

Table of Contents

Part I What's New	1
Part II General Information	10
1 Overview	11
2 Features	14
3 Requirements	19
4 Compatibility	19
5 Using Several DAC Products in One IDE	26
6 Component List	26
7 Hierarchy Chart	28
8 Editions	30
9 Licensing	33
10 Getting Support	36
11 FAQ	37
Part III Getting Started	45
1 Installation	51
2 Connecting to PostgreSQL	54
3 Deleting Data From Tables	58
4 Creating Database Objects	62
5 Inserting Data Into Tables	67
6 Retrieving Data From Tables	72
7 Modifying Data in Tables	75
8 Using Stored Procedures	79
9 Using Stored Functions with Result Sets	85
10 Demo Projects	87
11 Deployment	92
Part IV Using PgDAC	94
1 Connecting to Heroku Postgres	95
2 Updating Data with PgDAC Dataset Components	98
3 Master/Detail Relationships	99
4 Automatic Key Field Value Generation	101
5 Data Type Mapping	102
6 Data Encryption	108
7 Working in an Unstable Network	111
8 Secure Connections	112

Connecting via SSL	112
Connecting via SSH	118
Network Tunneling	125
9 Disconnected Mode	131
10 Batch Operations	133
11 Increasing Performance	137
12 Macros	139
13 DataSet Manager	140
14 TPgLoader Component	146
15 Large Objects	147
16 REFCURSOR Data Type	149
17 National and Unicode Characters	151
18 Connection Pooling	152
19 DBMonitor	154
20 Writing GUI Applications with PgDAC	155
21 Compatibility with Previous Versions	155
22 64-bit Development with Embarcadero RAD Studio XE2	156
23 Database Specific Aspects of 64-bit Development	162
Part V Reference	163
1 CRAccess	165
Classes	166
TCRCursor Class	166
Members	166
Types	167
TBeforeFetchProc Procedure Reference	167
Enumerations	168
TCRIsolationLevel Enumeration	168
TCRTransactionAction Enumeration	169
TCursorState Enumeration	170
2 CRBatchMove	171
Classes	171
TCRBatchMove Class	172
Members	172
Properties	174
AbortOnKeyViol Property	176
AbortOnProblem Property	177
ChangedCount Property	177
CommitCount Property	178
Destination Property	178
FieldMappingMode Property	178
KeyViolCount Property	179
Mappings Property	180
Mode Property	180
MovedCount Property	181
ProblemCount Property	181
RecordCount Property	182

Source Property.....	183
Methods	183
Execute Method.....	184
Events	184
OnBatchMoveProgress Event.....	185
Types	185
TCRBatchMoveProgressEvent Procedure Reference.....	185
Enumerations	186
TCRBatchMode Enumeration.....	186
TCRFieldMappingMode Enumeration.....	187
3 CREncryption	188
Classes	188
TCREncryptor Class.....	189
Members	189
Properties	190
DataHeader Property.....	191
EncryptionAlgorithm Property.....	191
HashAlgorithm Property.....	192
InvalidHashAction Property.....	192
Passw ord Property.....	193
Methods	193
SetKey Method.....	194
Enumerations	195
TCREncDataHeader Enumeration.....	195
TCREncryptionAlgorithm Enumeration.....	196
TCRHashAlgorithm Enumeration.....	196
TCRInvalidHashAction Enumeration.....	197
4 CRVio	197
Classes	198
THttpOptions Class.....	198
Members	199
Properties	200
Enabled Property.....	200
Passw ord Property.....	201
ProxyOptions Property.....	201
TrustServerCertificate Property.....	202
Url Property	202
Username Property.....	203
TProxyOptions Class.....	203
Members	204
Properties	204
Hostname Property.....	205
Passw ord Property.....	205
Port Property	206
Username Property.....	206
Enumerations	206
TIPVersion Enumeration.....	207
5 CRXml	208
Structs	208
TAttribute Record.....	208
6 DAAlerter	209
Classes	209
TDAAAlerter Class.....	210

Members	210
Properties	211
Active Property.....	212
AutoRegister Property.....	212
Connection Property.....	213
Methods	213
SendEvent Method.....	214
Start Method	214
Stop Method	215
Events	216
OnError Event.....	216
Types	217
TAlertOnErrorEvent Procedure Reference.....	217
7 DADump	217
Classes	218
TDADump Class.....	218
Members	219
Properties	221
Connection Property.....	222
Debug Property.....	222
Options Property.....	223
SQL Property	224
TableNames Property.....	224
Methods	225
Backup Method.....	226
BackupQuery Method.....	226
BackupToFile Method.....	227
BackupToStream Method.....	228
Restore Method.....	229
RestoreFromFile Method.....	229
RestoreFromStream Method.....	230
Events	231
OnBackupProgress Event.....	232
OnError Event.....	232
OnRestoreProgress Event.....	233
TDADumpOptions Class.....	234
Members	234
Properties	235
AddDrop Property.....	235
CompleteInsert Property.....	236
GenerateHeader Property.....	236
QuoteNames Property.....	237
Types	237
TDABackupProgressEvent Procedure Reference.....	238
TDARestoreProgressEvent Procedure Reference.....	238
8 DALoader	239
Classes	240
TDAColumn Class.....	240
Members	241
Properties	241
FieldType Property.....	242
Name Property.....	242
TDAColumns Class.....	243

Members	243
Properties	244
Items Property(Indexer).....	244
TDALoader Class.....	245
Members	245
Properties	246
Columns Property.....	247
Connection Property.....	248
TableName Property.....	248
Methods	249
CreateColumns Method.....	249
Load Method	250
LoadFromDataSet Method.....	251
PutColumnData Method.....	251
PutColumnData Method.....	252
PutColumnData Method.....	252
Events	253
OnGetColumnData Event.....	254
OnProgress Event.....	255
OnPutData Event.....	255
TDALoaderOptions Class.....	256
Members	257
Properties	257
UseBlankValues Property.....	258
Types	258
TDAPutDataEvent Procedure Reference.....	258
TGetColumnDataEvent Procedure Reference.....	259
TLoaderProgressEvent Procedure Reference.....	260
9 DAScript	260
Classes	261
TDAScript Class.....	262
Members	262
Properties	264
Connection Property.....	266
DataSet Property.....	266
Debug Property.....	267
Delimiter Property.....	267
EndLine Property.....	268
EndOffset Property.....	268
EndPos Property	269
Macros Property.....	269
SQL Property	270
StartLine Property.....	270
StartOffset Property.....	271
StartPos Property.....	271
Statements Property.....	272
Methods	273
BreakExec Method.....	274
ErrorOffset Method.....	274
Execute Method.....	275
ExecuteFile Method.....	275
ExecuteNext Method.....	276
ExecuteStream Method.....	276
FindMacro Method.....	277

