpdateExecute](#) event to assign the value to additional parameters

and the [TCustomDADataset.AfterUpdateExecute](#) event to read them.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

[TCustomPgDataSet](#)

TPgQuery

See Also

- [Updating Data with PgDAC Dataset Components](#)
- [Master/Detail Relationships](#)
- [TPgStoredProc](#)
- [TPgTable](#)

5.13.1.20.1 Members

[TPgQuery](#) class overview.

Properties

Name	Description
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
Conditions (inherited from TCustomDADataset)	Used to add WHERE conditions to a query
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from TCustomPgDataSet)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
DataTypeMap (inherited from	Used to set data type mapping rules

<u>TCustomDADataset</u>)	
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TCustomPgDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchAll</u>	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.

<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u> (inherited from <u>TCustomPgDataSet</u>)	Returns OID of the record inserted by the last query for table with OIDs.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>LockMode</u>	Used to specify what kind of lock will be performed when editing a record.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>Options</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the behaviour of TCustomPgDataSet object.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomPgDataset</u>)	Contains parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataset</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataset</u>)	Indicates whether a range is applied to a dataset.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>SequenceMode</u> (inherited from <u>TCustomPgDataset</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLLock</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to perform a record lock.
<u>SQLRecCount</u> (inherited from	Used to specify the SQL statement that is used to get the record count

<u>TCustomDADataset</u>)	when opening a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.
<u>SQLUpdate</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying an update to a dataset.
<u>UniDirectional</u> (inherited from <u>TCustomDADataset</u>)	Used if an application does not need bidirectional access to records in the result set.
<u>UpdateObject</u> (inherited from <u>TCustomPgDataset</u>)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
<u>UpdateRecordTypes</u> (inherited from <u>TMemDataset</u>)	Used to indicate the update status for the current record when cached updates are enabled.
<u>UpdatesPending</u> (inherited from <u>TMemDataset</u>)	Used to check the status of the cached updates buffer.
<u>UpdatingTable</u>	Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Methods

Name	Description
<u>AddWhere</u> (inherited from <u>TCustomDADataset</u>)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
<u>ApplyRange</u> (inherited from <u>TMemDataset</u>)	Applies a range to the dataset.

ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from TCustomDADataset)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
CreateBlobStream (inherited from TCustomDADataset)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from TCustomPgDataSet)	Generates the stored procedure call.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
DeleteWhere (inherited from TCustomDADataset)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from TMemDataSet)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from TMemDataSet)	Enables changing the starting value for an existing range.
Execute (inherited from TCustomDADataset)	Overloaded. Executes a SQL statement on the server.

<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from <u>TCustomDADataset</u>)	Description is not available at the moment.
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u> (inherited from <u>TCustomPgDataset</u>)	Searches for and returns a parameter with the specified name.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.

<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetKeyFieldNames</u> (inherited from <u>TCustomDADataset</u>)	Provides a list of available key field names.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetPgCursor</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgCursor object for a field with known name.
<u>GetPgDate</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgDate object for a field with known name.
<u>GetPgInterval</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgInterval object for a field with known name.
<u>GetPgLargeObject</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgLargeObject object for a field with known name.
<u>GetPgRow</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgRow object for a field with known name.
<u>GetPgTime</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTime object for a field with known name.
<u>GetPgTimeStamp</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTimeStamp object for a field with known name.

<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADataset</u>)	Locks the current record.
<u>MacroByName</u> (inherited from <u>TCustomDADataset</u>)	Finds a Macro with the name passed in Name.
<u>OpenNext</u> (inherited from <u>TCustomPgDataSet</u>)	Opens the next REFCURSOR for stored procedure that returns more than one cursor.
<u>ParamByName</u> (inherited from <u>TCustomPgDataSet</u>)	Searches for and returns a parameter with the specified name.
<u>Prepare</u> (inherited from <u>TCustomDADataset</u>)	Allocates, opens, and parses cursor for a query.
<u>RefreshRecord</u> (inherited from <u>TCustomDADataset</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADataset</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.

SaveSQL (inherited from TCustomDADataset)	Saves the SQL property value to BaseSQL.
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 (inherited from TCustomDADataset)	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 (inherited from TCustomDADataset)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from TCustomDADataset)	Releases a record lock.
UnPrepare (inherited from TMemDataSet)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from TMemDataSet)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from TMemDataSet)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
------	-------------

<u>AfterExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs after a component has executed a query to database.
<u>AfterFetch</u> (inherited from <u>TCustomDADataset</u>)	Occurs after dataset finishes fetching data from server.
<u>AfterUpdateExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs after executing insert, delete, update, lock and refresh operations.
<u>BeforeFetch</u> (inherited from <u>TCustomDADataset</u>)	Occurs before dataset is going to fetch block of records from the server.
<u>BeforeUpdateExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs before executing insert, delete, update, lock, and refresh operations.
<u>OnUpdateError</u> (inherited from <u>TMemDataSet</u>)	Occurs when an exception is generated while cached updates are applied to a database.
<u>OnUpdateRecord</u> (inherited from <u>TMemDataSet</u>)	Occurs when a single update component can not handle the updates.

5.13.1.20.2 Properties

Properties of the **TPgQuery** class.

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

Public

Name	Description
<u>BaseSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.

<u>Conditions</u> (inherited from <u>TCustomDADataset</u>)	Used to add WHERE conditions to a query
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>Cursor</u> (inherited from <u>TCustomPgDataSet</u>)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
<u>DataTypeMap</u> (inherited from <u>TCustomDADataset</u>)	Used to set data type mapping rules
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TCustomPgDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from	Used to get or set the list of fields on which the recordset is sorted.

<u>TMemDataSet</u>)	
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u> (inherited from <u>TCustomPgDataSet</u>)	Returns OID of the record inserted by the last query for table with OIDs.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.

<u>Options</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the behaviour of TCustomPgDataSet object.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomPgDataSet</u>)	Contains parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>SequenceMode</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.

<u>SQLLock</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to perform a record lock.
<u>SQLRecCount</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.
<u>SQLUpdate</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying an update to a dataset.
<u>UniDirectional</u> (inherited from <u>TCustomDADataset</u>)	Used if an application does not need bidirectional access to records in the result set.
<u>UpdateObject</u> (inherited from <u>TCustomPgDataset</u>)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
<u>UpdateRecordTypes</u> (inherited from <u>TMemDataSet</u>)	Used to indicate the update status for the current record when cached updates are enabled.
<u>UpdatesPending</u> (inherited from <u>TMemDataSet</u>)	Used to check the status of the cached updates buffer.

Published

Name	Description
<u>FetchAll</u>	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>LockMode</u>	Used to specify what kind of lock will be performed when editing a record.

[UpdatingTable](#)

Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

See Also

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

5.13.1.20.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

[TPgQuery](#)

Syntax

```
property FetchAll: boolean;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

Opening a dataset in FetchAll = False mode requires an active transaction. When the FetchAll property is False, the first call to [TMemDataSet.Locate](#) and [TMemDataSet.LocateEx](#) methods may take a lot of time to retrieve additional records to the client side.

5.13.1.20.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

[TPgQuery](#)

Syntax

```
property LockMode: TLockMode;
```


Remarks

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is ImNone.

See Also

- [TPgStoredProc.LockMode](#)
- [TPgTable.LockMode](#)

5.13.1.20.2.3 UpdatingTable Property

Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Class

[TPgQuery](#)

Syntax

```
property UpdatingTable: string;
```

Remarks

Use the UpdatingTable property to specify which table in a query is assumed to be the target for the subsequent data-modification queries as a result of user incentive to insert, update or delete records.

This property is used on Insert, Update, Delete or RefreshRecord (see also [TCustomPgDataSet.Options](#)) if appropriate SQL (SQLInsert, SQLUpdate or SQLDelete) is not provided.

If UpdatingTable is not set then the first table used in a query is assumed to be the target.

Example

Below are two examples for the query, where:

1. the only allowed value for UpdatingTable property is 'Dept';
2. allowed values for UpdatingTable are 'Dept' and 'Emp'.

In the first case (or by default) editable field is ShipName, in the second

5.13.1.21 TPgSQL Class

A component for executing SQL statements and calling stored procedures on the database server.

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

Unit

[PgAccess](#)

Syntax

```
TPgSQL = class(TCustomDASQL);
```

Remarks

The TPgSQL component is a direct descendant of the [TCustomDASQL](#) class.

Use The TPgSQL component when a client application must execute SQL statement or the PL/SQL block, and call stored procedure on the database server. The SQL statement should not retrieve rows from the database.

Inheritance Hierarchy

[TCustomDASQL](#)

TPgSQL

See Also

- [TPgQuery](#)

5.13.1.21.1 Members

[TPgSQL](#) class overview.

Properties

Name	Description
ChangeCursor (inherited from TCustomDASQL)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
CommandTimeout	The time to wait for a statement to be executed.
Connection	Used to specify a connection object that will be used to connect to a data store.

<u>Debug</u> (inherited from <u>TCustomDASQL</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>FinalSQL</u> (inherited from <u>TCustomDASQL</u>)	Used to return a SQL statement with expanded macros.
<u>LastInsertOID</u>	Returns OID of the record inserted by the last query for table with OIDs.
<u>MacroCount</u> (inherited from <u>TCustomDASQL</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDASQL</u>)	Makes it possible to change SQL queries easily.
<u>ParamCheck</u> (inherited from <u>TCustomDASQL</u>)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
<u>ParamCount</u> (inherited from <u>TCustomDASQL</u>)	Indicates the number of parameters in the Params property.
<u>Params</u>	Contains parameters for a query's SQL statement.
<u>ParamValues</u> (inherited from <u>TCustomDASQL</u>)	Used to get or set the values of individual field parameters that are identified by name.
<u>Prepared</u> (inherited from <u>TCustomDASQL</u>)	Used to indicate whether a query is prepared for execution.
<u>RowsAffected</u> (inherited from <u>TCustomDASQL</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>SQL</u> (inherited from <u>TCustomDASQL</u>)	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.
<u>UnpreparedExecute</u>	Used to apply simple executing to a SQL statement.
<u>UseParamTypes</u>	Used to disable automatic detection of the parameters data types.

Methods

Name	Description
Execute (inherited from TCustomDASQL)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from TCustomDASQL)	Checks whether TCustomDASQL still executes a SQL statement.
FindMacro (inherited from TCustomDASQL)	Searches for a macro with the specified name.
FindParam	Searches for and returns a parameter with the specified name.
MacroByName (inherited from TCustomDASQL)	Finds a Macro with the name passed in Name.
ParamByName	Searches for and returns a parameter with the specified name.
Prepare (inherited from TCustomDASQL)	Allocates, opens, and parses cursor for a query.
UnPrepare (inherited from TCustomDASQL)	Frees the resources allocated for a previously prepared query on the server and client sides.
WaitExecuting (inherited from TCustomDASQL)	Waits until TCustomDASQL executes a SQL statement.

Events

Name	Description
AfterExecute (inherited from TCustomDASQL)	Occurs after a SQL statement has been executed.

5.13.1.21.2 Properties

Properties of the **TPgSQL** class.

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

Public

Name	Description
<u>ChangeCursor</u> (inherited from <u>TCustomDASQL</u>)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
<u>Debug</u> (inherited from <u>TCustomDASQL</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>FinalSQL</u> (inherited from <u>TCustomDASQL</u>)	Used to return a SQL statement with expanded macros.
<u>LastInsertOID</u>	Returns OID of the record inserted by the last query for table with OIDs.
<u>MacroCount</u> (inherited from <u>TCustomDASQL</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDASQL</u>)	Makes it possible to change SQL queries easily.
<u>ParamCheck</u> (inherited from <u>TCustomDASQL</u>)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
<u>ParamCount</u> (inherited from <u>TCustomDASQL</u>)	Indicates the number of parameters in the Params property.
<u>ParamValues</u> (inherited from <u>TCustomDASQL</u>)	Used to get or set the values of individual field parameters that are identified by name.
<u>Prepared</u> (inherited from <u>TCustomDASQL</u>)	Used to indicate whether a query is prepared for execution.
<u>RowsAffected</u> (inherited from <u>TCustomDASQL</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>SQL</u> (inherited from <u>TCustomDASQL</u>)	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

Published

Name	Description
CommandTimeout	The time to wait for a statement to be executed.
Connection	Used to specify a connection object that will be used to connect to a data store.
Params	Contains parameters for a query's SQL statement.
UnpreparedExecute	Used to apply simple executing to a SQL statement.
UseParamTypes	Used to disable automatic detection of the parameters data types.

See Also

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

5.13.1.21.2.1 CommandTimeout Property

The time to wait for a statement to be executed.

Class

[TPgSQL](#)

Syntax

```
property CommandTimeout: integer default 0;
```

5.13.1.21.2.2 Connection Property

Used to specify a connection object that will be used to connect to a data store.

Class

[TPgSQL](#)

Syntax

```
property Connection: TPgConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TPgConnection objects.

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

See Also

- [TPgConnection](#)

5.13.1.21.2.3 LastInsertOID Property

Returns OID of the record inserted by the last query for table with OIDs.

Class

[TPgSQL](#)

Syntax

```
property LastInsertOID: Int64;
```

Remarks

Use the LastInsertOID property to get OID of the record inserted by the last query for table with OIDs.

5.13.1.21.2.4 Params Property

Contains parameters for a query's SQL statement.

Class

[TPgSQL](#)

Syntax

```
property Params: TPgParams stored False;
```

Remarks

Contains parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information).

Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

5.13.1.21.2.5 UnpreparedExecute Property

Used to apply simple executing to a SQL statement.

Class

[TPgSQL](#)

Syntax

```
property UnpreparedExecute: boolean default False;
```

Remarks

If the UnpreparedExecute property is set to True, the simple execute is used for SQL statement. Statement is not prepared before execute. It allows to add multiple statements separated by semicolon to the SQL property.

5.13.1.21.2.6 UseParamTypes Property

Used to disable automatic detection of the parameters data types.

Class

[TPgSQL](#)

Syntax

```
property UseParamTypes: boolean default False;
```

Remarks

Set the UseParamTypes option to True to disable automatic detection of parameter types. When this option is True, data types of parameters are set basing on the DataType property. When this option is False, data types of the parameters are detected by server automatically.

5.13.1.21.3 Methods

Methods of the **TPgSQL** class.

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

Public

Name	Description
------	-------------

Execute (inherited from TCustomDASQL)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from TCustomDASQL)	Checks whether TCustomDASQL still executes a SQL statement.
FindMacro (inherited from TCustomDASQL)	Searches for a macro with the specified name.
FindParam	Searches for and returns a parameter with the specified name.
MacroByName (inherited from TCustomDASQL)	Finds a Macro with the name passed in Name.
ParamByName	Searches for and returns a parameter with the specified name.
Prepare (inherited from TCustomDASQL)	Allocates, opens, and parses cursor for a query.
UnPrepare (inherited from TCustomDASQL)	Frees the resources allocated for a previously prepared query on the server and client sides.
WaitExecuting (inherited from TCustomDASQL)	Waits until TCustomDASQL executes a SQL statement.

See Also

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

5.13.1.21.3.1 FindParam Method

Searches for and returns a parameter with the specified name.

Class

[TPgSQL](#)

Syntax

```
function FindParam(const value: string): TPgParam;
```

Parameters

Value

Holds the stored procedure name.

Return Value

the parameter, if a match was found. Nil otherwise.

Remarks

Call the FindParam method to find a parameter with the name passed in the Name argument. If a match was found, FindParam returns the parameter. Otherwise, it returns nil.

See Also

- [TPgParam](#)
- [ParamByName](#)

5.13.1.21.3.2 ParamByName Method

Searches for and returns a parameter with the specified name.

Class

[TPgSQL](#)

Syntax

```
function ParamByName(const value: string): TPgParam;
```

Parameters*Value*

Holds the parameter name.

Return Value

the parameter, if a match was found. Otherwise an exception is raised.

Remarks

Call the ParamByName method to find a parameter with the name passed in the Name argument.

If a match is found, ParamByName returns the parameter. Otherwise, an exception is raised.

See Also

- [TPgParam](#)
- [FindParam](#)

5.13.1.22 TPgStoredProc Class

A component for accessing and executing stored procedures and functions.

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

Unit

[PgAccess](#)

Syntax

```
TPgStoredProc = class(TCustomPgStoredProc);
```

Remarks

Use TPgStoredProc to access stored procedures on the database server.

You need only to define the StoredProcName property, and the SQL statement to call the stored procedure will be generated automatically.

Use the Execute method at runtime to generate request that instructs server to execute procedure and PrepareSQL to describe parameters at run time

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

[TCustomPgDataSet](#)

[TCustomPgStoredProc](#)

TPgStoredProc

See Also

- [TPgQuery](#)
- [TPgSQL](#)
- [Updating Data with PgDAC Dataset Components](#)

5.13.1.22.1 Members

[TPgStoredProc](#) class overview.

Properties

Name	Description
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from TCustomPgDataSet)	Used to enable or disable the use of cached updates for a dataset.

<u>TMemDataSet</u>)	
<u>CommandTimeout</u>	The time to wait for a statement to be executed.
<u>Conditions</u> (inherited from <u>TCustomDADataset</u>)	Used to add WHERE conditions to a query
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>Cursor</u> (inherited from <u>TCustomPgDataSet</u>)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
<u>DataTypeMap</u> (inherited from <u>TCustomDADataset</u>)	Used to set data type mapping rules
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TCustomPgDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchAll</u> (inherited from <u>TCustomPgDataSet</u>)	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.

<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u> (inherited from <u>TCustomPgDataSet</u>)	Returns OID of the record inserted by the last query for table with OIDs.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>LockMode</u>	Used to specify what kind of lock will be performed when editing a record.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.

<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>Options</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the behaviour of TCustomPgDataSet object.
<u>Overload</u> (inherited from <u>TCustomPgStoredProc</u>)	Used to specify the overloading number.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomPgDataSet</u>)	Contains parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.
<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

SequenceMode (inherited from TCustomPgDataSet)	Used to specify the methods used internally to generate a sequenced field.
SQL (inherited from TCustomDADataset)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used when applying a deletion to a record.
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.
StoredProcName	Used to specify the name of the stored procedure to call on the server.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomPgDataSet)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.

UpdateRecordTypes (inherited from TMemDataSet)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from TMemDataSet)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from TCustomDADataset)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from TMemDataSet)	Applies a range to the dataset.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from TCustomDADataset)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
CreateBlobStream (inherited from TCustomDADataset)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from TCustomPgDataSet)	Generates the stored procedure call.

<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADataset</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>EditRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Enables changing the ending value for an existing range.
<u>EditRangeStart</u> (inherited from <u>TMemDataSet</u>)	Enables changing the starting value for an existing range.
<u>ExecProc</u> (inherited from <u>TCustomPgStoredProc</u>)	Executes a SQL statement on the server.
<u>Execute</u> (inherited from <u>TCustomDADataset</u>)	Overloaded. Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADataset</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADataset</u>)	Used to learn whether TCustomDADataset is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADataset</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from	Description is not available at the moment.

<u>TCustomDADataset</u>)	
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u> (inherited from <u>TCustomPgDataset</u>)	Searches for and returns a parameter with the specified name.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetKeyFieldNames</u> (inherited from <u>TCustomDADataset</u>)	Provides a list of available key field names.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetPgCursor</u> (inherited from <u>TCustomPgDataset</u>)	Retrieves a TPgCursor object for a field with known name.
<u>GetPgDate</u> (inherited from	Retrieves a TPgDate object for a field with known name.

<u>TCustomPgDataSet</u>)	
<u>GetPgInterval</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgInterval object for a field with known name.
<u>GetPgLargeObject</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgLargeObject object for a field with known name.
<u>GetPgRow</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgRow object for a field with known name.
<u>GetPgTime</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTime object for a field with known name.
<u>GetPgTimeStamp</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADataset</u>)	Locks the current record.
<u>MacroByName</u> (inherited from <u>TCustomDADataset</u>)	Finds a Macro with the name passed in Name.
<u>OpenNext</u> (inherited from <u>TCustomPgDataSet</u>)	Opens the next REFCURSOR for stored procedure that returns more than one cursor.

<u>ParamByName</u> (inherited from <u>TCustomPgDataSet</u>)	Searches for and returns a parameter with the specified name.
<u>Prepare</u> (inherited from <u>TCustomDADDataSet</u>)	Allocates, opens, and parses cursor for a query.
<u>PrepareSQL</u> (inherited from <u>TCustomPgStoredProc</u>)	Describes the parameters of a stored procedure.
<u>RefreshRecord</u> (inherited from <u>TCustomDADDataSet</u>)	Actualizes field values for the current record.
<u>RestoreSQL</u> (inherited from <u>TCustomDADDataSet</u>)	Restores the SQL property modified by AddWhere and SetOrderBy.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>SaveSQL</u> (inherited from <u>TCustomDADDataSet</u>)	Saves the SQL property value to BaseSQL.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SetOrderBy</u> (inherited from <u>TCustomDADDataSet</u>)	Builds an ORDER BY clause of a SELECT statement.
<u>SetRange</u> (inherited from <u>TMemDataSet</u>)	Sets the starting and ending values of a range, and applies it.
<u>SetRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Indicates that subsequent assignments to field values specify the end of the range of rows to

	include in the dataset.
SetRangeStart (inherited from TMemDataSet)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from TCustomDADataset)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from TCustomDADataset)	Releases a record lock.
UnPrepare (inherited from TMemDataSet)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from TMemDataSet)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from TMemDataSet)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from TCustomDADataset)	Occurs after a component has executed a query to database.
AfterFetch (inherited from TCustomDADataset)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from TCustomDADataset)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from TCustomDADataset)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from TCustomDADataset)	Occurs before executing insert, delete, update, lock, and refresh operations.

OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from TMemDataSet)	Occurs when a single update component can not handle the updates.

5.13.1.22.2 Properties

Properties of the **TPgStoredProc** class.

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

Public

Name	Description
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
Conditions (inherited from TCustomDADataset)	Used to add WHERE conditions to a query
Connection (inherited from TCustomDADataset)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from TCustomPgDataSet)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
DataTypeMap (inherited from TCustomDADataset)	Used to set data type mapping rules
Debug (inherited from TCustomDADataset)	Used to display executing statement, all its parameters' values, and the type of parameters.

<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TCustomPgDataset</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchAll</u> (inherited from <u>TCustomPgDataset</u>)	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataset</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataset</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataset</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.

<u>LastInsertOID</u> (inherited from <u>TCustomPgDataSet</u>)	Returns OID of the record inserted by the last query for table with OIDs.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>Options</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the behaviour of TCustomPgDataSet object.
<u>Overload</u> (inherited from <u>TCustomPgStoredProc</u>)	Used to specify the overloading number.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomPgDataSet</u>)	Contains parameters for a query's SQL statement.

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.
ReadOnly (inherited from TCustomDADataset)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions (inherited from TCustomDADataset)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from TCustomDADataset)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SequenceMode (inherited from TCustomPgDataSet)	Used to specify the methods used internally to generate a sequenced field.
SQL (inherited from TCustomDADataset)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used when applying a deletion to a record.
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 TCustomPgDataSet)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
UpdateRecordTypes (inherited from TMemDataSet)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from TMemDataSet)	Used to check the status of the cached updates buffer.

Published

Name	Description
CommandTimeout	The time to wait for a statement to be executed.
LockMode	Used to specify what kind of lock will be performed when editing a record.
StoredProcName	Used to specify the name of the stored procedure to call on the server.

See Also

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

5.13.1.22.2.1 CommandTimeout Property

The time to wait for a statement to be executed.

Class

[TPgStoredProc](#)

Syntax

```
property CommandTimeout: integer;
```

5.13.1.22.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

[TPgStoredProc](#)

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 ImNone.

5.13.1.22.2.3 StoredProcName Property

Used to specify the name of the stored procedure to call on the server.

Class

[TPgStoredProc](#)

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.

5.13.1.23 TPgTable Class

A component for retrieving and updating data in a single table without writing SQL statements.

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

Unit

[PgAccess](#)

Syntax

```
TPgTable = class(TCustomPgTable);
```

Remarks

The TPgTable component allows retrieving and updating data in a single table without writing SQL statements. Use TPgTable to access data in a table . Use the TableName property to specify table name. TPgTable uses the KeyFields property to build SQL statements for updating table data. KeyFields is a string containing a semicolon-delimited list of the field names.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

[TCustomPgDataSet](#)

[TCustomPgTable](#)

TPgTable

See Also

- [Updating Data with PgDAC Dataset Components](#)
- [Master/Detail Relationships](#)
- [TCustomPgDataSet](#)
- [TPgQuery](#)
- [TCustomPgTable](#)

5.13.1.23.1 Members

[TPgTable](#) class overview.

Properties

Name	Description
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.

<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>Conditions</u> (inherited from <u>TCustomDADataset</u>)	Used to add WHERE conditions to a query
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>Cursor</u> (inherited from <u>TCustomPgDataSet</u>)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
<u>DataTypeMap</u> (inherited from <u>TCustomDADataset</u>)	Used to set data type mapping rules
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TCustomPgDataSet</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchAll</u>	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from	Used to change the WHERE clause of SELECT statement and reopen a query.

<u>TCustomDADataset</u>)	
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataSet</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u> (inherited from <u>TCustomPgDataSet</u>)	Returns OID of the record inserted by the last query for table with OIDs.
<u>Limit</u> (inherited from <u>TCustomPgTable</u>)	Used to set the number of rows retrieved from the query.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>LockMode</u>	Used to specify what kind of lock will be performed when editing a record.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.

<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>Offset</u> (inherited from <u>TCustomPgTable</u>)	Used to allow retrieving data from the server starting from the specified row.
<u>Options</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the behaviour of TCustomPgDataSet object.
<u>OrderFields</u>	Used to build ORDER BY clause of SQL statements.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomPgDataSet</u>)	Contains parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.
<u>ReadOnly</u> (inherited from <u>TCustomDADataset</u>)	Used to prevent users from updating, inserting, or deleting data in the dataset.
<u>RefreshOptions</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate when the editing record is refreshed.

<u>RowsAffected</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
<u>SequenceMode</u> (inherited from <u>TCustomPgDataset</u>)	Used to specify the methods used internally to generate a sequenced field.
<u>SQL</u> (inherited from <u>TCustomDADataset</u>)	Used to provide a SQL statement that a query component executes when its Open method is called.
<u>SQLDelete</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying a deletion to a record.
<u>SQLInsert</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
<u>SQLLock</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to perform a record lock.
<u>SQLRecCount</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
<u>SQLRefresh</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used to refresh current record by calling the <u>TCustomDADataset.RefreshRecord</u> procedure.
<u>SQLUpdate</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a SQL statement that will be used when applying an update to a dataset.
<u>TableName</u>	Used to specify the name of the database table this component encapsulates.
<u>UniDirectional</u> (inherited from <u>TCustomDADataset</u>)	Used if an application does not need bidirectional access to records in the result set.

UpdateObject (inherited from TCustomPgDataSet)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
UpdateRecordTypes (inherited from TMemDataSet)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from TMemDataSet)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from TCustomDADataset)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from TMemDataSet)	Applies a range to the dataset.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from TCustomDADataset)	Breaks execution of the SQL statement on the server.
CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
CreateBlobStream (inherited from TCustomDADataset)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.

<u>CreateProcCall</u> (inherited from <u>TCustomPgDataSet</u>)	Generates the stored procedure call.
<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>DeleteWhere</u> (inherited from <u>TCustomDADDataSet</u>)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
<u>EditRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Enables changing the ending value for an existing range.
<u>EditRangeStart</u> (inherited from <u>TMemDataSet</u>)	Enables changing the starting value for an existing range.
<u>Execute</u> (inherited from <u>TCustomDADDataSet</u>)	Overloaded. Executes a SQL statement on the server.
<u>Executing</u> (inherited from <u>TCustomDADDataSet</u>)	Indicates whether SQL statement is still being executed.
<u>Fetches</u> (inherited from <u>TCustomDADDataSet</u>)	Used to learn whether TCustomDADDataSet has already fetched all rows.
<u>Fetching</u> (inherited from <u>TCustomDADDataSet</u>)	Used to learn whether TCustomDADDataSet is still fetching rows.
<u>FetchingAll</u> (inherited from <u>TCustomDADDataSet</u>)	Used to learn whether TCustomDADDataSet is fetching all rows to the end.
<u>FindKey</u> (inherited from <u>TCustomDADDataSet</u>)	Searches for a record which contains specified field values.
<u>FindMacro</u> (inherited from	Description is not available at the moment.

<u>TCustomDADataset</u>)	
<u>FindNearest</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>FindParam</u> (inherited from <u>TCustomPgDataset</u>)	Searches for and returns a parameter with the specified name.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>GetDataType</u> (inherited from <u>TCustomDADataset</u>)	Returns internal field types defined in the MemData and accompanying modules.
<u>GetFieldObject</u> (inherited from <u>TCustomDADataset</u>)	Returns a multireference shared object from field.
<u>GetFieldPrecision</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the precision of a number field.
<u>GetFieldScale</u> (inherited from <u>TCustomDADataset</u>)	Retrieves the scale of a number field.
<u>GetKeyFieldNames</u> (inherited from <u>TCustomDADataset</u>)	Provides a list of available key field names.
<u>GetOrderBy</u> (inherited from <u>TCustomDADataset</u>)	Retrieves an ORDER BY clause from a SQL statement.
<u>GetPgCursor</u> (inherited from <u>TCustomPgDataset</u>)	Retrieves a TPgCursor object for a field with known name.
<u>GetPgDate</u> (inherited from	Retrieves a TPgDate object for a field with known name.

<u>TCustomPgDataSet</u>)	
<u>GetPgInterval</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgInterval object for a field with known name.
<u>GetPgLargeObject</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgLargeObject object for a field with known name.
<u>GetPgRow</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgRow object for a field with known name.
<u>GetPgTime</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTime object for a field with known name.
<u>GetPgTimeStamp</u> (inherited from <u>TCustomPgDataSet</u>)	Retrieves a TPgTimeStamp object for a field with known name.
<u>GotoCurrent</u> (inherited from <u>TCustomDADataset</u>)	Sets the current record in this dataset similar to the current record in another dataset.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Lock</u> (inherited from <u>TCustomDADataset</u>)	Locks the current record.
<u>MacroByName</u> (inherited from <u>TCustomDADataset</u>)	Finds a Macro with the name passed in Name.
<u>OpenNext</u> (inherited from <u>TCustomPgDataSet</u>)	Opens the next REFCURSOR for stored procedure that returns more than one cursor.

ParamByName (inherited from TCustomPgDataSet)	Searches for and returns a parameter with the specified name.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
RefreshRecord (inherited from TCustomDADataset)	Actualizes field values for the current record.
RestoreSQL (inherited from TCustomDADataset)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from TMemDataSet)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from TCustomDADataset)	Saves the SQL property value to BaseSQL.
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 (inherited from TCustomDADataset)	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

<u>TMemDataSet</u>)	include in the dataset.
<u>SQLSaved</u> (inherited from <u>TCustomDADataset</u>)	Determines if the <u>SQL</u> property value was saved to the <u>BaseSQL</u> property.
<u>UnLock</u> (inherited from <u>TCustomDADataset</u>)	Releases a record lock.
<u>UnPrepare</u> (inherited from <u>TMemDataSet</u>)	Frees the resources allocated for a previously prepared query on the server and client sides.
<u>UpdateResult</u> (inherited from <u>TMemDataSet</u>)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
<u>UpdateStatus</u> (inherited from <u>TMemDataSet</u>)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
<u>AfterExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs after a component has executed a query to database.
<u>AfterFetch</u> (inherited from <u>TCustomDADataset</u>)	Occurs after dataset finishes fetching data from server.
<u>AfterUpdateExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs after executing insert, delete, update, lock and refresh operations.
<u>BeforeFetch</u> (inherited from <u>TCustomDADataset</u>)	Occurs before dataset is going to fetch block of records from the server.
<u>BeforeUpdateExecute</u> (inherited from <u>TCustomDADataset</u>)	Occurs before executing insert, delete, update, lock, and refresh operations.
<u>OnUpdateError</u> (inherited from	Occurs when an exception is generated while cached updates are

<u>TMemDataSet</u>)	applied to a database.
<u>OnUpdateRecord</u> (inherited from <u>TMemDataSet</u>)	Occurs when a single update component can not handle the updates.

5.13.1.23.2 Properties

Properties of the **TPgTable** class.

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

Public

Name	Description
<u>BaseSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
<u>CachedUpdates</u> (inherited from <u>TMemDataSet</u>)	Used to enable or disable the use of cached updates for a dataset.
<u>Conditions</u> (inherited from <u>TCustomDADataset</u>)	Used to add WHERE conditions to a query
<u>Connection</u> (inherited from <u>TCustomDADataset</u>)	Used to specify a connection object to use to connect to a data store.
<u>Cursor</u> (inherited from <u>TCustomPgDataSet</u>)	Used to fetch data from the REFCURSOR parameter or REFCURSOR field.
<u>DataTypeMap</u> (inherited from <u>TCustomDADataset</u>)	Used to set data type mapping rules
<u>Debug</u> (inherited from <u>TCustomDADataset</u>)	Used to display executing statement, all its parameters' values, and the type of parameters.
<u>DetailFields</u> (inherited from	Used to specify the fields that correspond to the foreign key fields from MasterFields when building

<u>TCustomDADataset</u>)	master/detail relationship.
<u>Disconnected</u> (inherited from <u>TCustomDADataset</u>)	Used to keep dataset opened after connection is closed.
<u>DMLRefresh</u> (inherited from <u>TCustomPgDataset</u>)	Used to refresh record by RETURNING clause when insert or update is performed.
<u>FetchRows</u> (inherited from <u>TCustomDADataset</u>)	Used to define the number of rows to be transferred across the network at the same time.
<u>FilterSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to change the WHERE clause of SELECT statement and reopen a query.
<u>FinalSQL</u> (inherited from <u>TCustomDADataset</u>)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<u>IndexFieldNames</u> (inherited from <u>TMemDataset</u>)	Used to get or set the list of fields on which the recordset is sorted.
<u>IsQuery</u> (inherited from <u>TCustomDADataset</u>)	Used to check whether SQL statement returns rows.
<u>KeyExclusive</u> (inherited from <u>TMemDataset</u>)	Specifies the upper and lower boundaries for a range.
<u>KeyFields</u> (inherited from <u>TCustomDADataset</u>)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
<u>KeySequence</u> (inherited from <u>TCustomPgDataset</u>)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
<u>LastInsertOID</u> (inherited from <u>TCustomPgDataset</u>)	Returns OID of the record inserted by the last query for table with OIDs.

<u>Limit</u> (inherited from <u>TCustomPgTable</u>)	Used to set the number of rows retrieved from the query.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>MacroCount</u> (inherited from <u>TCustomDADataset</u>)	Used to get the number of macros associated with the Macros property.
<u>Macros</u> (inherited from <u>TCustomDADataset</u>)	Makes it possible to change SQL queries easily.
<u>MasterFields</u> (inherited from <u>TCustomDADataset</u>)	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.
<u>MasterSource</u> (inherited from <u>TCustomDADataset</u>)	Used to specify the data source component which binds current dataset to the master one.
<u>Offset</u> (inherited from <u>TCustomPgTable</u>)	Used to allow retrieving data from the server starting from the specified row.
<u>Options</u> (inherited from <u>TCustomPgDataSet</u>)	Used to specify the behaviour of TCustomPgDataSet object.
<u>ParamCheck</u> (inherited from <u>TCustomDADataset</u>)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
<u>ParamCount</u> (inherited from <u>TCustomDADataset</u>)	Used to indicate how many parameters are there in the Params property.
<u>Params</u> (inherited from <u>TCustomPgDataSet</u>)	Contains parameters for a query's SQL statement.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.

Ranged (inherited from TMemDataSet)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from TCustomDADataset)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions (inherited from TCustomDADataset)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from TCustomDADataset)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SequenceMode (inherited from TCustomPgDataSet)	Used to specify the methods used internally to generate a sequenced field.
SQL (inherited from TCustomDADataset)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used when applying a deletion to a record.
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

<u>TCustomDADataset</u>)	to a dataset.
<u>UniDirectional</u> (inherited from <u>TCustomDADataset</u>)	Used if an application does not need bidirectional access to records in the result set.
<u>UpdateObject</u> (inherited from <u>TCustomPgDataset</u>)	Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.
<u>UpdateRecordTypes</u> (inherited from <u>TMemDataset</u>)	Used to indicate the update status for the current record when cached updates are enabled.
<u>UpdatesPending</u> (inherited from <u>TMemDataset</u>)	Used to check the status of the cached updates buffer.

Published

Name	Description
<u>FetchAll</u>	Defines whether to request all records of the query from database server when the dataset is being opened.
<u>LockMode</u>	Used to specify what kind of lock will be performed when editing a record.
<u>OrderFields</u>	Used to build ORDER BY clause of SQL statements.
<u>TableName</u>	Used to specify the name of the database table this component encapsulates.