MacroByName Method.....	278
Events	279
AfterExecute Event.....	279
BeforeExecute Event.....	280
OnError Event.....	280
TDAStatement Class.....	281
Members	281
Properties	282
EndLine Property.....	283
EndOffset Property.....	284
EndPos Property.....	284
Omit Property	285
Params Property.....	285
Script Property.....	286
SQL Property	286
StartLine Property.....	287
StartOffset Property.....	287
StartPos Property.....	287
Methods	288
Execute Method.....	288
TDAStatements Class.....	289
Members	289
Properties	290
Items Property(Indexer).....	290
Types	291
TAfterStatementExecuteEvent Procedure Reference.....	291
TBeforeStatementExecuteEvent Procedure Reference.....	292
TOnErrorEvent Procedure Reference.....	292
Enumerations	293
TErrorAction Enumeration.....	293
10 DASQLMonitor	294
Classes	295
TCustomDASQLMonitor Class.....	295
Members	296
Properties	297
Active Property.....	297
DBMonitorOptions Property.....	298
Options Property.....	298
TraceFlags Property.....	299
Events	299
OnSQL Event	300
TDBMonitorOptions Class.....	300
Members	301
Properties	301
Host Property	302
Port Property	303
ReconnectTimeout Property.....	303
SendTimeout Property.....	304
Types	304
TDATraceFlags Set.....	305
TMonitorOptions Set.....	305
TOnSQLEvent Procedure Reference.....	305
Enumerations	306
TDATraceFlag Enumeration.....	306

TMonitorOption Enumeration.....	307
11 DBAccess	308
Classes	311
EDAError Class	313
Members	314
Properties	314
Component Property.....	315
ErrorCode Property.....	315
TCRDataSource Class.....	316
Members	316
TCustomConnectDialog Class.....	316
Members	317
Properties	318
CancelButton Property.....	319
Caption Property.....	320
ConnectButton Property.....	320
DialogClass Property.....	320
LabelSet Property.....	321
PasswordLabel Property.....	322
Retries Property.....	322
SavePassword Property.....	322
ServerLabel Property.....	323
StoreLogInfo Property.....	323
UsernameLabel Property.....	324
Methods	324
Execute Method.....	325
GetServerList Method.....	325
TCustomDAConnection Class.....	326
Members	327
Properties	329
ConnectDialog Property.....	330
ConnectString Property.....	331
ConvertEOL Property.....	333
InTransaction Property.....	333
LoginPrompt Property.....	334
Options Property.....	334
Password Property.....	335
Pooling Property.....	336
PoolingOptions Property.....	337
Server Property.....	338
Username Property.....	338
Methods	339
ApplyUpdates Method.....	340
ApplyUpdates Method.....	341
ApplyUpdates Method.....	341
Commit Method.....	342
Connect Method.....	343
CreateSQL Method.....	344
Disconnect Method.....	344
ExecProc Method.....	345
ExecProcEx Method.....	347
ExecSQL Method.....	348
ExecSQLEx Method.....	349
GetDatabaseNames Method.....	350

GetKeyFieldNames Method.....	351
GetStoredProcNames Method.....	352
GetTableNames Method.....	353
MonitorMessage Method.....	354
Ping Method	354
RemoveFromPool Method.....	355
Rollback Method.....	355
StartTransaction Method.....	356
Events	357
OnConnectionLost Event.....	357
OnError Event.....	358
TCustomDADataSet Class.....	358
Members	359
Properties	367
BaseSQL Property.....	371
Conditions Property.....	371
Connection Property.....	372
DataTypeMap Property.....	372
Debug Property.....	373
DetailFields Property.....	373
Disconnected Property.....	374
FetchRows Property.....	375
FilterSQL Property.....	375
FinalSQL Property.....	376
IsQuery Property.....	377
KeyFields Property.....	377
MacroCount Property.....	378
Macros Property.....	378
MasterFields Property.....	379
MasterSource Property.....	380
Options Property.....	381
ParamCheck Property.....	383
ParamCount Property.....	384
Params Property.....	384
ReadOnly Property.....	385
RefreshOptions Property.....	386
RowsAffected Property.....	386
SQL Property	387
SQLDelete Property.....	388
SQLInsert Property.....	388
SQLLock Property.....	389
SQLRecCount Property.....	390
SQLRefresh Property.....	391
SQLUpdate Property.....	392
UniDirectional Property.....	393
Methods	394
AddWhere Method.....	398
BreakExec Method.....	398
CreateBlobStream Method.....	399
DeleteWhere Method.....	400
Execute Method.....	400
Execute Method.....	401
Execute Method.....	401
Executing Method.....	402

Fetched Method.....	403
Fetching Method.....	403
FetchingAll Method.....	404
FindKey Method.....	404
FindMacro Method.....	405
FindNearest Method.....	406
FindParam Method.....	407
GetData Type Method.....	407
GetFieldObject Method.....	408
GetFieldPrecision Method.....	409
GetFieldScale Method.....	409
GetKeyFieldNames Method.....	410
GetOrderBy Method.....	411
GotoCurrent Method.....	411
Lock Method.....	412
MacroByName Method.....	413
ParamByName Method.....	414
Prepare Method.....	415
RefreshRecord Method.....	415
RestoreSQL Method.....	416
SaveSQL Method.....	417
SetOrderBy Method.....	417
SQLSaved Method.....	418
UnLock Method.....	418
Events.....	419
AfterExecute Event.....	420
AfterFetch Event.....	420
AfterUpdateExecute Event.....	421
BeforeFetch Event.....	421
BeforeUpdateExecute Event.....	422
TCustomDASQL Class.....	422
Members.....	423
Properties.....	425
ChangeCursor Property.....	427
Connection Property.....	427
Debug Property.....	428
FinalSQL Property.....	429
MacroCount Property.....	429
Macros Property.....	430
ParamCheck Property.....	430
ParamCount Property.....	431
Params Property.....	432
ParamValues Property(Indexer).....	433
Prepared Property.....	433
RowsAffected Property.....	434
SQL Property.....	434
Methods.....	435
BreakExec Method.....	436
Execute Method.....	437
Execute Method.....	437
Execute Method.....	438
Executing Method.....	438
FindMacro Method.....	439
FindParam Method.....	440