See Also

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

5.13.1.23.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

[TPgTable](#)

Syntax

```
property FetchAll: boolean;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

Opening a dataset in FetchAll = False mode requires an active transaction. When the FetchAll property is False, the first call to [TMemDataSet.Locate](#) and [TMemDataSet.LocateEx](#) methods may take a lot of time to retrieve additional records to the client side.

5.13.1.23.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

[TPgTable](#)

Syntax

```
property LockMode: TLockMode;
```

Remarks

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is ImNone.

See Also

- [TPgStoredProc.LockMode](#)
- [TPgQuery.LockMode](#)

5.13.1.23.2.3 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class

[TPgTable](#)

Syntax

```
property OrderFields: string;
```

Remarks

TPgTable uses the OrderFields property to build ORDER BY clause of SQL statements. To set several field names to this property separate them with commas.

TPgTable is reopened when OrderFields is being changed.

See Also

- [TPgTable](#)

5.13.1.23.2.4 TableName Property

Used to specify the name of the database table this component encapsulates.

Class

[TPgTable](#)

Syntax

```
property TableName: string;
```

Remarks

Use the TableName property to specify the name of the database table this component encapsulates. If [TCustomDADDataSet.Connection](#) is assigned at design time, select a valid table name from the TableName drop-down list in Object Inspector.

5.13.1.24 TPgTimeField Class

A class providing access to the PostgreSQL time fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgTimeField = class(TCustomPgTimeStampField);
```

Remarks

The TPgTimeFields class provides access to the PostgreSQL time fields.

Inheritance Hierarchy

[TCustomPgTimeStampField](#)

TPgTimeField

5.13.1.24.1 Members

[TPgTimeField](#) class overview.

Properties

Name	Description
AsPgTime	Used to provide access to a TPgTime object.
AsPgTimeStamp (inherited from TCustomPgTimeStampField)	Used to provide access to a TPgTimeStamp object.

5.13.1.24.2 Properties

Properties of the **TPgTimeField** class.

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

Public

Name	Description
AsPgTime	Used to provide access to a TPgTime object.
AsPgTimeStamp (inherited from TCustomPgTimeStampField)	Used to provide access to a TPgTimeStamp object.

See Also

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

5.13.1.24.2.1 AsPgTime Property

Used to provide access to a TPgTime object.

Class

[TPgTimeField](#)

Syntax

```
property ASPgTime: TPgTime;
```

Remarks

Use the ASPgTime property to provide access to a TPgTime object you can use for manipulations with the time value.

5.13.1.25 TPgTimeStampField Class

A class providing access to the PostgreSQL timestamp fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgTimeStampField = class(TCustomPgTimeStampField);
```

Remarks

TPgTimeStampField provides access to PostgreSQL timestamp fields.

Inheritance Hierarchy

[TCustomPgTimeStampField](#)

TPgTimeStampField

5.13.1.25.1 Members

[TPgTimeStampField](#) class overview.

Properties

Name	Description
AsPgTimeStamp	Used to provide access to a TPgTimeStamp object.

5.13.1.25.2 Properties

Properties of the **TPgTimeStampField** class.

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

Public

Name	Description
AsPgTimeStamp	Used to provide access to a TPgTimeStamp object.

See Also

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

5.13.1.25.2.1 AsPgTimeStamp Property

Used to provide access to a TPgTimeStamp object.

Class

[TPgTimeStampField](#)

Syntax

```
property ASPgTimeStamp: TPgTimeStamp;
```

Remarks

Use the AsTimeStamp property to get access to a TPgTimeStamp object which you can use for manipulations with the timestamp value.

5.13.1.26 TPgTransaction Class

A component for managing transactions in an application.

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

Unit

[PgAccess](#)

Syntax

```
TPgTransaction = class(TDATransaction);
```

Remarks

The TPgTransaction component is used to provide discrete transaction control over connection. This component is used internally by PgDAC. For user, it is recommended to use methods of TPgConnection instead (such as StartTransaction, Commit, Rollback) for transaction control.

Inheritance Hierarchy

[TDATransaction](#)

TPgTransaction

See Also

- [TCustomDAConnection.StartTransaction](#)
- [TCustomDAConnection.Commit](#)
- [TCustomDAConnection.Rollback](#)

5.13.1.26.1 Members

[TPgTransaction](#) class overview.

Properties

Name	Description
Active (inherited from TDATransaction)	Used to determine if the transaction is active.
DefaultCloseAction (inherited from TDATransaction)	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.
IsolationLevel	Used to specify how the transactions containing database modifications are handled.

Methods

Name	Description
Commit (inherited from TDATransaction)	Commits the current transaction.
Rollback (inherited from TDATransaction)	Discards all modifications of data associated with the current transaction and ends the transaction.
StartTransaction (inherited from TDATransaction)	Begins a new transaction.

Events

Name	Description
OnCommit (inherited from TDATransaction)	Occurs after the transaction has been successfully committed.
OnCommitRetaining (inherited from TDATransaction)	Occurs after CommitRetaining has been executed.
OnError (inherited from TDATransaction)	Used to process errors that occur during executing a transaction.
OnRollback (inherited from TDATransaction)	Occurs after the transaction has been successfully rolled back.
OnRollbackRetaining (inherited from TDATransaction)	Occurs after RollbackRetaining has been executed.

5.13.1.26.2 Properties

Properties of the **TPgTransaction** class.

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

Public

Name	Description
Active (inherited from TDATransaction)	Used to determine if the transaction is active.

DefaultCloseAction (inherited from TDATransaction)	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.
IsolationLevel	Used to specify how the transactions containing database modifications are handled.

See Also

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

5.13.1.26.2.1 IsolationLevel Property

Used to specify how the transactions containing database modifications are handled.

Class

[TPgTransaction](#)

Syntax

```
property IsolationLevel: TCRIsolationLevel;
```

Remarks

Use the IsolationLevel property to specify how the transactions containing database modifications are handled.

5.13.1.27 TPgUpdateSQL Class

Lets you tune update operations for the DataSet component.

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

Unit

[PgAccess](#)

Syntax

```
TPgUpdateSQL = class(TCustomDAUpdateSQL);
```

Inheritance Hierarchy

[TCustomDAUpdateSQL](#)

TPgUpdateSQL

5.13.1.27.1 Members

[TPgUpdateSQL](#) class overview.

Properties

Name	Description
DataSet (inherited from TCustomDAUpdateSQL)	Used to hold a reference to the TCustomDADataSet object that is being updated.
DeleteObject (inherited from TCustomDAUpdateSQL)	Provides ability to perform advanced adjustment of the delete operations.
DeleteSQL (inherited from TCustomDAUpdateSQL)	Used when deleting a record.
InsertObject (inherited from TCustomDAUpdateSQL)	Provides ability to perform advanced adjustment of insert operations.
InsertSQL (inherited from TCustomDAUpdateSQL)	Used when inserting a record.
LockObject (inherited from TCustomDAUpdateSQL)	Provides ability to perform advanced adjustment of lock operations.
LockSQL (inherited from TCustomDAUpdateSQL)	Used to lock the current record.
ModifyObject (inherited from TCustomDAUpdateSQL)	Provides ability to perform advanced adjustment of modify operations.
ModifySQL (inherited from TCustomDAUpdateSQL)	Used when updating a record.

RefreshObject (inherited from TCustomDAUpdateSQL)	Provides ability to perform advanced adjustment of refresh operations.
RefreshSQL (inherited from TCustomDAUpdateSQL)	Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataset.RefreshRecord procedure.
SQL (inherited from TCustomDAUpdateSQL)	Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Methods

Name	Description
Apply (inherited from TCustomDAUpdateSQL)	Sets parameters for a SQL statement and executes it to update a record.
ExecSQL (inherited from TCustomDAUpdateSQL)	Executes a SQL statement.

5.13.2 Types

Types in the **PgAccess** unit.

Types

Name	Description
TPgNoticeEvent	This type is used for the TPgConnection.OnNotice event.
TPgNotificationEvent	This type is used for the TPgConnection.OnNotification event.

5.13.2.1 TPgNoticeEvent Procedure Reference

This type is used for the [TPgConnection.OnNotice](#) event.

Unit

[PgAccess](#)

Syntax

```
TPgNoticeEvent = procedure (Sender: TObject; Errors: TPgErrors) of  
object;
```

Parameters

Sender

An object that raised the event.

Errors

Holds a collection of TPgError objects containing notices.

5.13.2.2 TPgNotificationEvent Procedure Reference

This type is used for the [TPgConnection.OnNotification](#) event.

Unit

[PgAccess](#)

Syntax

```
TPgNotificationEvent = procedure (Sender: TObject; const  
EventName: string; PID: integer; const EventMessage: string) of  
object;
```

Parameters

Sender

An object that raised the event.

EventName

Holds the event name.

PID

Holds the process ID of a connection that has sent this notification.

EventMessage

Holds the event message.

Remarks

EventMessage parameter supported in PostgreSQL 9.0 or higher. For previous PostgreSQL versions EventMessage will be always empty.

5.13.3 Enumerations

Enumerations in the **PgAccess** unit.

Enumerations

Name	Description
TPgIsolationLevel	Specifies the way the transactions containing database modifications are handled.

5.13.3.1 TPgIsolationLevel Enumeration

Specifies the way the transactions containing database modifications are handled.

Unit

[PgAccess](#)

Syntax

```
TPgIsolationLevel = (pilReadCommitted, pilSerializable);
```

Values

Value	Meaning
<code>pilReadCommitted</code>	If the transaction contains DML that requires row locks held by another transaction, then the DML statement waits until the row locks are released. The default PostgreSQL behavior.
<code>pilSerializable</code>	Specifies serializable transaction isolation mode as defined in the SQL92 standard. If a serializable transaction contains data manipulation language (DML) that attempts to update any resource that may have been updated in a transaction uncommitted at the start of the serializable transaction, then the DML statement fails.

5.13.4 Constants

Constants in the **PgAccess** unit.

Constants

Name	Description
PgDACVersion	The version of PostgreSQL Data Access Components.

5.13.4.1 PgDACVersion Constant

The version of PostgreSQL Data Access Components.

Unit

[PgAccess](#)

Syntax

```
PgDACVersion = '5.2.6';
```

5.14 PgAlerter

This unit contains the implementation of the TPgAlerter component.

Classes

Name	Description
TPgAlerter	A component that allows to register interest in and asynchronously handle event notifications posted by PostgreSQL server.

5.14.1 Classes

Classes in the **PgAlerter** unit.

Classes

Name	Description
TPgAlerter	A component that allows to register interest in and asynchronously handle event notifications posted by PostgreSQL server.

5.14.1.1 TPgAlerter Class

A component that allows to register interest in and asynchronously handle event notifications posted by PostgreSQL server.

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

Unit

[PgAlerter](#)

Syntax


```
TPgAlerter = class(TDAAlerter);
```

Remarks

The TPgAlerter component allows you to register interest in and asynchronously handle event notifications posted by PostgreSQL server. Notifications are posted to all clients when a client executes the NOTIFY command.

Use TPgAlerter to handle events for responding to actions and database changes made by other applications.

To get events application must register the required events. To do it set the Events property to the required events and call the Start method. TPgAlerter calls the LISTEN command to listen on specified events. When one of the registered events occurs, the OnEvent handler is called.

Note: Notifications are transaction-based. This means that the waiting connection does not get event until the transaction posting the event commits.

Inheritance Hierarchy

[TDAAlerter](#)

TPgAlerter

See Also

- The Alerter demo program

5.14.1.1.1 Members

[TPgAlerter](#) class overview.

Properties

Name	Description
Active (inherited from TDAAlerter)	Used to determine if TDAAlerter waits for messages.
AutoRegister (inherited from TDAAlerter)	Used to automatically register events whenever connection opens.
Connection (inherited from TDAAlerter)	Used to specify the connection for TDAAlerter.
Events	Used to set the names of the events to wait.

Methods

Name	Description
SendEvent	Overloaded. Description is not available at the moment.
Start (inherited from TDAAlerter)	Starts waiting process.
Stop (inherited from TDAAlerter)	Stops waiting process.

Events

Name	Description
OnError (inherited from TDAAlerter)	Occurs if an exception occurs in waiting process
OnEvent	Occurs when waiting thread receives event notification from PostgreSQL server.

5.14.1.1.2 Properties

Properties of the **TPgAlerter** class.

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

Public

Name	Description
Active (inherited from TDAAlerter)	Used to determine if TDAAlerter waits for messages.
AutoRegister (inherited from TDAAlerter)	Used to automatically register events whenever connection opens.
Connection (inherited from TDAAlerter)	Used to specify the connection for TDAAlerter.

Published

Name	Description
------	-------------

[Events](#)

Used to set the names of the events to wait.

See Also

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

5.14.1.1.2.1 Events Property

Used to set the names of the events to wait.

Class

[TPgAlerter](#)

Syntax

```
property Events: string;
```

Remarks

Use the Events property to set the names of the events to wait.

5.14.1.1.3 Methods

Methods of the **TPgAlerter** class.

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

Public

Name	Description
SendEvent	Overloaded. Description is not available at the moment.
Start (inherited from TDAAlerter)	Starts waiting process.
Stop (inherited from TDAAlerter)	Stops waiting process.

See Also

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

5.14.1.1.3.1 SendEvent Method

Class

[TPgAlerter](#)

Overload List

Name	Description
SendEvent(const Name: string)	Sends an event notification with Name.
SendEvent(const Name: string; Message: string)	Sends an event notification with Name.

Sends an event notification with Name.

Class

[TPgAlerter](#)

Syntax

```
procedure SendEvent(const Name: string); overload;
```

Parameters*Name*

Message name.

Remarks

Call the SendEvent method to send an event notification with Name.

Sends an event notification with Name.

Class

[TPgAlerter](#)

Syntax

```
procedure SendEvent(const Name: string; Message: string);  
overload;
```

Parameters*Name*

Message name.

Message

Message text.

Remarks

Note: required PostgreSQL 9.0 or higher.

5.14.1.1.4 Events

Events of the **TPgAlerter** class.

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

Public

Name	Description
OnError (inherited from TDAAlerter)	Occurs if an exception occurs in waiting process

Published

Name	Description
OnEvent	Occurs when waiting thread receives event notification from PostgreSQL server.

See Also

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

5.14.1.1.4.1 OnEvent Event

Occurs when waiting thread receives event notification from PostgreSQL server.

Class

[TPgAlerter](#)

Syntax

```
property OnEvent: TPgNotificationEvent;
```

Remarks

OnEvent occurs when waiting thread receives event notification from PostgreSQL server.

5.15 PgClasses

This unit contains the implementation of internal PgDAC classes and types.

Classes

Name	Description
TPgCursor	A class for working with PostgreSQL cursors.

Enumerations

Name	Description
TProtocolVersion	Specifies version of protocol to be used when several versions are available.
TSSLMode	This option determines whether or with what priority an SSL connection will be negotiated with the server.

5.15.1 Classes

Classes in the **PgClasses** unit.

Classes

Name	Description
TPgCursor	A class for working with PostgreSQL cursors.

5.15.1.1 TPgCursor Class

A class for working with PostgreSQL cursors.

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

Unit

[PgClasses](#)

Syntax

```
TPgCursor = class(TCRCursor);
```

Remarks

The TPgCursor class is used to work with PostgreSQL cursors. This is a PgDAC internal class, and application should not use it directly.

Inheritance Hierarchy

[TSharedObject](#)

[TCRCursor](#)

TPgCursor

5.15.1.1.1 Members

[TPgCursor](#) class overview.

Properties

Name	Description
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
State	Used to set the cursor state.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.15.1.1.2 Properties

Properties of the **TPgCursor** class.

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

Public

Name	Description
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

[State](#)

Used to set the cursor state.

See Also

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

5.15.1.1.2.1 State Property

Used to set the cursor state.

Class

[TPgCursor](#)

Syntax

```
property State: TCursorState;
```

See Also

- [TCursorState](#)

5.15.2 Enumerations

Enumerations in the **PgClasses** unit.

Enumerations

Name	Description
TProtocolVersion	Specifies version of protocol to be used when several versions are available.
TSSLMode	This option determines whether or with what priority an SSL connection will be negotiated with the server.

5.15.2.1 TProtocolVersion Enumeration

Specifies version of protocol to be used when several versions are available.

Unit

[PgClasses](#)

Syntax


```
TProtocolVersion = (pv20, pv30);
```

Values

Value	Meaning
pv20	Version 2.0 protocol.
pv30	Version 3.0 protocol.

5.15.2.2 TSSLMode Enumeration

This option determines whether or with what priority an SSL connection will be negotiated with the server.

Unit

[PgClasses](#)

Syntax

```
TSSLMode = (smDisable, smRequire, smPrefer, smAllow);
```

Values

Value	Meaning
smAllow	Negotiates trying first a non-SSL connection, then if that fails, tries an SSL connection.
smDisable	Only an unencrypted SSL connection will be attempted.
smPrefer	Negotiates trying first an SSL connection, then if that fails, tries a regular non-SSL connection.
smRequire	Tries only an SSL connection.

Remarks

If PostgreSQL is compiled without SSL support, using option `smRequire` will cause an error, while options `smAllow` and `smPrefer` will be accepted, but PgDAC will not in fact attempt an SSL connection.

5.16 PgConnectionPool

This unit contains the `TPgConnectionPoolManager` class for managing connection pool.

Classes

Name	Description
------	-------------

[TPgConnectionPoolManager](#)

A class of methods that are used for managing PgDAC connection pool.

5.16.1 Classes

Classes in the **PgConnectionPool** unit.

Classes

Name	Description
TPgConnectionPoolManager	A class of methods that are used for managing PgDAC connection pool.

5.16.1.1 TPgConnectionPoolManager Class

A class of methods that are used for managing PgDAC connection pool.

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

Unit

[PgConnectionPool](#)

Syntax

```
TPgConnectionPoolManager = class(TCRConnectionPoolManager);
```

Remarks

Use the TPgConnectionPoolManager methods to manage PgDAC connection pool.

Inheritance Hierarchy

TCRConnectionPoolManager

TPgConnectionPoolManager

See Also

- [Connection Pooling](#)

5.16.1.1.1 Members

[TPgConnectionPoolManager](#) class overview.

5.17 PgDacVcl

This unit contains the visual constituent of PgDAC.

Classes

Name	Description
TPgConnectDialog	A component providing a dialog box for user to supply his login information.

5.17.1 Classes

Classes in the **PgDacVcl** unit.

Classes

Name	Description
TPgConnectDialog	A component providing a dialog box for user to supply his login information.

5.17.1.1 TPgConnectDialog Class

A component providing a dialog box for user to supply his login information.

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

Unit

[PgDacVcl](#)

Syntax