MacroByName Method.....	440
ParamByName Method.....	441
Prepare Method.....	442
UnPrepare Method.....	443
WaitExecuting Method.....	443
Events	444
AfterExecute Event.....	444
TCustomDAUpdateSQL Class.....	445
Members	445
Properties	447
DataSet Property.....	448
DeleteObject Property.....	449
DeleteSQL Property.....	449
InsertObject Property.....	450
InsertSQL Property.....	450
LockObject Property.....	451
LockSQL Property.....	451
ModifyObject Property.....	452
ModifySQL Property.....	452
RefreshObject Property.....	453
RefreshSQL Property.....	453
SQL Property(Indexer).....	454
Methods	455
Apply Method	455
ExecSQL Method.....	456
TDACondition Class.....	457
Members	457
Properties	458
Enabled Property.....	459
Name Property.....	459
Value Property.....	459
Methods	460
Disable Method.....	460
Enable Method.....	460
TDAConditions Class.....	461
Members	461
Properties	463
Condition Property(Indexer).....	463
Enabled Property.....	464
Items Property(Indexer).....	464
Text Property	465
WhereSQL Property.....	465
Methods	465
Add Method	466
Add Method	467
Add Method	467
Delete Method.....	468
Disable Method.....	469
Enable Method.....	469
Find Method	469
Get Method	470
IndexOf Method.....	470
Remove Method.....	471
TDAConnectionOptions Class.....	471

Members	471
Properties	472
Allow ImplicitConnect Property	473
DefaultSortType Property	474
DisconnectedMode Property	475
KeepDesignConnected Property	475
LocalFailover Property	476
TDAConnectionSSLOptions Class	476
Members	476
Properties	477
CACert Property	478
Cert Property	478
CipherList Property	478
Key Property	479
TDADataSetOptions Class	479
Members	480
Properties	482
AutoPrepare Property	485
CacheCalcFields Property	486
CompressBlobMode Property	486
DefaultValues Property	487
DetailDelay Property	487
FieldsOrigin Property	488
FlatBuffers Property	488
InsertAllSetFields Property	489
LocalMasterDetail Property	489
LongStrings Property	490
MasterFieldsNullable Property	490
NumberRange Property	491
QueryRecCount Property	491
QuoteNames Property	492
RemoveOnRefresh Property	492
RequiredFields Property	493
ReturnParams Property	493
SetFieldsReadOnly Property	494
StrictUpdate Property	494
TrimFixedChar Property	495
UpdateAllFields Property	495
UpdateBatchSize Property	496
TDAEncryption Class	496
Members	497
Properties	497
Encryptor Property	498
Fields Property	498
TDAMapRule Class	499
Members	499
Properties	500
DBLengthMax Property	501
DBLengthMin Property	502
DBScaleMax Property	502
DBScaleMin Property	503
DBType Property	503
FieldLength Property	504
FieldName Property	504

FieldScale Property.....	505
FieldType Property.....	505
IgnoreErrors Property.....	505
TDAMapRules Class.....	506
Members	506
Properties	507
IgnoreInvalidRules Property.....	507
TDAMetaData Class.....	508
Members	509
Properties	512
Connection Property.....	514
MetaDataKind Property.....	514
Restrictions Property.....	515
Methods	516
GetMetaDataKinds Method.....	518
GetRestrictions Method.....	519
TDAParam Class.....	520
Members	520
Properties	522
AsBlob Property.....	524
AsBlobRef Property.....	524
AsFloat Property.....	525
AsInteger Property.....	525
AsLargeInt Property.....	526
AsMemo Property.....	526
AsMemoRef Property.....	527
AsSQLTimeStamp Property.....	527
AsString Property.....	528
AsWideString Property.....	528
DataType Property.....	529
IsNull Property.....	529
ParamType Property.....	530
Size Property	530
Value Property.....	531
Methods	531
AssignField Method.....	532
AssignFieldValue Method.....	532
LoadFromFile Method.....	533
LoadFromStream Method.....	534
SetBlobData Method.....	534
SetBlobData Method.....	535
SetBlobData Method.....	535
TDAParams Class.....	536
Members	536
Properties	537
Items Property(Indexer).....	537
Methods	538
FindParam Method.....	539
ParamByName Method.....	539
TDATransaction Class.....	540
Members	541
Properties	542
Active Property.....	542
DefaultCloseAction Property.....	543

Methods	543
Commit Method	544
Rollback Method	545
StartTransaction Method	545
Events	546
OnCommit Event	547
OnCommitRetaining Event	547
OnError Event	548
OnRollback Event	549
OnRollbackRetaining Event	549
TMacro Class	550
Members	551
Properties	551
Active Property	552
AsDateTime Property	553
AsFloat Property	553
AsInteger Property	554
AsString Property	554
Name Property	555
Value Property	555
TMacros Class	555
Members	556
Properties	557
Items Property(Indexer)	557
Methods	558
AssignValues Method	559
Expand Method	559
FindMacro Method	560
IsEqual Method	560
MacroByName Method	561
Scan Method	562
TPoolingOptions Class	562
Members	562
Properties	563
ConnectionLifetime Property	564
MaxPoolSize Property	565
MinPoolSize Property	565
PoolId Property	566
Validate Property	566
TSmartFetchOptions Class	566
Members	567
Properties	567
Enabled Property	568
LiveBlock Property	568
PrefetchedFields Property	569
SQLGetKeyValues Property	569
Types	570
TAfterExecuteEvent Procedure Reference	571
TAfterFetchEvent Procedure Reference	571
TBeforeFetchEvent Procedure Reference	572
TConnectionLostEvent Procedure Reference	572
TDAConnectionErrorEvent Procedure Reference	573
TDATransactionErrorEvent Procedure Reference	573
TRefreshOptions Set	574