```
TPgConnectDialog = class(TCustomConnectDialog);
```

Remarks

The TPgConnectDialog class provides a dialog box for user to supply his login information. The TPgConnectDialog component is a direct descendant of the TCustomConnectDialog class. Use TPgConnectDialog to provide dialog box for a user to supply user name, password, server, port, and database name. You may want to customize appearance of the dialog box using properties of this class.

Inheritance Hierarchy

[TCustomConnectDialog](#)

TPgConnectDialog

5.17.1.1.1 Members

[TPgConnectDialog](#) class overview.

Properties

Name	Description
CancelButton (inherited from TCustomConnectDialog)	Used to specify the label for the Cancel button.
Caption (inherited from TCustomConnectDialog)	Used to set the caption of dialog box.
ConnectButton (inherited from TCustomConnectDialog)	Used to specify the label for the Connect button.
Connection	Holds the TPgConnection component which uses TPgConnectDialog object.
DatabaseLabel	Used to specify a prompt for database edit.
DialogClass (inherited from TCustomConnectDialog)	Used to specify the class of the form that will be displayed to enter login information.
LabelSet (inherited from TCustomConnectDialog)	Used to set the language of buttons and labels captions.
PasswordLabel (inherited from TCustomConnectDialog)	Used to specify a prompt for password edit.
PortLabel	Used to specify a prompt for port edit.
Retries (inherited from TCustomConnectDialog)	Used to indicate the number of retries of failed connections.

SavePassword (inherited from TCustomConnectDialog)	Used for the password to be displayed in ConnectDialog in asterisks.
ServerLabel (inherited from TCustomConnectDialog)	Used to specify a prompt for the server name edit.
ShowDatabase	Used to display a field for entering database at connect dialog.
ShowPort	Used to display a field for entering port at connect dialog.
StoreLogInfo (inherited from TCustomConnectDialog)	Used to specify whether the login information should be kept in system registry after a connection was established.
UsernameLabel (inherited from TCustomConnectDialog)	Used to specify a prompt for username edit.

Methods

Name	Description
Execute (inherited from TCustomConnectDialog)	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.
GetServerList (inherited from TCustomConnectDialog)	Retrieves a list of available server names.

5.17.1.1.2 Properties

Properties of the **TPgConnectDialog** class.

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

Public

Name	Description
------	-------------

<u>CancelButton</u> (inherited from <u>TCustomConnectDialog</u>)	Used to specify the label for the Cancel button.
<u>Caption</u> (inherited from <u>TCustomConnectDialog</u>)	Used to set the caption of dialog box.
<u>ConnectButton</u> (inherited from <u>TCustomConnectDialog</u>)	Used to specify the label for the Connect button.
<u>Connection</u>	Holds the TPgConnection component which uses TPgConnectDialog object.
<u>DatabaseLabel</u>	Used to specify a prompt for database edit.
<u>DialogClass</u> (inherited from <u>TCustomConnectDialog</u>)	Used to specify the class of the form that will be displayed to enter login information.
<u>LabelSet</u> (inherited from <u>TCustomConnectDialog</u>)	Used to set the language of buttons and labels captions.
<u>PasswordLabel</u> (inherited from <u>TCustomConnectDialog</u>)	Used to specify a prompt for password edit.
<u>PortLabel</u>	Used to specify a prompt for port edit.
<u>Retries</u> (inherited from <u>TCustomConnectDialog</u>)	Used to indicate the number of retries of failed connections.
<u>SavePassword</u> (inherited from <u>TCustomConnectDialog</u>)	Used for the password to be displayed in ConnectDialog in asterisks.
<u>ServerLabel</u> (inherited from <u>TCustomConnectDialog</u>)	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.

See Also

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

5.17.1.1.2.1 Connection Property

Holds the TPgConnection component which uses TPgConnectDialog object.

Class

[TPgConnectDialog](#)

Syntax

```
property Connection: TPgConnection;
```

Remarks

Use the Connection property to learn which TPgConnection component uses TPgConnectDialog object. This property is read-only.

See Also

- [TCustomDAConnection.ConnectDialog](#)

5.17.1.1.2.2 DatabaseLabel Property

Used to specify a prompt for database edit.

Class

[TPgConnectDialog](#)

Syntax

```
property DatabaseLabel: string;
```

Remarks

Use the DatabaseLabel property to specify a prompt for database edit.

See Also

- [ShowDatabase](#)

5.17.1.1.2.3 PortLabel Property

Used to specify a prompt for port edit.

Class

[TPgConnectDialog](#)

Syntax

```
property PortLabel: string;
```

Remarks

Use the PortLabel property to specify a prompt for port edit.

See Also

- [ShowPort](#)

5.17.1.1.2.4 Show Database Property

Used to display a field for entering database at connect dialog.

Class

[TPgConnectDialog](#)

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

- [DatabaseLabel](#)
- [ShowPort](#)

5.17.1.1.2.5 Show Port Property

Used to display a field for entering port at connect dialog.

Class

[TPgConnectDialog](#)

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](#)

5.18 PgDump

This unit contains the implementation of the TPgDump component.

Classes

Name	Description
TPgDump	A component for storing database or its parts as a script and also for restoring database from the received script.
TPgDumpOptions	This class allows setting up the behaviour of the TPgDump component.

Types

Name	Description
------	-------------

TPgDumpObjects	Represents the set of TPgDumpObject .
--------------------------------	---

Enumerations

Name	Description
TPgDumpMode	Specifies the mode of backup performed by TPgDump.
TPgDumpObject	Specifies the types of objects that are backed up by PgDump.

5.18.1 Classes

Classes in the **PgDump** unit.

Classes

Name	Description
TPgDump	A component for storing database or its parts as a script and also for restoring database from the received script.
TPgDumpOptions	This class allows setting up the behaviour of the TPgDump component.

5.18.1.1 TPgDump Class

A component for storing database or its parts as a script and also for restoring database from the received script.

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

Unit

[PgDump](#)

Syntax

```
TPgDump = class (TDADump) ;
```

Remarks

The TPgDump component is a component for storing database or its parts as a script and

also for restoring database from the received script.

Serves to store a database or its parts as a script and also to restore database from received script. TPgDump behaviour is similar to pg_dump program. Use SchemaNames, TableNames, and ObjectTypes properties to specify the objects to be stored. To generate a script call TDADump.Backup or TDADump.BackupQuery method. Resulted script can be viewed in [TDADump.SQL](#).

Inheritance Hierarchy

[TDADump](#)

TPgDump

5.18.1.1.1 Members

[TPgDump](#) class overview.

Properties

Name	Description
Connection (inherited from TDADump)	Used to specify a connection object that will be used to connect to a data store.
Debug (inherited from TDADump)	Used to display executing statement, all its parameters' values, and the type of parameters.
Mode	Used to specify the mode of backup performed by TPgDump.
ObjectTypes	Used to specify the types of objects that are backuped by PgDump.
Options	Specifies the behaviour of the TPgDump component.
SchemaNames	Used to set the names of the schemas to dump.
SQL (inherited from TDADump)	Used to set or get the dump script.
TableNames (inherited from TDADump)	Used to set the names of the tables to dump.

Methods

Name	Description
Backup (inherited from TDADump)	Dumps database objects to the TDADump.SQL property.
BackupQuery (inherited from TDADump)	Dumps the results of a particular query.
BackupToFile (inherited from TDADump)	Dumps database objects to the specified file.
BackupToStream (inherited from TDADump)	Dumps database objects to the stream.
Restore (inherited from TDADump)	Executes a script contained in the SQL property.
RestoreFromFile (inherited from TDADump)	Executes a script from a file.
RestoreFromStream (inherited from TDADump)	Executes a script received from the stream.

Events

Name	Description
OnBackupProgress (inherited from TDADump)	Occurs to indicate the TDADump.Backup , M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.
OnError (inherited from TDADump)	Occurs when PostgreSQL raises some error on TDADump.Restore .
OnRestoreProgress (inherited from TDADump)	Occurs to indicate the TDADump.Restore , TDADump.RestoreFromFile , or TDADump.RestoreFromStream method execution progress.

5.18.1.1.2 Properties

Properties of the **TPgDump** class.

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

Public

Name	Description
Connection (inherited from TDADump)	Used to specify a connection object that will be used to connect to a data store.

Published

Name	Description
Debug (inherited from TDADump)	Used to display executing statement, all its parameters' values, and the type of parameters.
Mode	Used to specify the mode of backup performed by TPgDump.
ObjectTypes	Used to specify the types of objects that are backed up by PgDump.
Options	Specifies the behaviour of the TPgDump component.
SchemaNames	Used to set the names of the schemas to dump.
SQL (inherited from TDADump)	Used to set or get the dump script.
TableNames (inherited from TDADump)	Used to set the names of the tables to dump.

See Also

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

5.18.1.1.2.1 Mode Property

Used to specify the mode of backup performed by TPgDump.

Class

[TPgDump](#)

Syntax

```
property Mode: TPgDumpMode default dmAll;
```

Remarks

Use the Mode property to specify the mode of backup performed by TPgDump.

5.18.1.1.2.2 ObjectTypes Property

Used to specify the types of objects that are backed up by PgDump.

Class

[TPgDump](#)

Syntax

```
property ObjectTypes: TPgDumpObjects default [doSchemas,  
doLanguages, doDomains, doTypes, doTables, doViews, doSequences,  
doStoredProcs, doTriggers, doIndexes];
```

Remarks

Use the ObjectTypes property to specify the types of objects that are backed up by PgDump.

5.18.1.1.2.3 Options Property

Specifies the behaviour of the TPgDump component.

Class

[TPgDump](#)

Syntax

```
property options: TPgDumpOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TPgDump component.

Descriptions of all options are in the table below.

Descriptions of all options are in the table below.

Option Name	Description
CreateConstraints	Used to add statements to create table constraints to the dump.

5.18.1.1.2.4 SchemaNames Property

Used to set the names of the schemas to dump.

Class

[TPgDump](#)

Syntax

```
property SchemaNames: string;
```

Remarks

Use the SchemaNames property to set the names of the schemas to dump. Table names must be separated with semicolons. If it is empty, the Backup method will dump all available schemas.

5.18.1.2 TPgDumpOptions Class

This class allows setting up the behaviour of the TPgDump component. For a list of all members of this type, see [TPgDumpOptions](#) members.

Unit

[PgDump](#)

Syntax

```
TPgDumpOptions = class(TDADumpOptions);
```

Inheritance Hierarchy

[TDADumpOptions](#)

TPgDumpOptions

5.18.1.2.1 Members

[TPgDumpOptions](#) class overview.

Properties

Name	Description
AddDrop (inherited from TDADumpOptions)	Used to add drop statements to a script before creating statements.

CreateConstraints	Used to add statements to create table constraints to the dump.
GenerateHeader (inherited from TDADumpOptions)	Used to add a comment header to a script.
QuoteNames (inherited from TDADumpOptions)	Used for TDADump to quote all database object names in generated SQL statements.

5.18.1.2.2 Properties

Properties of the **TPgDumpOptions** class.

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

Published

Name	Description
AddDrop (inherited from TDADumpOptions)	Used to add drop statements to a script before creating statements.
CreateConstraints	Used to add statements to create table constraints to the dump.
GenerateHeader (inherited from TDADumpOptions)	Used to add a comment header to a script.
QuoteNames (inherited from TDADumpOptions)	Used for TDADump to quote all database object names in generated SQL statements.

See Also

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

5.18.1.2.2.1 CreateConstraints Property

Used to add statements to create table constraints to the dump.

Class

[TPgDumpOptions](#)

Syntax

```
property CreateConstraints: boolean default True;
```

Remarks

If True (the default value), statements to create table constraints are added to the dump. If False, tables are created without constraints.

5.18.2 Types

Types in the **PgDump** unit.

Types

Name	Description
TPgDumpObjects	Represents the set of TPgDumpObject .

5.18.2.1 TPgDumpObjects Set

Represents the set of [TPgDumpObject](#).

Unit

[PgDump](#)

Syntax

```
TPgDumpObjects = set of TPgDumpObject;
```

5.18.3 Enumerations

Enumerations in the **PgDump** unit.

Enumerations

Name	Description
TPgDumpMode	Specifies the mode of backup performed by TPgDump.
TPgDumpObject	Specifies the types of objects that are backed up by PgDump.

5.18.3.1 TPgDumpMode Enumeration

Specifies the mode of backup performed by TPgDump.

Unit

[PgDump](#)

Syntax

```
TPgDumpMode = (dmAll, dmData, dmSchema);
```

Values

Value	Meaning
dmAll	Backup of schema objects and table data is performed. The default value.
dmData	Backup of table data only is performed.
dmSchema	Backup of schema only is performed.

5.18.3.2 TPgDumpObject Enumeration

Specifies the types of objects that are backed up by PgDump.

Unit

[PgDump](#)

Syntax

```
TPgDumpObject = (doDatabase, doDomains, doIndexes, doLanguages,  
doSchemas, doSequences, doStoredProcs, doTables, doTriggers,  
doTypes, doViews);
```

Values

Value	Meaning
doDatabase	If set, CREATE DATABASE statement is added to the dump.
doDomains	Backup of domains is performed.
doIndexes	Backup of indexes is performed.
doLanguages	Backup of languages is performed.
doSchemas	Backup of schemas is performed.
doSequences	Backup of sequences is performed.
doStoredProcs	Backup of stored procedures is performed.
doTables	Backup of tables is performed.

doTriggers	Backup of triggers is performed.
doTypes	Backup of types is performed.
doViews	Backup of views is performed.

5.19 PgError

This unit contains the EPgError exception class.

Classes

Name	Description
EPgError	Represents a PostgreSQL notice, warning, and error messages.

Enumerations

Name	Description
TPgSeverity	Specifies the severity of the event occurred.

5.19.1 Classes

Classes in the **PgError** unit.

Classes

Name	Description
EPgError	Represents a PostgreSQL notice, warning, and error messages.

5.19.1.1 EPgError Class

Represents a PostgreSQL notice, warning, and error messages.

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

Unit

[PgError](#)

Syntax

```
EPgError = class(EDAError);
```

Remarks

The EPgError object contains information that the server returns in the form of error, warning, log, notice etc. messages.

Inheritance Hierarchy

[EDAEError](#)

EPgError

5.19.1.1.1 Members

[EPgError](#) class overview.

Properties

Name	Description
CallStack	Includes a call stack traceback of active procedural language functions and internally-generated queries.
Component (inherited from EDAEError)	Contains the component that caused the error.
DetailMsg	Holds a secondary error message.
ErrorCode	Holds the SQLSTATE code.
FileName	Used to provide the file name of the source-code location where the error was reported.
Hint	Provides an optional suggestion on what to do about the problem.
LineNumber	Holds the line number of the source-code location where an error has occurred.
Position	Holds the error cursor position.
ProcedureName	Holds the name of the routine where an error has occurred.
Severity	Holds the TPgSeverity value.

5.19.1.1.2 Properties

Properties of the **EPgError** class.

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

Public

Name	Description
CallStack	Includes a call stack traceback of active procedural language functions and internally-generated queries.
Component (inherited from EDAError)	Contains the component that caused the error.
DetailMsg	Holds a secondary error message.
ErrorCode	Holds the SQLSTATE code.
FileName	Used to provide the file name of the source-code location where the error was reported.
Hint	Provides an optional suggestion on what to do about the problem.
LineNumber	Holds the line number of the source-code location where an error has occurred.
Position	Holds the error cursor position.
ProcedureName	Holds the name of the routine where an error has occurred.
Severity	Holds the TPgSeverity value.

See Also

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

5.19.1.1.2.1 CallStack Property

Includes a call stack traceback of active procedural language functions and internally-generated queries.

Class

[EPgError](#)

Syntax

```
property callstack: string;
```

Remarks

The CallStack property includes a call stack traceback of active procedural language functions and internally-generated queries. The trace is one entry per line, most recent first.

5.19.1.1.2.2 DetailMsg Property

Holds a secondary error message.

Class

[EPgError](#)

Syntax

```
property detailMsg: string;
```

Remarks

The DetailMsg property holds a secondary error message.

5.19.1.1.2.3 ErrorCode Property

Holds the SQLSTATE code.

Class

[EPgError](#)

Syntax

```
property errorCode: string;
```

Remarks

The ErrorCode property holds the SQLSTATE code.

5.19.1.1.2.4 FileName Property

Used to provide the file name of the source-code location where the error was reported.

Class

[EPgError](#)

Syntax

```
property FileName: string;
```

Remarks

Use the FileName property to get the file name of the source-code location.

5.19.1.1.2.5 Hint Property

Provides an optional suggestion on what to do about the problem.

Class

[EPgError](#)

Syntax

```
property Hint: string;
```

Remarks

The Hint property provides a piece of advice (potentially inappropriate) on how the problem can be solved.

5.19.1.1.2.6 LineNumber Property

Holds the line number of the source-code location where an error has occurred.

Class

[EPgError](#)

Syntax

```
property LineNumber: integer;
```

Remarks

The LineNumber property holds the line number of the source-code location.

5.19.1.1.2.7 Position Property

Holds the error cursor position.

Class

[EPgError](#)

Syntax

```
property Position: integer;
```

Remarks

The Position property holds the error cursor position as 1-based index in the original query string. The position is measured in characters.

5.19.1.1.2.8 ProcedureName Property

Holds the name of the routine where an error has occurred.

Class

[EPgError](#)

Syntax

```
property ProcedureName: string;
```

Remarks

The ProcedureName property holds the name of the source-code routine where an error has occurred.

5.19.1.1.2.9 Severity Property

Holds the [TPgSeverity](#) value.

Class

[EPgError](#)

Syntax

```
property Severity: TPgSeverity;
```

Remarks

The Severity property holds the message severity value.

5.19.2 Enumerations

Enumerations in the **PgError** unit.

Enumerations

Name	Description
TPgSeverity	Specifies the severity of the event occurred.

5.19.2.1 TPgSeverity Enumeration

Specifies the severity of the event occurred.

Unit

[PgError](#)

Syntax

```
TPgSeverity = (sError, sFatal, sPanic, sWarning, sNotice, sDebug, sInfo, sLog);
```

Values

Value	Meaning
sDebug	Information for developers containing internal server operations.
sError	Information about a user error.
sFatal	Fatal situation after which recovery isn't possible.
sInfo	Useful information of insignificant type.
sLog	Log of some commands execution by the server.
sNotice	Notification for the user about the occurred events.
sPanic	Critical error after which recovery isn't possible.
sWarning	Warnings for a user that can include information about possible problems.

5.20 PgLoader

This unit contains the implementation of the TPgLoader component.

Classes

Name	Description
------	-------------

TPgLoader	Description of TPgLoader is not available at the moment
TPgLoaderColumn	A base class holding a collection of TPgLoaderColumn objects.

5.20.1 Classes

Classes in the **PgLoader** unit.

Classes

Name	Description
TPgLoader	
TPgLoaderColumn	A base class holding a collection of TPgLoaderColumn objects.

5.20.1.1 TPgLoader Class

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

Unit

[PgLoader](#)

Syntax

```
TPgLoader = class (TDALoader);
```

Remarks

TPgLoader allows to load external data into the PostgreSQL database. It uses the COPY command to load data. To specify the name of the loading table set the TableName property. Use the Columns property to access individual columns. Write the OnGetColumnData or OnPutData event handler to read external data and pass it to the database. Call the Load method to start loading data.

Inheritance Hierarchy

[TDALoader](#)

TPgLoader

5.20.1.1.1 Members

[TPgLoader](#) class overview.

Properties

Name	Description
BufferSize	Holds the size of the memory buffer used by TPgLoader.
Columns (inherited from TDALoader)	Used to add a TDAColumn object for each field that will be loaded.
Connection	Used to specify a connection object and populate the columns connection.
TableName (inherited from TDALoader)	Used to specify the name of the table to which data will be loaded.
TextMode	Used to load data in the text mode.