TUpdateExecuteEvent Procedure Reference.....	574
Enumerations	575
TLabelSet Enumeration.....	575
TLockMode Enumeration.....	576
TRefreshOption Enumeration.....	577
TRetryMode Enumeration.....	577
Variables	578
BaseSQLOldBehavior Variable.....	578
ChangeCursor Variable.....	579
SQLGeneratorCompatibility Variable.....	579
12 MemData	580
Classes	581
TBlob Class.....	582
Members	583
Properties	584
AsString Property.....	585
AsWideString Property.....	585
IsUnicode Property.....	586
Size Property	586
Methods	587
Assign Method.....	588
Clear Method	588
LoadFromFile Method.....	589
LoadFromStream Method.....	589
Read Method	590
SaveToFile Method.....	591
SaveToStream Method.....	592
Truncate Method.....	592
Write Method	593
TCompressedBlob Class.....	594
Members	595
Properties	596
Compressed Property.....	597
CompressedSize Property.....	598
TDBObject Class.....	598
Members	599
TMemData Class.....	599
Members	600
TObjectType Class.....	600
Members	600
Properties	601
AttributeCount Property.....	602
Attributes Property(Indexer).....	602
DataType Property.....	603
Size Property	603
Methods	604
FindAttribute Method.....	605
TSharedObject Class.....	605
Members	606
Properties	607
RefCount Property.....	607
Methods	608
AddRef Method.....	608
Release Method.....	609

Types	609
TLocateExOptions Set	610
TUpdateRecKinds Set	610
Enumerations	610
TCompressBlobMode Enumeration	611
TConnLostCause Enumeration	612
TDANumericType Enumeration	613
TLocateExOption Enumeration	613
TSortType Enumeration	614
TUpdateRecKind Enumeration	615
13 MemDS	615
Classes	616
TMemDataSet Class	616
Members	617
Properties	620
CachedUpdates Property	621
IndexFieldNames Property	622
KeyExclusive Property	624
LocalConstraints Property	624
LocalUpdate Property	625
Prepared Property	625
Ranged Property	626
UpdateRecordTypes Property	626
UpdatesPending Property	627
Methods	628
ApplyRange Method	630
ApplyUpdates Method	631
ApplyUpdates Method	631
ApplyUpdates Method	633
CancelRange Method	633
CancelUpdates Method	634
CommitUpdates Method	635
DeferredPost Method	636
EditRangeEnd Method	636
EditRangeStart Method	637
GetBlob Method	638
GetBlob Method	638
GetBlob Method	639
Locate Method	640
Locate Method	640
Locate Method	641
LocateEx Method	642
LocateEx Method	643
LocateEx Method	643
Prepare Method	644
RestoreUpdates Method	645
RevertRecord Method	646
SaveToXML Method	646
SaveToXML Method	647
SaveToXML Method	648
SetRange Method	648
SetRangeEnd Method	650
SetRangeStart Method	650
UnPrepare Method	651

UpdateResult Method.....	652
UpdateStatus Method.....	653
Events	653
OnUpdateError Event.....	654
OnUpdateRecord Event.....	655
Variables	656
DoNotRaiseExcetionOnUaFail Variable.....	656
SendDataSetChangeEventAfterOpen Variable.....	657
14 PgAccess	657
Classes	660
TCustomPgDataSet Class.....	662
Members	663
Properties	672
Cursor Property.....	676
DMLRefresh Property.....	677
FetchAll Property.....	677
KeySequence Property.....	678
LastInsertOID Property.....	679
Options Property.....	679
Params Property.....	681
SequenceMode Property.....	681
UpdateObject Property.....	682
Methods	682
CreateProcCall Method.....	687
FindParam Method.....	688
GetPgCursor Method.....	688
GetPgDate Method.....	689
GetPgInterval Method.....	689
GetPgLargeObject Method.....	690
GetPgRow Method.....	691
GetPgTime Method.....	691
GetPgTimeStamp Method.....	692
OpenNext Method.....	693
ParamByName Method.....	693
TCustomPgStoredProc Class.....	694
Members	695
Properties	704
Overload Property.....	708
Methods	709
ExecProc Method.....	713
PrepareSQL Method.....	714
TCustomPgTable Class.....	714
Members	715
Properties	724
Limit Property	729
Offset Property.....	729
TCustomPgTimeStampField Class.....	730
Members	730
Properties	731
AsPgTimeStamp Property.....	731
TPgConnection Class.....	732
Members	732
Properties	736
ConnectionTimeout Property.....	738

Database Property.....	739
Options Property.....	740
Password Property.....	741
Port Property	742
ProcessID Property.....	742
ProtocolVersion Property.....	743
Schema Property.....	743
Server Property.....	744
ServerVersion Property.....	744
ServerVersionFull Property.....	745
SSLOptions Property.....	745
Username Property.....	746
Methods	747
BreakExec Method.....	749
CreateDataSet Method.....	749
CreateMetaData Method.....	750
CreateSQL Method.....	750
CreateTransaction Method.....	751
GetRow Type Method.....	751
GetRow Type Method.....	752
GetRow Type Method.....	752
ReleaseSavepoint Method.....	753
RollbackToSavepoint Method.....	754
Savepoint Method.....	754
StartTransaction Method.....	755
StartTransaction Method.....	755
StartTransaction Method.....	756
Events	757
OnNotice Event.....	758
OnNotification Event.....	758
TPgConnectionOptions Class.....	759
Members	759
Properties	761
ApplicationName Property.....	764
Charset Property.....	764
EnableBCD Property.....	765
EnableComposites Property.....	765
EnableDomains Property.....	766
EnableFMTBCD Property.....	766
EnableGeometrics Property.....	767
EnablePgTimeStamps Property.....	767
ImmediateNotices Property.....	768
IPVersion Property.....	768
MessagesCharset Property.....	769
MultipleConnections Property.....	770
UseUnicode Property.....	770
TPgConnectionSSLOptions Class.....	771
Members	771
Properties	772
Mode Property.....	772
TPgCursorField Class.....	773
Members	773
Properties	774
AsCursor Property.....	774