Methods

Name	Description
CreateColumns (inherited from TDALoader)	Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName .
Load (inherited from TDALoader)	Starts loading data.
LoadFromDataSet (inherited from TDALoader)	Loads data from the specified dataset.
PutColumnData (inherited from TDALoader)	Overloaded. Puts the value of individual columns.

Events

Name	Description
OnGetColumnData (inherited from TDALoader)	Occurs when it is needed to put column values.

OnProgress (inherited from TDALoader)	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.
OnPutData (inherited from TDALoader)	Occurs when putting loading data by rows is needed.

5.20.1.1.2 Properties

Properties of the **TPgLoader** class.

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

Public

Name	Description
BufferSize	Holds the size of the memory buffer used by TPgLoader.
Columns (inherited from TDALoader)	Used to add a TDAColumn object for each field that will be loaded.
TableName (inherited from TDALoader)	Used to specify the name of the table to which data will be loaded.
TextMode	Used to load data in the text mode.

Published

Name	Description
Connection	Used to specify a connection object and populate the columns connection.

See Also

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

5.20.1.1.2.1 BufferSize Property

Holds the size of the memory buffer used by TPgLoader.

Class

[TPgLoader](#)

Syntax

```
property BufferSize: integer;
```

Remarks

The BufferSize property contains the size of the memory buffer used by TPgLoader. When buffer is filled, the loader sends block of data to the server.

5.20.1.1.2.2 Connection Property

Used to specify a connection object and populate the columns connection.

Class

[TPgLoader](#)

Syntax

```
property Connection: TPgConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store and to populate the columns collection.

5.20.1.1.2.3 TextMode Property

Used to load data in the text mode.

Class

[TPgLoader](#)

Syntax

```
property TextMode: boolean;
```

Remarks

Use the TextMode property to load data in the text mode.

TPgLoader supports two load modes: text and binary. By default the binary mode is used for a connection with 3.0 protocol. Set TextMode property to True to force text mode. In binary mode TPgLoader may work slightly faster but some data type are not supported in this mode. In text mode you can load data to columns with any PostgreSQL data type.

5.20.1.2 TPgLoaderColumn Class

A base class holding a collection of TPgLoaderColumn objects.

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

Unit

[PgLoader](#)

Syntax

```
TPgLoaderColumn = class(TDAColumn);
```

Remarks

Each TPgLoader uses TPgLoaderColumn to maintain a collection of TPgLoaderColumn objects. TPgLoaderColumn object represents the attributes for column loading. Every TPgLoaderColumn object corresponds to one of the table fields with the same name as its Name property.

To create columns at design time use column editor of TPgLoader component.

Inheritance Hierarchy

[TDAColumn](#)

TPgLoaderColumn

See Also

- [TPgLoader](#)

5.20.1.2.1 Members

[TPgLoaderColumn](#) class overview.

Properties

Name	Description
FieldType (inherited from TDAColumn)	Used to specify the types of values that will be loaded.
Name (inherited from TDAColumn)	Used to specify the field name of loading table.
RowTypeName	Holds the name of a composite type that corresponds to this field.

5.20.1.2.2 Properties

Properties of the **TPgLoaderColumn** class.

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

Published

Name	Description
FieldType (inherited from TDAColumn)	Used to specify the types of values that will be loaded.
Name (inherited from TDAColumn)	Used to specify the field name of loading table.
RowTypeName	Holds the name of a composite type that corresponds to this field.

See Also

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

5.20.1.2.2.1 Row TypeName Property

Holds the name of a composite type that corresponds to this field.

Class

[TPgLoaderColumn](#)

Syntax

```
property RowTypeName: string;
```

Remarks

The RowTypeName property holds the name of a composite type that corresponds to this field. If a field is of the composite (ROW) data type, assign the name of the type to this property. This is required for loading values of composite type in the binary mode.

5.21 PgObjects

This unit contains classes for PostgreSQL specific data types.

Classes

Name	Description
<u>TCustomPgTimeStamp</u>	A base class for the TPgDate, TPgTime, and TPgTimeStamp classes.
<u>TPgAttribute</u>	A class holding the description of an attribute of PostgreSQL composite type.
<u>TPgBox</u>	A class for working with PostgreSQL BOX data type.
<u>TPgCircle</u>	A class for working with PostgreSQL CIRCLE data type.
<u>TPgDate</u>	A class for working with PostgreSQL DATE data type.
<u>TPgGeometric</u>	A base class for classes that work with geometric data types.
<u>TPgInterval</u>	A class providing support of PostgreSQL INTERVAL datatype.
<u>TPgLSeg</u>	A class for working with PostgreSQL LSEG (line segment) data type.
<u>TPgPath</u>	A class for working with PostgreSQL PATH data type.
<u>TPgPoint</u>	A class for working with PostgreSQL POINT data type.
<u>TPgPointsArray</u>	A base class for working with geometric data types.
<u>TPgPolygon</u>	A class for working with PostgreSQL POLYGON data type.
<u>TPgRefCursor</u>	A class for working with REFCURSOR values.
<u>TPgRow</u>	A class for working with PostgreSQL composite (ROW) data types.
<u>TPgRowType</u>	A class holding the description of a PostgreSQL composite (ROW) type.
<u>TPgSQLLargeObject</u>	A class for working with PostgreSQL large objects.

TPgTime	A class for working with PostgreSQL TIME and TIME WITH TIMEZONE data type.
TPgTimeStamp	A class for working with PostgreSQL TIMESTAMP and TIMESTAMP WITH TIMEZONE data types.

5.21.1 Classes

Classes in the **PgObjects** unit.

Classes

Name	Description
TCustomPgTimeStamp	A base class for the TPgDate, TPgTime, and TPgTimeStamp classes.
TPgAttribute	A class holding the description of an attribute of PostgreSQL composite type.
TPgBox	A class for working with PostgreSQL BOX data type.
TPgCircle	A class for working with PostgreSQL CIRCLE data type.
TPgDate	A class for working with PostgreSQL DATE data type.
TPgGeometric	A base class for classes that work with geometric data types.
TPgInterval	A class providing support of PostgreSQL INTERVAL datatype.
TPgLSeg	A class for working with PostgreSQL LSEG (line segment) data type.
TPgPath	A class for working with PostgreSQL PATH data type.
TPgPoint	A class for working with PostgreSQL POINT data type.
TPgPointsArray	A base class for working with geometric data types.

TPgPolygon	A class for working with PostgreSQL POLYGON data type.
TPgRefCursor	A class for working with REFCURSOR values.
TPgRow	A class for working with PostgreSQL composite (ROW) data types.
TPgRowType	A class holding the description of a PostgreSQL composite (ROW) type.
TPgSQLLargeObject	A class for working with PostgreSQL large objects.
TPgTime	A class for working with PostgreSQL TIME and TIME WITH TIMEZONE data type.
TPgTimeStamp	A class for working with PostgreSQL TIMESTAMP and TIMESTAMP WITH TIMEZONE data types.

5.21.1.1 TCustomPgTimeStamp Class

A base class for the TPgDate, TPgTime, and TPgTimeStamp classes.

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

Unit

[PgObjects](#)

Syntax

```
TCustomPgTimeStamp = class(TSharedObject);
```

Remarks

The TCustomPgTimeStamp class is a base class for the TPgDate, TPgTime, and TPgTimeStamp classes.

Inheritance Hierarchy

[TSharedObject](#)

TCustomPgTimeStamp

5.21.1.1.1 Members

[TCustomPgTimeStamp](#) class overview.

Properties

Name	Description
AsDateTime	Used to get and set the timestamp value as TDateTime.
AsSQLTimeStamp	Used to get and set the timestamp value as TSQLTimeStamp record.
AsString	Used to get and set the timestamp value as string.
Days	Holds the date part of timestamp.
HasTimeZone	Used to specify whether the timestamp object has time zone.
IsInfinity	Determines if a timestamp object has +/-infinity value.
IsNegInfinity	Determines if a timestamp object has -infinity value.
IsPosInfinity	Determines if a timestamp object has +infinity value.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Ticks	Holds the time part of timestamp in microseconds.
TimeZoneOffset	Specifies the time zone offset of a timestamp object (in seconds).

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Assigns a value from another TCustomPgTimeStamp object to this

	object.
Compare	Compares two TCustomPgTimeStamp objects.
DecodeDate	Provides year, month, and day from the timestamp object.
DecodeDateTime	Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
DecodeTime	Provides hour, minute, second, and microsecond from the timestamp object.
EncodeDate	Sets year, month, and day in the timestamp object.
EncodeDateTime	Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
EncodeTime	Sets hour, minute, second, and microsecond in the timestamp object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.1.2 Properties

Properties of the **TCustomPgTimeStamp** class.

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

Public

Name	Description
AsDateTime	Used to get and set the timestamp value as TDateTime.
AsSQLTimeStamp	Used to get and set the timestamp value as TSQLTimeStamp record.
AsString	Used to get and set the timestamp value as string.
Days	Holds the date part of timestamp.

HasTimeZone	Used to specify whether the timestamp object has time zone.
IsInfinity	Determines if a timestamp object has +/-infinity value.
IsNegInfinity	Determines if a timestamp object has -infinity value.
IsPosInfinity	Determines if a timestamp object has +infinity value.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Ticks	Holds the time part of timestamp in microseconds.
TimeZoneOffset	Specifies the time zone offset of a timestamp object (in seconds).

See Also

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

5.21.1.1.2.1 AsDateTime Property

Used to get and set the timestamp value as TDateTime.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property AsDateTime: TDateTime;
```

Remarks

Use the AsDateTime property to get and set the timestamp value as TDateTime.

5.21.1.1.2.2 AsSQLTimeStamp Property

Used to get and set the timestamp value as TSQLTimeStamp record.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property AsSQLTimeStamp: TSQLTimeStamp;
```

Remarks

Use the AsString property to get and set the timestamp value as TSQLTimeStamp record.

5.21.1.1.2.3 AsString Property

Used to get and set the timestamp value as string.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to get and set the timestamp value as string.

5.21.1.1.2.4 Days Property

Holds the date part of timestamp.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property Days: integer;
```

Remarks

The Days property holds the date part of timestamp as a number of days since 01-01-2000.

See Also

- [DecodeDate](#)
- [EncodeDate](#)

5.21.1.1.2.5 HasTimeZone Property

Used to specify whether the timestamp object has time zone.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property HasTimeZone: boolean;
```

Remarks

Use the HasTimeZone property to specify whether the timestamp object has time zone.

5.21.1.1.2.6 IsInfinity Property

Determines if a timestamp object has +/-infinity value.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property IsInfinity: boolean;
```

Remarks

Use the IsInfinity property to determine if a timestamp object has +/-infinity value. The IsInfinity is True, if timestamp value is +infinity or -infinity, and False, if the value is not infinity.

5.21.1.1.2.7 IsNegInfinity Property

Determines if a timestamp object has -infinity value.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property IsNegInfinity: boolean;
```

Remarks

Use the IsNegInfinity property to determine if a timestamp object has -infinity value. The IsNegInfinity property is True, if timestamp value is -infinity.

5.21.1.1.2.8 IsPosInfinity Property

Determines if a timestamp object has +infinity value.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property IsPosInfinity: boolean;
```

Remarks

Use the IsPosInfinity property to determine if a timestamp object has +infinity value. The IsPosInfinity property is True, if timestamp value is +infinity.

5.21.1.1.2.9 Ticks Property

Holds the time part of timestamp in microseconds.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property Ticks: int64;
```

Remarks

The Ticks property holds the time part of timestamp in microseconds.

See Also

- [DecodeTime](#)
- [EncodeTime](#)

5.21.1.1.2.10 TimeZoneOffset Property

Specifies the time zone offset of a timestamp object (in seconds).

Class

[TCustomPgTimeStamp](#)

Syntax

```
property TimeZoneOffset: integer;
```


Remarks

Use the `TimeZoneOffset` property to specify the time zone offset of a timestamp object (in seconds).

For values of `TIMESTAMP WITH TIMEZONE` read this property to get the offset of the timestamp value from Universal Time (UTC).

The value of `TimeZoneOffset` is specified in seconds.

5.21.1.1.3 Methods

Methods of the **`TCustomPgTimeStamp`** class.

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

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the <code>TSharedObject</code> object.
Assign	Assigns a value from another <code>TCustomPgTimeStamp</code> object to this object.
Compare	Compares two <code>TCustomPgTimeStamp</code> objects.
DecodeDate	Provides year, month, and day from the timestamp object.
DecodeDateTime	Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
DecodeTime	Provides hour, minute, second, and microsecond from the timestamp object.
EncodeDate	Sets year, month, and day in the timestamp object.
EncodeDateTime	Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
EncodeTime	Sets hour, minute, second, and microsecond in the timestamp object.

[Release](#) (inherited from [TSharedObject](#))

Decrements the reference count.

See Also

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

5.21.1.1.3.1 Assign Method

Assigns a value from another TCustomPgTimeStamp object to this object.

Class

[TCustomPgTimeStamp](#)

Syntax

```
procedure Assign(Source: TCustomPgTimeStamp); virtual;
```

Parameters

Source

Holds the TCustomPgTimeStamp object to assign a value from.

Remarks

Call the Assign property to assign a value from another TCustomPgTimeStamp object to this object.

5.21.1.1.3.2 Compare Method

Compares two TCustomPgTimeStamp objects.

Class

[TCustomPgTimeStamp](#)

Syntax

```
function Compare(Value: TCustomPgTimeStamp): integer; virtual;  
abstract;
```

Parameters

Value

Holds a TCustomPgTimeStamp object to compare with.

Remarks

Call the Compare method to compare two TCustomPgTimeStamp objects.

If the current object represent date and time that is greater than Value, Compare returns a positive number.

If the current object represent date and time that is lower than Value, Compare returns a negative number.

If the current object equals to Value, Compare returns 0.

5.21.1.1.3.3 DecodeDate Method

Provides year, month, and day from the timestamp object.

Class

[TCustomPgTimeStamp](#)

Syntax

```
procedure DecodeDate(var Year: integer; var Month: integer; var Day: integer);
```

Parameters

Year

Holds the year.

Month

Holds the month.

Day

Holds the day.

Remarks

Call the DecodeDate method to get year, month, and day from the timestamp object.

5.21.1.1.3.4 DecodeDateTime Method

Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.

Class

[TCustomPgTimeStamp](#)

Syntax

```
procedure DecodeDateTime(var Year: integer; var Month: integer;
```

```
var Day: integer; var Hour: integer; var Minute: integer; var
Second: integer; var Microsecond: integer);
```

Parameters*Year*

Holds the year.

Month

Holds the month.

Day

Holds the day.

Hour

Holds the hour.

Minute

Holds the minute.

Second

Holds the second.

Microsecond

Holds the microsecond.

Remarks

Call the DecodeDateTime method to get the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.

5.21.1.1.3.5 DecodeTime Method

Provides hour, minute, second, and microsecond from the timestamp object.

Class[TCustomPgTimeStamp](#)**Syntax**

```
procedure DecodeTime(var Hour: integer; var Minute: integer; var
Second: integer; var Microsecond: integer);
```

Parameters*Hour*

Holds the hour.

Minute

Holds the minute.

Second

Holds the second.

Microsecond

Holds the microsecond.

Remarks

Call the DecodeTime method to get hour, minute, second, and microsecond from the timestamp object.

5.21.1.1.3.6 EncodeDate Method

Sets year, month, and day in the timestamp object.

Class

[TCustomPgTimeStamp](#)

Syntax

```
procedure EncodeDate(Year: integer; Month: integer; Day: integer);
```

Parameters

Year

Holds the year.

Month

Holds the month.

Day

Holds the day.

Remarks

Call the EncodeDate property to set year, month, and day in the timestamp object.

5.21.1.1.3.7 EncodeDateTime Method

Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.

Class

[TCustomPgTimeStamp](#)

Syntax

```
procedure EncodeDateTime(Year: integer; Month: integer; Day:  
integer; Hour: integer; Minute: integer; Second: integer;  
Microsecond: integer);
```

Parameters

Year

Holds the year.

Month

Holds the month.

Day

Holds the day.

Hour

Holds the hour.

Minute

Holds the minute.

Second

Holds the second.

Microsecond

Holds the microsecond.

Remarks

Call the `EncodeDateTime` method to set the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.

5.21.1.1.3.8 `EncodeTime` Method

Sets hour, minute, second, and microsecond in the timestamp object.

Class

[`TCustomPgTimeStamp`](#)

Syntax

```
procedure EncodeTime(Hour: integer; Minute: integer; Second:  
integer; Microsecond: integer);
```

Parameters

Hour

Holds the hour.

Minute

Holds the minute.

Second

Holds the second.

Microsecond

Holds the microsecond.

Remarks

Call the EncodeTime method to set hour, minute, second, and microsecond in the timestamp object.

5.21.1.2 TPgAttribute Class

A class holding the description of an attribute of PostgreSQL composite type.

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

Unit

[PgObjects](#)

Syntax

```
TPgAttribute = class(TAttribute);
```

Remarks

The TPgAttribute class holds the description of an attribute of PostgreSQL composite type. You can use TPgRowType.Attributes to access individual attributes.

Inheritance Hierarchy

[TAttribute](#)

TPgAttribute

5.21.1.2.1 Members

[TPgAttribute](#) class overview.

Properties

Name	Description
AttributeNo (inherited from TAttribute)	Returns an attribute's ordinal position in object.
DataSize (inherited from TAttribute)	Returns the size of an attribute value in internal representation.
DataType (inherited from TAttribute)	Returns the type of data that was assigned to the Attribute.
Length (inherited from TAttribute)	Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.
ObjectType (inherited from TAttribute)	Returns a TObjectType object for an object attribute.

Offset (inherited from TAttribute)	Returns an offset of the attribute value in internal representation.
Owner (inherited from TAttribute)	Indicates TObjectType that uses the attribute to represent one of its attributes.
Scale (inherited from TAttribute)	Returns the scale of dtFloat and dtInteger attributes.
Size (inherited from TAttribute)	Returns the size of an attribute value in external representation.

5.21.1.3 TPgBox Class

A class for working with PostgreSQL BOX data type.

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

Unit

[PgObjects](#)

Syntax

```
TPgBox = class (TPgPointsArray);
```

Remarks

The TPgBox class is used to work with PostgreSQL BOX data type.

Inheritance Hierarchy

[TSharedObject](#)

[TPgGeometric](#)

[TPgPointsArray](#)

TPgBox

5.21.1.3.1 Members

[TPgBox](#) class overview.

Properties

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.

LowerLeft	Holds the lower-left corner of the box.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
UpperRight	Holds the upper-right corner of the box.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.3.2 Properties

Properties of the **TPgBox** class.

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

Public

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
LowerLeft	Holds the lower-left corner of the box.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
UpperRight	Holds the upper-right corner of the box.

See Also

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

5.21.1.3.2.1 LowerLeft Property

Holds the lower-left corner of the box.

Class

[TPgBox](#)

Syntax

```
property LowerLeft: TPgPoint;
```

Remarks

The LowerLeft property holds the lower-left corner of the box.

5.21.1.3.2.2 UpperRight Property

Holds the upper-right corner of the box.

Class

[TPgBox](#)

Syntax

```
property UpperRight: TPgPoint;
```

Remarks

The UpperRight property holds the upper-right corner of the box.

5.21.1.4 TPgCircle Class

A class for working with PostgreSQL CIRCLE data type.

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

Unit

[PgObjects](#)

Syntax

```
TPgCircle = class(TPgGeometric);
```

Remarks

The TPgCircle class is used to work with PostgreSQL CIRCLE data type.

Inheritance Hierarchy

[TSharedObject](#)

[TPgGeometric](#)

TPgCircle

5.21.1.4.1 Members

[TPgCircle](#) class overview.

Properties

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
Center	Holds the center point of the circle.
Radius	Holds the radius of the circle.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.4.2 Properties

Properties of the **TPgCircle** class.

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

Public

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
Center	Holds the center point of the circle.
Radius	Holds the radius of the circle.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

See Also

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

5.21.1.4.2.1 Center Property

Holds the center point of the circle.

Class

[TPgCircle](#)

Syntax

```
property Center: TPgPoint;
```

Remarks

The Center property holds the center point of the circle.

5.21.1.4.2.2 Radius Property

Holds the radius of the circle.

Class

[TPgCircle](#)

Syntax

```
property Radius: Double;
```

Remarks

The Radius property holds the radius of the circle.

5.21.1.5 TPgDate Class

A class for working with PostgreSQL DATE data type.

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

Unit

[PgObjects](#)

Syntax

```
TPgDate = class (TCustomPgTimeStamp);
```

Remarks

The TPgDate class is used to work with PostgreSQL DATE data type.

Inheritance Hierarchy

[TSharedObject](#)

[TCustomPgTimeStamp](#)

TPgDate

5.21.1.5.1 Members

[TPgDate](#) class overview.

Properties

Name	Description
AsDateTime (inherited from TCustomPgTimeStamp)	Used to get and set the timestamp value as TDateTime.
AsSQLTimeStamp (inherited from TCustomPgTimeStamp)	Used to get and set the timestamp value as TSQLTimeStamp record.
AsString (inherited from TCustomPgTimeStamp)	Used to get and set the timestamp value as string.
Days (inherited from TCustomPgTimeStamp)	Holds the date part of timestamp.

<u>TCustomPgTimeStamp</u>)	
<u>HasTimeZone</u> (inherited from <u>TCustomPgTimeStamp</u>)	Used to specify whether the timestamp object has time zone.
<u>IsInfinity</u> (inherited from <u>TCustomPgTimeStamp</u>)	Determines if a timestamp object has +/-infinity value.
<u>IsNegInfinity</u> (inherited from <u>TCustomPgTimeStamp</u>)	Determines if a timestamp object has -infinity value.
<u>IsPosInfinity</u> (inherited from <u>TCustomPgTimeStamp</u>)	Determines if a timestamp object has +infinity value.
<u>RefCount</u> (inherited from <u>TSharedObject</u>)	Used to return the count of reference to a TSharedObject object.
<u>Ticks</u> (inherited from <u>TCustomPgTimeStamp</u>)	Holds the time part of timestamp in microseconds.
<u>TimeZoneOffset</u> (inherited from <u>TCustomPgTimeStamp</u>)	Specifies the time zone offset of a timestamp object (in seconds).

Methods

Name	Description
<u>AddRef</u> (inherited from <u>TSharedObject</u>)	Increments the reference count for the number of references dependent on the TSharedObject object.
<u>Assign</u> (inherited from <u>TCustomPgTimeStamp</u>)	Assigns a value from another TCustomPgTimeStamp object to this object.
<u>Compare</u> (inherited from <u>TCustomPgTimeStamp</u>)	Compares two TCustomPgTimeStamp objects.

DecodeDate (inherited from TCustomPgTimeStamp)	Provides year, month, and day from the timestamp object.
DecodeDateTime (inherited from TCustomPgTimeStamp)	Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
DecodeTime (inherited from TCustomPgTimeStamp)	Provides hour, minute, second, and microsecond from the timestamp object.
EncodeDate (inherited from TCustomPgTimeStamp)	Sets year, month, and day in the timestamp object.
EncodeDateTime (inherited from TCustomPgTimeStamp)	Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
EncodeTime (inherited from TCustomPgTimeStamp)	Sets hour, minute, second, and microsecond in the timestamp object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.6 TPgGeometric Class

A base class for classes that work with geometric data types.

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

Unit

[PgObjects](#)

Syntax

```
TPgGeometric = class(TSharedObject);
```

Remarks

The TPgGeometric class is a class for classes that work with geometric data types.

Inheritance Hierarchy

[TSharedObject](#)**TPgGeometric**

5.21.1.6.1 Members

[TPgGeometric](#) class overview.

Properties

Name	Description
AsString	Used to get and set the geometric value as string.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Assigns a value from another TPgGeometric object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.6.2 Properties

Properties of the **TPgGeometric** class.

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

Public

Name	Description
AsString	Used to get and set the geometric value as string.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

See Also

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

5.21.1.6.2.1 AsString Property

Used to get and set the geometric value as string.

Class

[TPgGeometric](#)

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to get and set the geometric value as string.

5.21.1.6.3 Methods

Methods of the **TPgGeometric** class.

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

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Assigns a value from another TPgGeometric object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

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

5.21.1.6.3.1 Assign Method

Assigns a value from another TPgGeometric object to this object.

Class

[TPgGeometric](#)

Syntax

```
procedure Assign(Source: TPgGeometric); virtual;
```

Parameters

Source

Holds a TPgGeometric object to assign the value from.

Remarks

Call the Assign property to assign a value from another TPgGeometric object to this object.

5.21.1.7 TPgInterval Class

A class providing support of PostgreSQL INTERVAL datatype.

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

Unit

[PgObjects](#)

Syntax

```
TPgInterval = class(TSharedObject);
```

Remarks

The TPgInterval class is used to work with PostgreSQL INTERVAL data type.

Inheritance Hierarchy

[TSharedObject](#)

TPgInterval

5.21.1.7.1 Members

[TPgInterval](#) class overview.

Properties

Name	Description
AsString	Used to get and set the INTERVAL value as string.
Days	Used to get the days part of an interval.
MonthsFull	Used to get the months part of an interval.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
SecondsFull	Used to get the time part of an interval represented in seconds.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Assigns a value from another TPgInterval object to this object.
Compare	Compares two TPgInterval objects.
DecodeInterval	Provides the value of interval as years, months, days, hours, minutes, seconds, and microseconds.
EncodeInterval	Sets the value of interval as years, months, days, hours, minutes, seconds, and microseconds.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.7.2 Properties

Properties of the **TPgInterval** class.

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

Public

Name	Description
------	-------------

AsString	Used to get and set the INTERVAL value as string.
Days	Used to get the days part of an interval.
MonthsFull	Used to get the months part of an interval.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
SecondsFull	Used to get the time part of an interval represented in seconds.

See Also

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

5.21.1.7.2.1 AsString Property

Used to get and set the INTERVAL value as string.

Class

[TPgInterval](#)

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to get and set the INTERVAL value as string.

5.21.1.7.2.2 Days Property

Used to get the days part of an interval.

Class

[TPgInterval](#)

Syntax

```
property Days: integer;
```

Remarks

Use the Days property to get the days part of an interval.

5.21.1.7.2.3 MonthsFull Property

Used to get the months part of an interval.

Class

[TPgInterval](#)

Syntax

```
property MonthsFull: integer;
```

Remarks

Use the MonthsFull property to get the months part of an interval.

The months part of an interval that includes months and years converted to months.

5.21.1.7.2.4 SecondsFull Property

Used to get the time part of an interval represented in seconds.

Class

[TPgInterval](#)

Syntax

```
property SecondsFull: double;
```

Remarks

Use the SecondsFull property to get the time part of an interval represented in seconds.

5.21.1.7.3 Methods

Methods of the **TPgInterval** class.

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

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.

Assign	Assigns a value from another TPgInterval object to this object.
Compare	Compares two TPgInterval objects.
DecodeInterval	Provides the value of interval as years, months, days, hours, minutes, seconds, and microseconds.
EncodeInterval	Sets the value of interval as years, months, days, hours, minutes, seconds, and microseconds.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

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

5.21.1.7.3.1 Assign Method

Assigns a value from another TPgInterval object to this object.

Class

[TPgInterval](#)

Syntax

```
procedure Assign(Source: TPgInterval);
```

Parameters

Source

Holds the TPgInterval object to assign value from.

Remarks

Call the Assign method to assign a value from another TPgInterval object to this object.

5.21.1.7.3.2 Compare Method

Compares two TPgInterval objects.

Class

[TPgInterval](#)

Syntax

```
function Compare(Value: TPgInterval): integer;
```

Parameters

Value

Holds a TPgInterval object to compare with.

Remarks

Call the Compare method to compare two TPgInterval objects.

If the current object represents an interval that is longer than Value, Compare returns a positive number. If the current object represents an interval that is shorter than Value, Compare returns a negative number. If the current object equals to Value, Compare returns 0. Internally INTERVAL values are stored as months, days, and seconds. Compare supposes that months value has more priority than days, and days has more priority than seconds.

5.21.1.7.3.3 DecodeInterval Method

Provides the value of interval as years, months, days, hours, minutes, seconds, and microseconds.

Class

[TPgInterval](#)

Syntax

```
procedure DecodeInterval(var Years: integer; var Months:  
integer; var Days: integer; var Hours: integer; var Minutes:  
integer; var Seconds: integer; var Microseconds: integer);
```

Parameters

Years

Holds the years value.

Months

Holds the months value.

Days

Holds the days value.

Hours

Holds the hours value.

Minutes

Holds the minutes value.

Seconds

Holds the seconds value.

Microseconds

Holds the microseconds value.

Remarks

Call the DecodeInterval method to get the value of interval as years, months, days, hours, minutes, seconds, and microseconds.

5.21.1.7.3.4 EncodeInterval Method

Sets the value of interval as years, months, days, hours, minutes, seconds, and microseconds.

Class

[TPgInterval](#)

Syntax

```
procedure EncodeInterval(Years: integer; Months: integer; Days:  
integer; Hours: integer; Minutes: integer; Seconds: integer;  
Microseconds: integer);
```

Parameters

Years

Holds the years value.

Months

Holds the months value.

Days

Holds the days value.

Hours

Holds the hours value.

Minutes

Holds the minutes value.

Seconds

Holds the seconds value.

Microseconds

Holds the microseconds value.

Remarks

Call the EncodeInterval method to set the value of interval as years, months, days, hours, minutes, seconds, and microseconds.

5.21.1.8 TPgLSeg Class

A class for working with PostgreSQL LSEG (line segment) data type.

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

Unit

[PgObjects](#)

Syntax

```
TPgLSeg = class(TPgPointsArray);
```

Remarks

The TPgLSeg class is used to work with PostgreSQL LSEG (line segment) data type.

Inheritance Hierarchy

[TSharedObject](#)

[TPgGeometric](#)

[TPgPointsArray](#)

TPgLSeg

5.21.1.8.1 Members

[TPgLSeg](#) class overview.

Properties

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
EndPoint	Holds the end point of the line segment.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
StartPoint	Holds the start point of the line segment.

Methods

Name	Description
------	-------------

AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.8.2 Properties

Properties of the **TPgLSeg** class.

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

Public

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
EndPoint	Holds the end point of the line segment.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
StartPoint	Holds the start point of the line segment.

See Also

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

5.21.1.8.2.1 EndPoint Property

Holds the end point of the line segment.

Class

[TPgLSeg](#)

Syntax

```
property EndPoint: TPgPoint;
```

Remarks

The EndPoint property holds the end point of the line segment.

5.21.1.8.2.2 StartPoint Property

Holds the start point of the line segment.

Class

[TPgLSeg](#)

Syntax

```
property StartPoint: TPgPoint;
```

Remarks

The StartPoint property holds the start point of the line segment.

5.21.1.9 TPgPath Class

A class for working with PostgreSQL PATH data type.

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

Unit

[PgObjects](#)

Syntax

```
TPgPath = class(TPgPointsArray);
```

Remarks

The TPgPath class is used to work with PostgreSQL PATH data type.

Inheritance Hierarchy

[TSharedObject](#)

[TPgGeometric](#)

[TPgPointsArray](#)

TPgPath

5.21.1.9.1 Members

[TPgPath](#) class overview.

Properties

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
Count	Holds the count of points in the path.
IsClosedPath	If True, the path is closed.
Points	Used to access a point in the path by its index.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.9.2 Properties

Properties of the **TPgPath** class.

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

Public

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.

Count	Holds the count of points in the path.
IsClosedPath	If True, the path is closed.
Points	Used to access a point in the path by its index.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

See Also

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

5.21.1.9.2.1 Count Property

Holds the count of points in the path.

Class

[TPgPath](#)

Syntax

```
property Count: integer;
```

Remarks

The Count property holds the count of points in the path.

5.21.1.9.2.2 IsClosedPath Property

If True, the path is closed.

Class

[TPgPath](#)

Syntax

```
property IsClosedPath: boolean;
```

Remarks

If the IsClosedPath property is True, the path is closed. This assumes that the last point in the

path is connected with the first point by a line.
If False, the path is not closed.

5.21.1.9.2.3 Points Property

Used to access a point in the path by its index.

Class

[TPgPath](#)

Syntax

```
property Points: array of TPgPoint;
```

Remarks

Use the Points property to access a point in the path by its index.

5.21.1.10 TPgPoint Class

A class for working with PostgreSQL POINT data type.
For a list of all members of this type, see [TPgPoint](#) members.

Unit

[PgObjects](#)

Syntax

```
TPgPoint = class(TPgGeometric);
```

Remarks

The TPgPoint class is used to work with PostgreSQL POINT data type.

Inheritance Hierarchy

[TSharedObject](#)

[TPgGeometric](#)

TPgPoint

5.21.1.10.1 Members

[TPgPoint](#) class overview.

Properties

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
X	Holds the X coordinate of the point.
Y	Holds the Y coordinate of the point.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Assigns a value from another TPgPoint object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.10.2 Properties

Properties of the **TPgPoint** class.

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

Public

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
X	Holds the X coordinate of the point.
Y	Holds the Y coordinate of the point.

See Also

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

5.21.1.10.2.1 X Property

Holds the X coordinate of the point.

Class

[TPgPoint](#)

Syntax

```
property X: Double;
```

Remarks

The X property holds the X coordinate of the point.

5.21.1.10.2.2 Y Property

Holds the Y coordinate of the point.

Class

[TPgPoint](#)

Syntax

```
property Y: Double;
```

Remarks

The Y property holds the Y coordinate of the point.

5.21.1.10.3 Methods

Methods of the **TPgPoint** class.

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

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.

Assign	Assigns a value from another TPgPoint object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

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

5.21.1.10.3.1 Assign Method

Assigns a value from another TPgPoint object to this object.

Class

[TPgPoint](#)

Syntax

```
procedure Assign(Source: TPgGeometric); override;
```

Parameters

Source

Holds the TPgPoint object to assign value from.

Remarks

Call the Assign method to assign a value from another TPgPoint object to this object.

5.21.1.11 TPgPointsArray Class

A base class for working with geometric data types.

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

Unit

[PgObjects](#)

Syntax

```
TPgPointsArray = class(TPgGeometric);
```

Remarks

The TPgPointsArray is a base class for most classes that work with geometric data types.

Inheritance Hierarchy

[TSharedObject](#)

[TPgGeometric](#)

TPgPointsArray

5.21.1.11.1 Members

[TPgPointsArray](#) class overview.

Properties

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.12 TPgPolygon Class

A class for working with PostgreSQL POLYGON data type.

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

Unit

[PgObjects](#)

Syntax

```
TPgPolygon = class(TPgPointsArray);
```

Remarks

The TPgPolygon class is used to work with PostgreSQL POLYGON data type.

Inheritance Hierarchy

[TSharedObject](#)

[TPgGeometric](#)

[TPgPointsArray](#)

TPgPolygon

5.21.1.12.1 Members

[TPgPolygon](#) class overview.

Properties

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
Count	Holds the count of points in the polygon.
Points	Used to access a point in the polygon by its index.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.12.2 Properties

Properties of the **TPgPolygon** class.

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

Public

Name	Description
AsString (inherited from TPgGeometric)	Used to get and set the geometric value as string.
Count	Holds the count of points in the polygon.
Points	Used to access a point in the polygon by its index.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.

See Also

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

5.21.1.12.2.1 Count Property

Holds the count of points in the polygon.

Class

[TPgPolygon](#)

Syntax

```
property Count: integer;
```

Remarks

The Count property holds the count of points in the polygon.

5.21.1.12.2.2 Points Property

Used to access a point in the polygon by its index.

Class

[TPgPolygon](#)

Syntax

```
property Points: array of TPgPoint;
```

Remarks

Use the Points property to access a point in the polygon by its index.

5.21.1.13 TPgRefCursor Class

A class for working with REFCURSOR values.

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

Unit

[PgObjects](#)

Syntax

```
TPgRefCursor = class(TPgCursor);
```

Remarks

The TPgRefCursor class is used to work with REFCURSOR values. An instance of TPgRefCursor holds the cursor name that is returned, for example, from a stored procedure, and can be used to fetch data from the cursor. Assign an instance of the TPgRefCursor class to the Cursor property of TPgQuery and call the TPgQuery.Open method to fetch data from the cursor.

Note: all operations with REFCURSOR require an active transaction.

Inheritance Hierarchy

[TSharedObject](#)

[TCRCursor](#)

[TPgCursor](#)

TPgRefCursor

5.21.1.13.1 Members

[TPgRefCursor](#) class overview.

Properties

Name	Description
------	-------------

CursorName	Used to get the name of a cursor.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
State (inherited from TPgCursor)	Used to set the cursor state.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.13.2 Properties

Properties of the **TPgRefCursor** class.

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

Public

Name	Description
CursorName	Used to get the name of a cursor.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
State (inherited from TPgCursor)	Used to set the cursor state.

See Also

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

5.21.1.13.2.1 CursorName Property

Used to get the name of a cursor.

Class

[TPgRefCursor](#)

Syntax

```
property CursorName: string;
```

Remarks

Use the CursorName property to get the name of a cursor.

5.21.1.14 TPgRow Class

A class for working with PostgreSQL composite (ROW) data types.

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

Unit

[PgObjects](#)

Syntax