TPgDataSetOptions Class	775
Members	775
Properties	779
AutoDeleteBlob Property	783
CacheBlobs Property	784
CursorWithHold Property	784
DeferredBlobRead Property	785
DistinctParams Property	785
EnableBCD Property	786
EnableFMTBCD Property	786
ExtendedFieldsInfo Property	787
FullRefresh Property	787
OIDAsInt Property	788
PrefetchRows Property	789
PrepareUpdateSQL Property	789
SetEmptyStrToNull Property	790
UnknownAsString Property	790
UnpreparedExecute Property	791
UseParamTypes Property	791
TPgDataSource Class	792
Members	792
TPgDateField Class	792
Members	793
Properties	793
AsPgDate Property	794
TPgEncryptor Class	795
Members	795
TPgGeometricField Class	796
Members	796
Properties	797
AsPgGeometric Property	797
TPgIntervalField Class	798
Members	798
Properties	798
AsPgInterval Property	799
TPgLargeObject Class	799
Members	800
Properties	802
Connection Property	803
TPgLargeObjectField Class	803
Members	804
Properties	804
AsLargeObject Property	805
TPgMetaData Class	805
Members	806
TPgParam Class	809
Members	810
Properties	812
AsPgDate Property	814
AsPgInterval Property	814
AsPgTime Property	815
AsPgTimeStamp Property	815
TPgParams Class	816
Members	817

Properties	817
Items Property(Indexer).....	818
TPgQuery Class.....	818
Members	820
Properties	829
FetchAll Property.....	833
LockMode Property.....	834
UpdatingTable Property.....	835
TPgSQL Class.....	836
Members	836
Properties	839
CommandTimeout Property.....	841
Connection Property.....	841
LastInsertOID Property.....	842
Params Property.....	842
UnpreparedExecute Property.....	843
UseParamTypes Property.....	843
Methods	844
FindParam Method.....	845
ParamByName Method.....	845
TPgStoredProc Class.....	846
Members	847
Properties	856
CommandTimeout Property.....	861
LockMode Property.....	861
StoredProcName Property.....	862
TPgTable Class.....	862
Members	863
Properties	873
FetchAll Property.....	877
LockMode Property.....	878
OrderFields Property.....	879
TableName Property.....	879
TPgTimeField Class.....	880
Members	880
Properties	881
AsPgTime Property.....	881
TPgTimeStampField Class.....	882
Members	882
Properties	883
AsPgTimeStamp Property.....	883
TPgUpdateSQL Class.....	884
Members	884
Types	885
TPgNoticeEvent Procedure Reference.....	886
TPgNotificationEvent Procedure Reference.....	886
Enumerations	887
TPgIsolationLevel Enumeration.....	887
Constants	888
PgDACVersion Constant.....	888
15 PgAlerter	889
Classes	889
TPgAlerter Class.....	889
Members	890

Properties	891
Events Property	892
Methods	893
SendEvent Method	893
SendEvent Method	893
SendEvent Method	894
Events	895
OnEvent Event	895
16 PgClasses	896
Classes	896
TPgCursor Class	897
Members	897
Properties	898
State Property	898
Enumerations	899
TProtocolVersion Enumeration	899
TSSLMode Enumeration	900
17 PgConnectionPool	901
Classes	901
TPgConnectionPoolManager Class	902
Members	902
18 PgDacVcl	902
Classes	903
TPgConnectDialog Class	903
Members	904
Properties	905
Connection Property	907
DatabaseLabel Property	907
PortLabel Property	908
Show Database Property	908
Show Port Property	909
19 PgDataTypeMap	910
Constants	911
pgBigInt Constant	912
pgBigSerial Constant	913
pgBit Constant	913
pgBitVarying Constant	914
pgBoolean Constant	914
pgBytea Constant	915
pgCharacter Constant	915
pgCharacterVarying Constant	915
pgDate Constant	916
pgDoublePrecision Constant	916
pgInteger Constant	917
pgMoney Constant	917
pgNumeric Constant	918
pgReal Constant	918
pgSerial Constant	919
pgSmallint Constant	919
pgText Constant	919
pgTime Constant	920
pgTimeStamp Constant	920
pgTimeStampWithTimeZone Constant	921

pgTimeWithTimeZone Constant.....	921
pgUUID Constant.....	922
20 PgDump	922
Classes	923
TPgDump Class.....	923
Members	924
Properties	926
Mode Property.....	927
ObjectTypes Property.....	927
Options Property.....	928
SchemaNames Property	929
TPgDumpOptions Class.....	929
Members	930
Properties	930
CreateConstraints Property.....	931
Types	931
TPgDumpObjects Set.....	932
Enumerations	932
TPgDumpMode Enumeration.....	933
TPgDumpObject Enumeration.....	933
21 PgError	934
Classes	934
EPgError Class.....	935
Members	935
Properties	936
CallStack Property.....	937
DetailMsg Property.....	938
ErrorCode Property.....	938
FileName Property.....	939
Hint Property	939
LineNumber Property.....	939
Position Property.....	940
ProcedureName Property.....	940
Severity Property.....	941
Enumerations	941
TPgSeverity Enumeration.....	942
22 PgLoader	942
Classes	943
TPgLoader Class.....	943
Members	944
Properties	945
BufferSize Property.....	946
Columns Property.....	947
Connection Property.....	947
Options Property.....	947
TableName Property.....	948
TextMode Property.....	948
Events	949
OnGetColumnData Event.....	949
OnPutData Event.....	950
TPgLoaderColumn Class.....	950
Members	951
Properties	951

Row TypeName Property.....	952
TPgLoaderOptions Class.....	952
Members	953
Properties	954
BufferSize Property.....	954
QuoteNames Property.....	955
TextMode Property.....	955
UseBlankValues Property.....	956
23 PgObjects	956
Classes	958
TCustomPgTimeStamp Class.....	959
Members	960
Properties	961
AsDateTime Property.....	962
AsSQLTimeStamp Property.....	963
Days Property.....	963
HasTimeZone Property.....	964
IsInfinity Property.....	964
IsNegInfinity Property.....	965
IsPosInfinity Property.....	965
Ticks Property.....	966
TimeZoneOffset Property.....	966
Methods	967
Assign Method.....	968
Compare Method.....	968
DecodeDate Method.....	969
DecodeDateTime Method.....	970
DecodeTime Method.....	971
EncodeDate Method.....	971
EncodeDateTime Method.....	972
EncodeTime Method.....	973
TPgAttribute Class.....	974
Members	974
TPgBox Class.....	974
Members	975
Properties	976
LowerLeft Property.....	976
UpperRight Property.....	977
TPgCircle Class.....	977
Members	978
Properties	978
Center Property.....	979
Radius Property.....	979
TPgDate Class.....	980
Members	980
TPgGeometric Class.....	982
Members	982
Methods	983
Assign Method.....	983
TPgInterval Class.....	984
Members	984
Properties	985
Days Property.....	986
MonthsFull Property.....	986

SecondsFull Property.....	987
Methods	987
Assign Method.....	988
Compare Method.....	988
DecodeInterval Method.....	989
EncodeInterval Method.....	990
TPgLseg Class.....	991
Members	991
Properties	992
EndPoint Property.....	992
StartPoint Property.....	993
TPgPath Class.....	993
Members	994
Properties	994
Count Property.....	995
IsClosedPath Property.....	995
Points Property.....	996
TPgPoint Class.....	996
Members	997
Properties	998
X Property	998
Y Property	999
Methods	999
Assign Method.....	1000
TPgPointsArray Class.....	1000
Members	1001
TPgPolygon Class.....	1001
Members	1002
Properties	1002
Count Property.....	1003
Points Property.....	1003
TPgRefCursor Class.....	1004
Members	1004
Properties	1005
CursorName Property.....	1006
TPgRow Class.....	1006
Members	1007
Properties	1009
AsString Property.....	1010
AttrAsPgBox Property(Indexer).....	1011
AttrAsPgCircle Property(Indexer).....	1011
AttrAsPgCursor Property(Indexer).....	1012
AttrAsPgDate Property(Indexer).....	1012
AttrAsPgInterval Property(Indexer).....	1013
AttrAsPgLargObject Property(Indexer).....	1013
AttrAsPgLseg Property(Indexer).....	1014
AttrAsPgPath Property(Indexer).....	1014
AttrAsPgPoint Property(Indexer).....	1015
AttrAsPgPolygon Property(Indexer).....	1016
AttrAsPgRow Property(Indexer).....	1016
AttrAsPgTime Property(Indexer).....	1017
AttrAsPgTimeStamp Property(Indexer).....	1017
AttrIsNull Property(Indexer).....	1018
AttrValue Property(Indexer).....	1018

Row Type Property.....	1019
Methods	1019
Assign Method.....	1020
TPgRow Type Class.....	1020
Members	1021
Properties	1022
BufferSize Property.....	1023
TPgSQLLargeObject Class.....	1024
Members	1024
Properties	1026
Cached Property.....	1028
Connection Property.....	1028
OID Property	1029
Methods	1029
CloseObject Method.....	1031
CreateObject Method.....	1031
OpenObject Method.....	1031
ReadBlob Method.....	1032
UnlinkObject Method.....	1032
WriteBlob Method.....	1033
TPgTime Class.....	1033
Members	1034
Properties	1035
TimeZoneOffset Property.....	1036
TPgTimeStamp Class.....	1037
Members	1038
TPgType Class.....	1039
Members	1040
Properties	1041
BaseTypeOID Property.....	1042
IsDescribed Property.....	1042
TableOID Property.....	1043
TypeOID Property.....	1043
Methods	1044
Describe Method.....	1044
Describe Method.....	1045
Describe Method.....	1045
24 PgScript	1046
Classes	1046
TPgScript Class.....	1046
Members	1047
Properties	1049
Connection Property.....	1050
25 PgSQLMonitor	1051
Classes	1051
TPgSQLMonitor Class.....	1051
Members	1052
26 PgTransaction	1053
Classes	1053
TPgTransaction Class.....	1053
Members	1054
Properties	1055
IsolationLevel Property.....	1056

27 VirtualDataSet	1057
Classes	1058
TCustomVirtualDataSet Class	1058
Members	1058
TVirtualDataSet Class	1061
Members	1062
Types	1065
TOnDeleteRecordEvent Procedure Reference	1066
TOnGetFieldValueEvent Procedure Reference	1066
TOnGetRecordCountEvent Procedure Reference	1067
TOnModifyRecordEvent Procedure Reference	1067
28 VirtualTable	1068
Classes	1068
TVirtualTable Class	1068
Members	1069
Properties	1072
DefaultSortType Property	1073
Methods	1074
Assign Method	1076
LoadFromFile Method	1077
LoadFromStream Method	1078

1 What's New

New Features in PgDAC 8.2

- Added support for RAD Studio 12 Athens Release 1
- Added support for Lazarus 3.2
- Improved work with generated fields

New Features in PgDAC 8.1

- Added support for Lazarus 3.0
- Improved work with column default values
- Improved work with RefreshRecord queries
- Improved work with arrays in queries

New Features in PgDAC 8.0

- Added support for RAD Studio 12
- Added support for PostgreSQL 16
- Added support for macOS Sonoma
- Added support for iOS 17
- Added support for Android 13
- Added support PREPARE/EXECUTE commands
- Added support for nested Macros in SQL queries
- Added support Display Format for Aggregate fields
- Added SHA-2(SHA-256, SHA-512) in hash algorithm for encryption

New Features in PgDAC 7.3

- Added support for RAD Studio 11 Alexandria Release 2
- Added support for Lazarus 2.2.2
- Added support for iOS 15
- Added support for Android 12

- Added support for PostGIS
- Added the CloneCursor method for Query and Table components that allows sharing data between datasets
- Improved TimeZoneOffset support for TPgTimeStamp
- Improved the Truncate method of the TPgLargeObject class
- Improved the performance of exporting to XML
- Fixed bug with executing a SELECT statement via the Execute method of the Query component
- Fixed bug with the Ping method of the Connection component when the MultipleConnections property is set to False
- Fixed bug with the "Record was changed by another user" exception when editing a dataset that contains double precision fields
- Fixed bug with "Invalid TimeStamp string" when the date has YYYY-MM-DD format
- Fixed bug when a connection string parameter value contains a single quote
- Fixed bug with international characters in error messages