```
TPgRow = class(TDBObject);
```

Remarks

The TPgRow class is used to work with PostgreSQL composite (ROW) data types.

Inheritance Hierarchy

[TSharedObject](#)

[TDBObject](#)

TPgRow

5.21.1.14.1 Members

[TPgRow](#) class overview.

Properties

Name	Description
------	-------------

<u>AsString</u>	Used to get and set the row value as string.
<u>AttrAsPgBox</u>	Used to get a reference to a TPgBox object for an attribute of ftPgBox type.
<u>AttrAsPgCircle</u>	Used to get a reference to a TPgCircle object for an attribute of ftPgCircle type.
<u>AttrAsPgCursor</u>	Used to get a reference to a TPgCursor object for an attribute of ftPgCursor type.
<u>AttrAsPgDate</u>	Used to get a reference to TPgDate object for attribute of ftPgDate type.
<u>AttrAsPgInterval</u>	Used to get a reference to a TPgInterval object for an attribute of ftPgInterval type.
<u>AttrAsPgLargeObject</u>	Used to get a reference to a TPgLargeObject object for an attribute of ftPgLargeObject type.
<u>AttrAsPgLSeg</u>	Used to get a reference to a TPgLSeg object for an attribute of ftPgLSeg type.
<u>AttrAsPgPath</u>	Used to get a reference to a TPgPath object for an attribute of ftPgPath type.
<u>AttrAsPgPoint</u>	Used to get a reference to a TPgPoint object for an attribute of ftPgPoint type.
<u>AttrAsPgPolygon</u>	Used to get a reference to a TPgPolygon object for an attribute of ftPgPolygon type.
<u>AttrAsPgRow</u>	Used to get a reference to a TPgRow object for an attribute of composite type.
<u>AttrAsPgTime</u>	Used to get a reference to a TPgTime object for an attribute of ftPgTime type.
<u>AttrAsPgTimeStamp</u>	Used to get a reference to a TPgTimeStamp object for an attribute of ftPgTimeStamp type.
<u>AttrIsNull</u>	Used to find out if an attribute is NULL, or to set an attribute value to NULL.

AttrValue	Used to get or set the value of an attribute.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
RowType	Holds the reference to a TPgRowType object containing information about corresponding composite type.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Assigns a value from another TPgRow object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.14.2 Properties

Properties of the **TPgRow** class.

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

Public

Name	Description
AsString	Used to get and set the row value as string.
AttrAsPgBox	Used to get a reference to a TPgBox object for an attribute of ftPgBox type.
AttrAsPgCircle	Used to get a reference to a TPgCircle object for an attribute of ftPgCircle type.
AttrAsPgCursor	Used to get a reference to a TPgCursor object for an attribute of ftPgCursor type.
AttrAsPgDate	Used to get a reference to TPgDate object for attribute of ftPgDate type.

AttrAsPgInterval	Used to get a reference to a TPgInterval object for an attribute of ftPgInterval type.
AttrAsPgLargeObject	Used to get a reference to a TPgLargeObject object for an attribute of ftPgLargeObject type.
AttrAsPgLSeg	Used to get a reference to a TPgLSeg object for an attribute of ftPgLSeg type.
AttrAsPgPath	Used to get a reference to a TPgPath object for an attribute of ftPgPath type.
AttrAsPgPoint	Used to get a reference to a TPgPoint object for an attribute of ftPgPoint type.
AttrAsPgPolygon	Used to get a reference to a TPgPolygon object for an attribute of ftPgPolygon type.
AttrAsPgRow	Used to get a reference to a TPgRow object for an attribute of composite type.
AttrAsPgTime	Used to get a reference to a TPgTime object for an attribute of ftPgTime type.
AttrAsPgTimeStamp	Used to get a reference to a TPgTimeStamp object for an attribute of ftPgTimeStamp type.
AttrIsNull	Used to find out if an attribute is NULL, or to set an attribute value to NULL.
AttrValue	Used to get or set the value of an attribute.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
RowType	Holds the reference to a TPgRowType object containing information about corresponding composite type.

See Also

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

5.21.1.14.2.1 AsString Property

Used to get and set the row value as string.

Class

[TPgRow](#)

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to get and set the row value as string.

5.21.1.14.2.2 AttrAsPgBox Property(Indexer)

Used to get a reference to a TPgBox object for an attribute of ftPgBox type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgBox[const Name: string]: TPgBox;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrAsPgBox property to get a reference to a TPgBox object for an attribute of ftPgBox type.

5.21.1.14.2.3 AttrAsPgCircle Property(Indexer)

Used to get a reference to a TPgCircle object for an attribute of ftPgCircle type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgCircle[const Name: string]: TPgCircle;
```

Parameters*Name*

Holds the name of an attribute.

Remarks

Use the AttrAsPgCircle property to get a reference to a TPgCircle object for an attribute of ftPgCircle type.

5.21.1.14.2.4 AttrAsPgCursor Property(Indexer)

Used to get a reference to a TPgCursor object for an attribute of ftPgCursor type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgCursor[const Name: string]: TPgCursor;
```

Parameters*Name*

Holds the name of an attribute.

Remarks

Use the AttrAsPgCursor property to get a reference to a TPgCursor object for an attribute of ftPgCursor type.

5.21.1.14.2.5 AttrAsPgDate Property(Indexer)

Used to get a reference to TPgDate object for attribute of ftPgDate type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgDate[const Name: string]: TPgDate;
```

Parameters*Name*

Holds the name of an attribute.

Remarks

Use the `AttrAsPgDate` property to get a reference to a `TPgDate` object for an attribute of `ftPgDate` type.

5.21.1.14.2.6 `AttrAsPgInterval` Property(Indexer)

Used to get a reference to a `TPgInterval` object for an attribute of `ftPgInterval` type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgInterval[const Name: string]: TPgInterval;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the `AttrAsPgInterval` property to get a reference to a `TPgInterval` object for an attribute of `ftPgInterval` type.

5.21.1.14.2.7 `AttrAsPgLargeObject` Property(Indexer)

Used to get a reference to a `TPgLargeObject` object for an attribute of `ftPgLargeObject` type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgLargeObject[const Name: string]:  
TPgSQLLargeObject;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the `AttrAsPgLargeObject` property to get a reference to a `TPgLargeObject` object for an attribute of `ftPgLargeObject` type.

5.21.1.14.2.8 AttrAsPgLSeg Property(Indexer)

Used to get a reference to a TPgLSeg object for an attribute of ftPgLSeg type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgLSeg[const Name: string]: TPgLSeg;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrAsPgLSeg property to get a reference to a TPgLSeg object for an attribute of ftPgLSeg type.

5.21.1.14.2.9 AttrAsPgPath Property(Indexer)

Used to get a reference to a TPgPath object for an attribute of ftPgPath type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgPath[const Name: string]: TPgPath;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrAsPgPath property to get a reference to a TPgPath object for an attribute of ftPgPath type.

5.21.1.14.2.10 AttrAsPgPoint Property(Indexer)

Used to get a reference to a TPgPoint object for an attribute of ftPgPoint type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgPoint[const Name: string]: TPgPoint;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrAsPgPoint property to get a reference to a TPgPoint object for an attribute of ftPgPoint type.

5.21.1.14.2.11 AttrAsPgPolygon Property(Indexer)

Used to get a reference to a TPgPolygon object for an attribute of ftPgPolygon type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgPolygon[const Name: string]: TPgPolygon;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrAsPgPolygon property to get a reference to a TPgPolygon object for an attribute of ftPgPolygon type.

5.21.1.14.2.12 AttrAsPgRow Property(Indexer)

Used to get a reference to a TPgRow object for an attribute of composite type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgRow[const Name: string]: TPgRow;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrAsPgRow property to get a reference to a TPgRow object for an attribute of composite type.

5.21.1.14.2.13 AttrAsPgTime Property(Indexer)

Used to get a reference to a TPgTime object for an attribute of ftPgTime type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgTime[const Name: string]: TPgTime;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrAsPgTime property to get a reference to a TPgTime object for an attribute of ftPgTime type.

5.21.1.14.2.14 AttrAsPgTimeStamp Property(Indexer)

Used to get a reference to a TPgTimeStamp object for an attribute of ftPgTimeStamp type.

Class

[TPgRow](#)

Syntax

```
property AttrAsPgTimeStamp[const Name: string]: TPgTimeStamp;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrAsPgTimeStamp property to get a reference to a TPgTimeStamp object for an attribute of ftPgTimeStamp type.

5.21.1.14.2.15 AttrIsNull Property(Indexer)

Used to find out if an attribute is NULL, or to set an attribute value to NULL.

Class

[TPgRow](#)

Syntax

```
property AttrIsNull[const Name: string]: boolean;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrIsNull property to find out if an attribute is NULL, or to set an attribute value to NULL.

5.21.1.14.2.16 AttrValue Property(Indexer)

Used to get or set the value of an attribute.

Class

[TPgRow](#)

Syntax

```
property AttrValue[const Name: string]: variant;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the AttrValue property to get or set the value of an attribute.

5.21.1.14.2.17 Row Type Property

Holds the reference to a TPgRowType object containing information about corresponding composite type.

Class

[TPgRow](#)

Syntax

```
property RowType: TPgRowType;
```

Remarks

The RowType property holds the reference to a TPgRowType object containing information about corresponding composite type. This property must be set before you can read or write values to attributes.

5.21.1.14.3 Methods

Methods of the **TPgRow** class.

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

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Assigns a value from another TPgRow object to this object.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

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

5.21.1.14.3.1 Assign Method

Assigns a value from another TPgRow object to this object.

Class

[TPgRow](#)

Syntax

```
procedure Assign(Source: TPgRow);
```

Parameters

Source

Holds the TPgRow object to assign a value from.

Remarks

Call the Assign method to assign a value from another TPgRow object to this object.

5.21.1.15 TPgRowType Class

A class holding the description of a PostgreSQL composite (ROW) type.

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

Unit

[PgObjects](#)

Syntax

```
TPgRowType = class(TPgType);
```

Remarks

The TPgRowType class holds the description of a PostgreSQL composite (ROW) type. Use the GetRowType method of TPgConnection to get an instance of a TPgRowType object.

Inheritance Hierarchy

[TSharedObject](#)

[TObjectType](#)

TPgType

TPgRowType

5.21.1.15.1 Members

[TPgRowType](#) class overview.

Methods

Name	Description
Describe	Overloaded. Provides information about a composite type from the database.

5.21.1.15.2 Methods

Methods of the **TPgRowType** class.

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

See Also

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

5.21.1.15.2.1 Describe Method

Provides information about a composite type from the database.

Class

[TPgRowType](#)

Overload List

Name	Description
Describe	Provides information about a composite type from the database by the OID.
Describe	Provides information about a composite type from the database by the name.

Provides information about a composite type from the database by the OID.

Unit

Syntax

Remarks

Call the Describe method to get information about a composite type from the database by the OID.

Provides information about a composite type from the database by the name.

Unit

Syntax

Remarks

Call the Describe method to get information about a composite type from the database by the name.

5.21.1.16 TPgSQLLargeObject Class

A class for working with PostgreSQL large objects.

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

Unit

[PgObjects](#)

Syntax

```
TPgSQLLargeObject = class(TCompressedBlob);
```

Remarks

The TPgSQLLargeObject class is used to work with PostgreSQL large objects. An instance of TPgSQLLargeObject holds an OID of large object and can be used for operations with this large object.

Note: all operations with large objects require an active transaction.

Inheritance Hierarchy

[TSharedObject](#)

[TBlob](#)

[TCompressedBlob](#)

TPgSQLLargeObject

5.21.1.16.1 Members

[TPgSQLLargeObject](#) class overview.

Properties

Name	Description
AsString (inherited from TBlob)	Used to manipulate BLOB value as string.
AsWideString (inherited from TBlob)	Used to manipulate BLOB value as Unicode string.
Cached	Used to specify whether the content of large object should be stored in memory buffer on the client.
Compressed (inherited from TCompressedBlob)	Used to indicate if the Blob is compressed.

CompressedSize (inherited from TCompressedBlob)	Used to indicate compressed size of the Blob data.
Connection	Used to specify a connection object that is used to perform operations with large object.
IsUnicode (inherited from TBlob)	Gives choice of making TBlob store and process data in Unicode format or not.
OID	Holds the OID of a large object.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size (inherited from TBlob)	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TBlob)	Sets BLOB value from another TBlob object.
Clear (inherited from TBlob)	Deletes the current value in TBlob object.
CloseObject	Closes the large object that was previously opened by the OpenObject method.
CreateObject	Creates a new large object database.
LoadFromFile (inherited from TBlob)	Loads the contents of a file into a TBlob object.
LoadFromStream (inherited from TBlob)	Copies the contents of a stream into the TBlob object.
OpenObject	Opens the large object specified by the OID property.

Read (inherited from TBlob)	Acquires a raw sequence of bytes from the data stored in TBlob.
ReadBlob	Reads the content of the large object from the database.
Release (inherited from TSharedObject)	Decrements the reference count.
SaveToFile (inherited from TBlob)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from TBlob)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from TBlob)	Sets new TBlob size and discards all data over it.
UnlinkObject	Deletes the large object specified by the OID property from the database.
Write (inherited from TBlob)	Stores a raw sequence of bytes into a TBlob object.
WriteBlob	Writes the content of the large object to the database.

5.21.1.16.2 Properties

Properties of the **TPgSQLLargeObject** class.

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

Public

Name	Description
AsString (inherited from TBlob)	Used to manipulate BLOB value as string.
AsWideString (inherited from TBlob)	Used to manipulate BLOB value as Unicode string.
Cached	Used to specify whether the content of large object should be stored in memory buffer on the client.
Compressed (inherited from	Used to indicate if the Blob is compressed.

<u>TCompressedBlob</u>)	
<u>CompressedSize</u> (inherited from <u>TCompressedBlob</u>)	Used to indicate compressed size of the Blob data.
<u>Connection</u>	Used to specify a connection object that is used to perform operations with large object.
<u>IsUnicode</u> (inherited from <u>TBlob</u>)	Gives choice of making TBlob store and process data in Unicode format or not.
<u>OID</u>	Holds the OID of a large object.
<u>RefCount</u> (inherited from <u>TSharedObject</u>)	Used to return the count of reference to a TSharedObject object.
<u>Size</u> (inherited from <u>TBlob</u>)	Used to learn the size of the TBlob value in bytes.

See Also

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

5.21.1.16.2.1 Cached Property

Used to specify whether the content of large object should be stored in memory buffer on the client.

Class

[TPgSQLLargeObject](#)

Syntax

```
property cached: boolean;
```

Remarks

Use the Cached property to specify whether the content of large object should be stored in memory buffer on the client.

By default the whole content of large object is stored in the memory buffer on the client. Set Cached to False if you want to reduce memory usage in your application when working with

very large object. When `Cached` is `False`, the `Read` and `Write` methods read/write data directly from/to the database.

5.21.1.16.2.2 Connection Property

Used to specify a connection object that is used to perform operations with large object.

Class

[TPgSQLLargeObject](#)

Syntax

```
property Connection: TPgSQLConnection;
```

Remarks

Use the `Connection` property to specify a connection object that is used to perform operations with large object.

5.21.1.16.2.3 OID Property

Holds the `OID` of a large object.

Class

[TPgSQLLargeObject](#)

Syntax

```
property OID: OID;
```

Remarks

Read the `OID` property to get the `OID` of the object created by the `CreateObject` method. Set the `OID` property to work with an existing large object.

5.21.1.16.3 Methods

Methods of the **TPgSQLLargeObject** class.

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

Public

Name	Description
------	-------------

AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TBlob)	Sets BLOB value from another TBlob object.
Clear (inherited from TBlob)	Deletes the current value in TBlob object.
CloseObject	Closes the large object that was previously opened by the OpenObject method.
CreateObject	Creates a new large object database.
LoadFromFile (inherited from TBlob)	Loads the contents of a file into a TBlob object.
LoadFromStream (inherited from TBlob)	Copies the contents of a stream into the TBlob object.
OpenObject	Opens the large object specified by the OID property.
Read (inherited from TBlob)	Acquires a raw sequence of bytes from the data stored in TBlob.
ReadBlob	Reads the content of the large object from the database.
Release (inherited from TSharedObject)	Decrements the reference count.
SaveToFile (inherited from TBlob)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from TBlob)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from TBlob)	Sets new TBlob size and discards all data over it.
UnlinkObject	Deletes the large object specified by the OID property from the database.
Write (inherited from TBlob)	Stores a raw sequence of bytes into a TBlob object.
WriteBlob	Writes the content of the large object to the database.

See Also

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

5.21.1.16.3.1 CloseObject Method

Closes the large object that was previously opened by the OpenObject method.

Class

[TPgSQLLargeObject](#)

Syntax

```
procedure CloseObject;
```

Remarks

Call the CloseObject property to close the large object that was previously opened by the OpenObject method.

5.21.1.16.3.2 CreateObject Method

Creates a new large object database.

Class

[TPgSQLLargeObject](#)

Syntax

```
procedure CreateObject;
```

Remarks

Call the CreateObject method to create a new large object database.

5.21.1.16.3.3 OpenObject Method

Opens the large object specified by the OID property.

Class

[TPgSQLLargeObject](#)

Syntax

```
procedure OpenObject;
```

Remarks

Call the OpenObject method to open the large object specified by the OID property.

5.21.1.16.3.4 ReadBlob Method

Reads the content of the large object from the database.

Class

[TPgSQLLargeObject](#)

Syntax

```
procedure ReadBlob; overload; procedure ReadBlob(var SharedPiece:  
PPieceHeader); overload;
```

Remarks

Call the ReadBlob method to get the content of large object from the database. When reading such properties as AsString or AsWideString, this method is called automatically.

5.21.1.16.3.5 UnlinkObject Method

Deletes the large object specified by the OID property from the database.

Class

[TPgSQLLargeObject](#)

Syntax

```
procedure UnlinkObject;
```

Remarks

Call the UnlinkObject method to delete the large object specified by the OID property from the database.

5.21.1.16.3.6 WriteBlob Method

Writes the content of the large object to the database.

Class

[TPgSQLLargeObject](#)

Syntax

```
procedure writeBlob;
```

Remarks

Call the WriteBlob method to write the content of the large object to the database.
When the Cached property is set to True, call WriteLob to write the cached content of the large object to the database.

5.21.1.17 TPgTime Class

A class for working with PostgreSQL TIME and TIME WITH TIMEZONE data type.
For a list of all members of this type, see [TPgTime](#) members.

Unit

[PgObjects](#)

Syntax