New Features in PgDAC 7.2

- RAD Studio 11 Alexandria Release 1 is supported
- Lazarus 2.2.0 is supported
- Windows 11 is supported
- macOS Monterey is supported
- The AddDelete property for the Dump component is added
- Dumping of stored procedures via the Dump component is added
- The SCRAM-SHA-256-PLUS authentication mechanism is supported
- "ON CONFLICT" in batch operations is supported
- Open connection performance is improved

New Features in PgDAC 7.1

- PostgreSQL 14 is supported

- OUT parameters in stored procedures for PostgreSQL 14 are supported
- The PoolId connection pool option is added

New Features in PgDAC 7.0

- RAD Studio 11 Alexandria is supported
- macOS ARM is supported
- Added demo project for FastReport FMX

New Features in PgDAC 6.4

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

New Features in PgDAC 6.3

- Lazarus 2.0.10 and FPC 3.2.0 are supported
- Performance of Batch Insert, Update, and Delete operations is improved

New Features in PgDAC 6.2

- RAD Studio 10.4 Sydney is supported
- Lazarus 2.0.8 is supported
- macOS 64-bit in Lazarus is supported
- The Line geometric type is supported

New Features in PgDAC 6.1

- Android 64-bit is supported
- PostgreSQL 12 is supported
- OpenSSL 1.1 library is supported
- Now Trial edition for macOS and Linux is fully functional

New Features in PgDAC 6.0

- macOS 64-bit is supported
- Release 2 for RAD Studio 10.3 Rio, Delphi 10.3 Rio, and C++Builder 10.3 Rio is now required
- TPgConnectionSSLOptions.IgnoreServerCertificateInsecurity property is added

New Features in PgDAC 5.4

- Lazarus 2.0.2 is supported
- The pmAuto value for the ProtocolVersion property is added
- Now ProtocolVersion is set to pmAuto by default, which significantly improved performance
- Possibility to use function calls in batch operations is added
- The mVerifyCA and smVerifyFull options for the SSLOptions.Mode property of the TPgConnection component are added
- The DefaultSortType property for TVirtualTable is added
- Performance of the SaveToFile/LoadFromFile methods of TVirtualTable is significantly increased

New Features in PgDAC 5.3

- RAD Studio 10.3 Rio is supported
- PostgreSQL 11 is supported
- Support of UPPER and LOWER functions for Unified SQL is added

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

New Features in PgDAC 5.1

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

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

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

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

New Features in PgDAC 4.5

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

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

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

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

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

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

New Features in PgDAC 3.6

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

New Features in PgDAC 3.5

- Rad Studio XE3 is supported
- Windows 8 is supported

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

New Features in PostgreSQL Data Access Components 3.00

- Embarcadero RAD Studio XE2 is supported
- Application development for 64-bit Windows is supported
- FireMonkey application development platform is supported
- Support of master/detail relationship for TVirtualTable is added
- OnProgress event in TVirtualTable is added

- TDADatasetOptions.SetEmptyStrToNull property that allows inserting NULL value instead of empty string is added

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)

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

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 TDATransaction
- 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

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

New Features in PostgreSQL Data Access Components 1.10

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2 General Information

This section contains general information about PostgreSQL Server Data Access Components

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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 [Features](#).

Native Connection Options

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

Optimized Code

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

Compatibility with other Connectivity Methods

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

Development and Support

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

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

Key Features

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

- Direct access to server data without using client library. Does not require installation of other data provider layers (such as BDE and ODBC)
- Full support of [the latest versions of PostgreSQL Server](#)

- Support for all PostgreSQL Server data types
- Disconnected Model with automatic connection control for working with data offline
- Local Failover for detecting connection loss and implicitly reexecuting certain operations
- All types of local sorting and filtering, including by calculated and lookup fields
- [Automatic data updating](#) with [TPgQuery](#), [TPgTable](#), and [TPgStoredProc](#) components
- Unicode and national charset support
- Supports many PostgreSQL-specific features, such as notifications, notices, and sequences
- Advanced script execution functionality with the [TPgScript](#) component
- Support for [using macros](#) in SQL
- Lets you use Professional Edition of [Delphi and C++Builder](#) to develop client/server applications
- Includes annual [PgDAC Subscription](#) with [Priority Support](#)
- Licensed royalty-free per developer, per team, or per site

The full list of PgDAC features is available in [Features](#).

How does PgDAC work?

PgDAC connects to PostgreSQL directly without using client software.

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

PgDAC Connection



BDE Connection



© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.2 Features

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

Supported target platforms

- Windows, 32-bit and 64-bit
- macOS 64-bit
- Mac ARM
- iOS 64-bit
- iOS Simulator ARM 64-bit
- Android 32-bit and 64-bit

- Linux 32-bit (only in Lazarus and Free Pascal) and 64-bit

General usability:

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

Network and connectivity:

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

Compatibility:

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

PostgreSQL Server technology support:

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

PostgreSQL DataTypes:

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

Performance:

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

Local data storage operations:

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

Data access and data management automation:

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

Extended data access functionality:

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

Data exchange:

- Transferring data between all types of TDataSet descendants with [TCRBatchMove](#)

component

- Data [export](#) and [import](#) to/from XML (ADO format)
- Ability to [synchronize positions in different DataSets](#)
- Extended data management with [TPgDump](#) component

Script execution:

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

SQL execution monitoring:

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

Visual extensions:

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

Design-time enhancements:

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

Resources:

- Code documentation and guides in the CHM, PDF, and HXS formats
- Many helpful [demo](#) projects

Licensing and support:

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

© 1997-2024

Devert. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.3 Requirements

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

© 1997-2024

Devert. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.4 Compatibility

PostgreSQL Compatibility

PgDAC supports PostgreSQL server versions from 8.0 to 16.

Microsoft Azure Database for PostgreSQL Compatibility

PgDAC supports Microsoft Azure Database for PostgreSQL.

Amazon RDS for PostgreSQL Compatibility

PgDAC supports Amazon RDS for PostgreSQL and Amazon Aurora.

Google Cloud for PostgreSQL Compatibility

PgDAC supports Google Cloud for PostgreSQL.

Heroku Postgres Compatibility

PgDAC supports Heroku Postgres.