```
TPgTime = class(TCustomPgTimeStamp);
```

Remarks

The TPgTime class is used to work with PostgreSQL TIME and TIME WITH TIMEZONE data type.

Inheritance Hierarchy

[TSharedObject](#)

[TCustomPgTimeStamp](#)

TPgTime

5.21.1.17.1 Members

[TPgTime](#) class overview.

Properties

Name	Description
AsDateTime (inherited from TCustomPgTimeStamp)	Used to get and set the timestamp value as TDateTime.
AsSQLTimeStamp (inherited from TCustomPgTimeStamp)	Used to get and set the timestamp value as TSQLTimeStamp record.

<u>TCustomPgTimeStamp</u>)	
<u>AsString</u> (inherited from <u>TCustomPgTimeStamp</u>)	Used to get and set the timestamp value as string.
<u>Days</u> (inherited from <u>TCustomPgTimeStamp</u>)	Holds the date part of timestamp.
<u>HasTimeZone</u> (inherited from <u>TCustomPgTimeStamp</u>)	Used to specify whether the timestamp object has time zone.
<u>IsInfinity</u> (inherited from <u>TCustomPgTimeStamp</u>)	Determines if a timestamp object has +/-infinity value.
<u>IsNegInfinity</u> (inherited from <u>TCustomPgTimeStamp</u>)	Determines if a timestamp object has -infinity value.
<u>IsPosInfinity</u> (inherited from <u>TCustomPgTimeStamp</u>)	Determines if a timestamp object has +infinity value.
<u>RefCount</u> (inherited from <u>TSharedObject</u>)	Used to return the count of reference to a TSharedObject object.
<u>Ticks</u> (inherited from <u>TCustomPgTimeStamp</u>)	Holds the time part of timestamp in microseconds.
<u>TimeZoneOffset</u>	Used to get or set the time zone offset.

Methods

Name	Description
<u>AddRef</u> (inherited from <u>TSharedObject</u>)	Increments the reference count for the number of references dependent on the TSharedObject object.

Assign (inherited from TCustomPgTimeStamp)	Assigns a value from another TCustomPgTimeStamp object to this object.
Compare (inherited from TCustomPgTimeStamp)	Compares two TCustomPgTimeStamp objects.
DecodeDate (inherited from TCustomPgTimeStamp)	Provides year, month, and day from the timestamp object.
DecodeDateTime (inherited from TCustomPgTimeStamp)	Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
DecodeTime (inherited from TCustomPgTimeStamp)	Provides hour, minute, second, and microsecond from the timestamp object.
EncodeDate (inherited from TCustomPgTimeStamp)	Sets year, month, and day in the timestamp object.
EncodeDateTime (inherited from TCustomPgTimeStamp)	Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
EncodeTime (inherited from TCustomPgTimeStamp)	Sets hour, minute, second, and microsecond in the timestamp object.
Release (inherited from TSharedObject)	Decrements the reference count.

5.21.1.17.2 Properties

Properties of the **TPgTime** class.

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

Public

Name	Description
------	-------------

<u>AsDateTime</u> (inherited from <u>TCustomPgTimeStamp</u>)	Used to get and set the timestamp value as TDateTime.
<u>AsSQLTimeStamp</u> (inherited from <u>TCustomPgTimeStamp</u>)	Used to get and set the timestamp value as TSQLTimeStamp record.
<u>AsString</u> (inherited from <u>TCustomPgTimeStamp</u>)	Used to get and set the timestamp value as string.
<u>Days</u> (inherited from <u>TCustomPgTimeStamp</u>)	Holds the date part of timestamp.
<u>HasTimeZone</u> (inherited from <u>TCustomPgTimeStamp</u>)	Used to specify whether the timestamp object has time zone.
<u>IsInfinity</u> (inherited from <u>TCustomPgTimeStamp</u>)	Determines if a timestamp object has +/-infinity value.
<u>IsNegInfinity</u> (inherited from <u>TCustomPgTimeStamp</u>)	Determines if a timestamp object has -infinity value.
<u>IsPosInfinity</u> (inherited from <u>TCustomPgTimeStamp</u>)	Determines if a timestamp object has +infinity value.
<u>RefCount</u> (inherited from <u>TSharedObject</u>)	Used to return the count of reference to a TSharedObject object.
<u>Ticks</u> (inherited from <u>TCustomPgTimeStamp</u>)	Holds the time part of timestamp in microseconds.
<u>TimeZoneOffset</u>	Used to get or set the time zone offset.

See Also

- [TPgTime Class](#)

- [TPgTime Class Members](#)

5.21.1.17.2.1 TimeZoneOffset Property

Used to get or set the time zone offset.

Class

[TPgTime](#)

Syntax

```
property TimeZoneOffset: integer;
```

Remarks

Use the TimeZoneOffset property to get or set the time zone offset (in seconds).

For values of TIME WITH TIMEZONE use this property to get or set offset of time value from Universal Time (UTC).

Value of TimeZoneOffset is in seconds.

5.21.1.18 TPgTimeStamp Class

A class for working with PostgreSQL TIMESTAMP and TIMESTAMP WITH TIMEZONE data types.

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

Unit

[PgObjects](#)

Syntax

```
TPgTimeStamp = class(TCustomPgTimeStamp);
```

Remarks

The TPgTimeStamp class is used to work with PostgreSQL TIMESTAMP and TIMESTAMP WITH TIMEZONE data types.

Inheritance Hierarchy

[TSharedObject](#)

[TCustomPgTimeStamp](#)

TPgTimeStamp

5.21.1.18.1 Members

[TPgTimeStamp](#) class overview.

Properties

Name	Description
AsDateTime (inherited from TCustomPgTimeStamp)	Used to get and set the timestamp value as TDateTime.
AsSQLTimeStamp (inherited from TCustomPgTimeStamp)	Used to get and set the timestamp value as TSQLTimeStamp record.
AsString (inherited from TCustomPgTimeStamp)	Used to get and set the timestamp value as string.
Days (inherited from TCustomPgTimeStamp)	Holds the date part of timestamp.
HasTimeZone (inherited from TCustomPgTimeStamp)	Used to specify whether the timestamp object has time zone.
IsInfinity (inherited from TCustomPgTimeStamp)	Determines if a timestamp object has +/-infinity value.
IsNegInfinity (inherited from TCustomPgTimeStamp)	Determines if a timestamp object has -infinity value.
IsPosInfinity (inherited from TCustomPgTimeStamp)	Determines if a timestamp object has +infinity value.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Ticks (inherited from TCustomPgTimeStamp)	Holds the time part of timestamp in microseconds.

TimeZoneOffset (inherited from TCustomPgTimeStamp)	Specifies the time zone offset of a timestamp object (in seconds).
---	--

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from TCustomPgTimeStamp)	Assigns a value from another TCustomPgTimeStamp object to this object.
Compare (inherited from TCustomPgTimeStamp)	Compares two TCustomPgTimeStamp objects.
DecodeDate (inherited from TCustomPgTimeStamp)	Provides year, month, and day from the timestamp object.
DecodeDateTime (inherited from TCustomPgTimeStamp)	Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
DecodeTime (inherited from TCustomPgTimeStamp)	Provides hour, minute, second, and microsecond from the timestamp object.
EncodeDate (inherited from TCustomPgTimeStamp)	Sets year, month, and day in the timestamp object.
EncodeDateTime (inherited from TCustomPgTimeStamp)	Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
EncodeTime (inherited from TCustomPgTimeStamp)	Sets hour, minute, second, and microsecond in the timestamp object.

[Release](#) (inherited from [TSharedObject](#))

Decrements the reference count.

5.22 PgScript

This unit contains the implementation of the TPgScript component.

Classes

Name	Description
TPgScript	Executes sequences of SQL statements.

5.22.1 Classes

Classes in the **PgScript** unit.

Classes

Name	Description
TPgScript	Executes sequences of SQL statements.

5.22.1.1 TPgScript Class

Executes sequences of SQL statements.

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

Unit

[PgScript](#)

Syntax

```
TPgScript = class(TDAScript);
```

Inheritance Hierarchy

[TDAScript](#)

TPgScript

5.22.1.1.1 Members

[TPgScript](#) class overview.

Properties

Name	Description
Connection (inherited from TDAScript)	Used to specify the connection in which the script will be executed.
DataSet (inherited from TDAScript)	Refers to a dataset that holds the result set of query execution.
Debug (inherited from TDAScript)	Used to display the script execution and all its parameter values.
Delimiter (inherited from TDAScript)	Used to set the delimiter string that separates script statements.
EndLine (inherited from TDAScript)	Used to get the current statement last line number in a script.
EndOffset (inherited from TDAScript)	Used to get the offset in the last line of the current statement.
EndPos (inherited from TDAScript)	Used to get the end position of the current statement.
Macros (inherited from TDAScript)	Used to change SQL script text in design- or run-time easily.
SQL (inherited from TDAScript)	Used to get or set script text.
StartLine (inherited from TDAScript)	Used to get the current statement start line number in a script.
StartOffset (inherited from TDAScript)	Used to get the offset in the first line of the current statement.
StartPos (inherited from TDAScript)	Used to get the start position of the current statement in a script.
Statements (inherited from TDAScript)	Contains a list of statements obtained from the SQL property.

Methods

Name	Description
------	-------------

BreakExec (inherited from TDAScript)	Stops script execution.
ErrorOffset (inherited from TDAScript)	Used to get the offset of the statement if the Execute method raised an exception.
Execute (inherited from TDAScript)	Executes a script.
ExecuteFile (inherited from TDAScript)	Executes SQL statements contained in a file.
ExecuteNext (inherited from TDAScript)	Executes the next statement in the script and then stops.
ExecuteStream (inherited from TDAScript)	Executes SQL statements contained in a stream object.
MacroByName (inherited from TDAScript)	Finds a Macro with the name passed in Name.

Events

Name	Description
AfterExecute (inherited from TDAScript)	Occurs after a SQL script execution.
BeforeExecute (inherited from TDAScript)	Occurs when taking a specific action before executing the current SQL statement is needed.
OnError (inherited from TDAScript)	Occurs when PostgreSQL raises an error.

5.23 PgSQLMonitor

This unit contains the implementation of the TPgSQLMonitor component.

Classes

Name	Description
TPgSQLMonitor	This component serves for monitoring dynamic SQL execution in PgDAC-based applications.

5.23.1 Classes

Classes in the **PgSQLMonitor** unit.

Classes

Name	Description
TPgSQLMonitor	This component serves for monitoring dynamic SQL execution in PgDAC-based applications.

5.23.1.1 TPgSQLMonitor Class

This component serves for monitoring dynamic SQL execution in PgDAC-based applications. For a list of all members of this type, see [TPgSQLMonitor](#) members.

Unit

[PgSQLMonitor](#)

Syntax

```
TPgSQLMonitor = class(TCustomDASQLMonitor);
```

Remarks

Use TPgSQLMonitor to monitor dynamic SQL execution in PgDAC-based applications. TPgSQLMonitor provides two ways of displaying debug information: with dialog window, [DBMonitor](#) or Borland SQL Monitor. Furthermore to receive debug information the [TCustomDASQLMonitor.OnSQL](#) event can be used. Also it is possible to use all these ways at the same time, though an application may have only one TPgSQLMonitor object. If an application has no TPgSQLMonitor instance, the Debug window is available to display SQL statements to be sent.

Inheritance Hierarchy

[TCustomDASQLMonitor](#)

TPgSQLMonitor

See Also

- [TCustomDADataset.Debug](#)
- [TCustomDASQL.Debug](#)
- [DBMonitor](#)

5.23.1.1.1 Members

[TPgSQLMonitor](#) class overview.

Properties

Name	Description
Active (inherited from TCustomDASQLMonitor)	Used to activate monitoring of SQL.
DBMonitorOptions (inherited from TCustomDASQLMonitor)	Used to set options for dbMonitor.
Options (inherited from TCustomDASQLMonitor)	Used to include the desired properties for TCustomDASQLMonitor.
TraceFlags (inherited from TCustomDASQLMonitor)	Used to specify which database operations the monitor should track in an application at runtime.

Events

Name	Description
OnSQL (inherited from TCustomDASQLMonitor)	Occurs when tracing of SQL activity on database components is needed.

5.24 VirtualDataSet

5.24.1 Classes

Classes in the **VirtualDataSet** unit.

Classes

Name	Description
TCustomVirtualDataSet	A base class for representation of arbitrary data in tabular form.

[TVirtualDataSet](#)

Dataset that processes arbitrary non-tabular data.

5.24.1.1 TCustomVirtualDataSet Class

A base class for representation of arbitrary data in tabular form.

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

Unit

virtualDataSet

Syntax

```
TCustomVirtualDataSet = class(TMemDataSet);
```

Inheritance Hierarchy

[TMemDataSet](#)

TCustomVirtualDataSet

5.24.1.1.1 Members

[TCustomVirtualDataSet](#) class overview.

Properties

Name	Description
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
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

Name	Description
ApplyRange (inherited from TMemDataSet)	Applies a range to the dataset.
ApplyUpdates (inherited from TMemDataSet)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from TMemDataSet)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from TMemDataSet)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
EditRangeEnd (inherited from TMemDataSet)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from TMemDataSet)	Enables changing the starting value for an existing range.

GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
Locate (inherited from TMemDataSet)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from TMemDataSet)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from TMemDataSet)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from TMemDataSet)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from TMemDataSet)	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

Name	Description
OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from TMemDataSet)	Occurs when a single update component can not handle the updates.

5.24.1.2 TVirtualDataSet Class

Dataset that processes arbitrary non-tabular data.

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

Unit

virtualDataSet

Syntax

```
TVirtualDataSet = class(TCustomVirtualDataSet);
```

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomVirtualDataSet](#)

TVirtualDataSet

5.24.1.2.1 Members

[TVirtualDataSet](#) class overview.

Properties

Name	Description
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.

<u>TMemDataSet</u>)	
<u>KeyExclusive</u> (inherited from <u>TMemDataSet</u>)	Specifies the upper and lower boundaries for a range.
<u>LocalConstraints</u> (inherited from <u>TMemDataSet</u>)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.
<u>UpdateRecordTypes</u> (inherited from <u>TMemDataSet</u>)	Used to indicate the update status for the current record when cached updates are enabled.
<u>UpdatesPending</u> (inherited from <u>TMemDataSet</u>)	Used to check the status of the cached updates buffer.

Methods

Name	Description
<u>ApplyRange</u> (inherited from <u>TMemDataSet</u>)	Applies a range to the dataset.
<u>ApplyUpdates</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Writes dataset's pending cached updates to a database.
<u>CancelRange</u> (inherited from <u>TMemDataSet</u>)	Removes any ranges currently in effect for a dataset.
<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.

<u>TMemDataSet</u>)	
<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>EditRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Enables changing the ending value for an existing range.
<u>EditRangeStart</u> (inherited from <u>TMemDataSet</u>)	Enables changing the starting value for an existing range.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Prepare</u> (inherited from <u>TMemDataSet</u>)	Allocates resources and creates field components for a dataset.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SetRange</u> (inherited from <u>TMemDataSet</u>)	Sets the starting and ending values of a range, and applies it.
<u>SetRangeEnd</u> (inherited from <u>TMemDataSet</u>)	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

Name	Description
OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from TMemDataSet)	Occurs when a single update component can not handle the updates.

5.24.2 Types

Types in the **VirtualDataSet** unit.

Types

Name	Description
TOnDeleteRecordEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.
TOnGetFieldValueEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.
TOnGetRecordCountEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.

[TOnModifyRecordEvent](#)

This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.

5.24.2.1 TOnDeleteRecordEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.

Unit

virtualDataSet

Syntax

```
TOnDeleteRecordEvent = procedure (Sender: TObject; RecNo: Integer) of object;
```

Parameters

Sender

An object that raised the event.

RecNo

Number of the record being deleted.

5.24.2.2 TOnGetFieldValueEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.

Unit

virtualDataSet

Syntax

```
TOnGetFieldValueEvent = procedure (Sender: TObject; Field: TField; RecNo: Integer; out Value: Variant) of object;
```

Parameters

Sender

An object that raised the event.

Field

The field, which data has to be returned.

RecNo

The number of the record, which data has to be returned.

Value

Requested field value.

5.24.2.3 TOnGetRecordCountEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.

Unit

virtualDataSet

Syntax

```
TOnGetRecordCountEvent = procedure (Sender: TObject; out Count: Integer) of object;
```

Parameters

Sender

An object that raised the event.

Count

The number of records that the virtual dataset will contain.

5.24.2.4 TOnModifyRecordEvent Procedure Reference

This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.

Unit

virtualDataSet

Syntax

```
TOnModifyRecordEvent = procedure (Sender: TObject; var RecNo: Integer) of object;
```

Parameters

Sender

An object that raised the event.

RecNo

Number of the record being inserted or modified.

5.25 VirtualTable

5.25.1 Classes

Classes in the **VirtualTable** unit.

Classes

Name	Description
TVirtualTable	Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

5.25.1.1 TVirtualTable Class

Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

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

Unit

virtualTable

Syntax

```
TVirtualTable = class(TMemDataSet);
```

Inheritance Hierarchy

[TMemDataSet](#)

TVirtualTable

5.25.1.1.1 Members

[TVirtualTable](#) class overview.

Properties

Name	Description
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
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

<u>TMemDataSet</u>)	opening TMemDataSet.
<u>LocalUpdate</u> (inherited from <u>TMemDataSet</u>)	Used to prevent implicit update of rows on database server.
<u>Prepared</u> (inherited from <u>TMemDataSet</u>)	Determines whether a query is prepared for execution or not.
<u>Ranged</u> (inherited from <u>TMemDataSet</u>)	Indicates whether a range is applied to a dataset.
<u>UpdateRecordTypes</u> (inherited from <u>TMemDataSet</u>)	Used to indicate the update status for the current record when cached updates are enabled.
<u>UpdatesPending</u> (inherited from <u>TMemDataSet</u>)	Used to check the status of the cached updates buffer.

Methods

Name	Description
<u>ApplyRange</u> (inherited from <u>TMemDataSet</u>)	Applies a range to the dataset.
<u>ApplyUpdates</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Writes dataset's pending cached updates to a database.
<u>CancelRange</u> (inherited from <u>TMemDataSet</u>)	Removes any ranges currently in effect for a dataset.
<u>CancelUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears all pending cached updates from cache and restores dataset in its prior state.
<u>CommitUpdates</u> (inherited from <u>TMemDataSet</u>)	Clears the cached updates buffer.
<u>DeferredPost</u> (inherited from <u>TMemDataSet</u>)	Makes permanent changes to the database server.
<u>EditRangeEnd</u> (inherited from	Enables changing the ending value for an existing range.

<u>TMemDataSet</u>)	
<u>EditRangeStart</u> (inherited from <u>TMemDataSet</u>)	Enables changing the starting value for an existing range.
<u>GetBlob</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
<u>Locate</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
<u>LocateEx</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Excludes features that don't need to be included to the <u>TMemDataSet.Locate</u> method of TDataSet.
<u>Prepare</u> (inherited from <u>TMemDataSet</u>)	Allocates resources and creates field components for a dataset.
<u>RestoreUpdates</u> (inherited from <u>TMemDataSet</u>)	Marks all records in the cache of updates as unapplied.
<u>RevertRecord</u> (inherited from <u>TMemDataSet</u>)	Cancels changes made to the current record when cached updates are enabled.
<u>SaveToXML</u> (inherited from <u>TMemDataSet</u>)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
<u>SetRange</u> (inherited from <u>TMemDataSet</u>)	Sets the starting and ending values of a range, and applies it.
<u>SetRangeEnd</u> (inherited from <u>TMemDataSet</u>)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
<u>SetRangeStart</u> (inherited from <u>TMemDataSet</u>)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
<u>UnPrepare</u> (inherited from <u>TMemDataSet</u>)	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

Name	Description
OnUpdateError (inherited from TMemDataSet)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from TMemDataSet)	Occurs when a single update component can not handle the updates.