IDE Compatibility

PgDAC is compatible with the following IDEs:

Embarcadero RAD Studio 12.1 Athens

- Embarcadero Delphi 12.1 Athens for Windows
- Embarcadero Delphi 12.1 Athens for macOS
- Embarcadero Delphi 12.1 Athens for Linux
- Embarcadero Delphi 12.1 Athens for iOS
- Embarcadero Delphi 12.1 Athens for Android
- Embarcadero C++Builder 12.1 Athens for Windows
- Embarcadero C++Builder 12.1 Athens for iOS
- Embarcadero C++Builder 12.1 Athens for Android

Embarcadero RAD Studio 12 Athens

- Embarcadero Delphi 12 Athens for Windows
- Embarcadero Delphi 12 Athens for macOS
- Embarcadero Delphi 12 Athens for Linux
- Embarcadero Delphi 12 Athens for iOS
- Embarcadero Delphi 12 Athens for Android
- Embarcadero C++Builder 12 Athens for Windows
- Embarcadero C++Builder 12 Athens for iOS
- Embarcadero C++Builder 12 Athens for Android

Embarcadero RAD Studio 11.1 Alexandria

- Embarcadero Delphi 11.1 Alexandria for Windows
- Embarcadero Delphi 11.1 Alexandria for macOS
- Embarcadero Delphi 11.1 Alexandria for Linux
- Embarcadero Delphi 11.1 Alexandria for iOS
- Embarcadero Delphi 11.1 Alexandria for Android
- Embarcadero C++Builder 11.1 Alexandria for Windows
- Embarcadero C++Builder 11.1 Alexandria for iOS
- Embarcadero C++Builder 11.1 Alexandria for Android

Embarcadero RAD Studio 10.4 Sydney (Requires Release 1 or Release 2)

- Embarcadero Delphi 10.4 Sydney for Windows
- Embarcadero Delphi 10.4 Sydney for macOS
- Embarcadero Delphi 10.4 Sydney for Linux
- Embarcadero Delphi 10.4 Sydney for iOS
- Embarcadero Delphi 10.4 Sydney for Android
- Embarcadero C++Builder 10.4 Sydney for Windows
- Embarcadero C++Builder 10.4 Sydney for iOS
- Embarcadero C++Builder 10.4 Sydney for Android

Embarcadero RAD Studio 10.3 Rio (Requires [Release 2](#) or [Release 3](#))

- Embarcadero Delphi 10.3 Rio for Windows
- Embarcadero Delphi 10.3 Rio for macOS
- Embarcadero Delphi 10.3 Rio for Linux
- Embarcadero Delphi 10.3 Rio for iOS
- Embarcadero Delphi 10.3 Rio for Android
- Embarcadero C++Builder 10.3 Rio for Windows
- Embarcadero C++Builder 10.3 Rio for macOS
- Embarcadero C++Builder 10.3 Rio for iOS
- Embarcadero C++Builder 10.3 Rio for Android

Embarcadero RAD Studio 10.2 Tokyo (Incompatible with Release 1)

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

- Embarcadero C++Builder 10.2 Tokyo for Android

Embarcadero RAD Studio 10.1 Berlin

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

Embarcadero RAD Studio 10 Seattle

- Embarcadero Delphi 10 Seattle for Windows
- Embarcadero Delphi 10 Seattle for macOS
- Embarcadero Delphi 10 Seattle for iOS
- Embarcadero Delphi 10 Seattle for Android
- Embarcadero C++Builder 10 Seattle for Windows
- Embarcadero C++Builder 10 Seattle for macOS
- Embarcadero C++Builder 10 Seattle for iOS
- Embarcadero C++Builder 10 Seattle for Android

Embarcadero RAD Studio XE8

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

Embarcadero RAD Studio XE7

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

Embarcadero RAD Studio XE6

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

Embarcadero RAD Studio XE5 (Requires [Update 2](#))

- Embarcadero Delphi XE5 for Windows
- Embarcadero Delphi XE5 for macOS
- Embarcadero Delphi XE5 for iOS
- Embarcadero Delphi XE5 for Android
- Embarcadero C++Builder XE5 for Windows
- Embarcadero C++Builder XE5 for macOS
- Embarcadero C++Builder XE5 for iOS

Embarcadero RAD Studio XE4

- Embarcadero Delphi XE4 for Windows

- Embarcadero Delphi XE4 for macOS
- Embarcadero Delphi XE4 for iOS
- Embarcadero C++Builder XE4 for Windows
- Embarcadero C++Builder XE4 for macOS

Embarcadero RAD Studio XE3 (Requires [Update 2](#))

- Embarcadero Delphi XE3 for Windows
- Embarcadero Delphi XE3 for macOS
- Embarcadero C++Builder XE3 for Windows
- Embarcadero C++Builder XE3 for macOS

Embarcadero RAD Studio XE2 (Requires [Update 4 Hotfix 1](#))

- Embarcadero Delphi XE2 for Windows
- Embarcadero Delphi XE2 for macOS
- Embarcadero C++Builder XE2 for Windows
- Embarcadero C++Builder XE2 for macOS

Embarcadero RAD Studio XE

- Embarcadero Delphi XE
- Embarcadero C++Builder XE

Embarcadero RAD Studio 2010

- Embarcadero Delphi 2010
- Embarcadero C++Builder 2010

CodeGear RAD Studio 2009 (Requires [Update 3](#))

- CodeGear Delphi 2009
- CodeGear C++Builder 2009

CodeGear RAD Studio 2007

- CodeGear Delphi 2007
- CodeGear C++Builder 2007

Borland Developer Studio 2006

- Borland Delphi 2006
- Borland C++Builder 2006

Borland Delphi 7

Borland Delphi 6 (Requires [Update Pack 2](#) – Delphi 6 Build 6.240)
Borland C++Builder 6 (Requires [Update Pack 4](#) – C++Builder 6 Build 10.166)
[Lazarus](#) 3.2.0 and [Free Pascal](#) 3.2.2 for Windows, macOS, and Linux.

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

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

Supported Target Platforms

- Windows 32-bit and 64-bit
- macOS 64-bit and ARM (Apple Silicon M1)
- Linux 32-bit (only in Lazarus and Free Pascal) and 64-bit
- iOS 64-bit
- iOS Simulator ARM 64-bit
- Android 32-bit and 64-bit

Support for Windows 64-bit is available since RAD Studio XE2. Support for iOS 64-bit is available since RAD Studio XE8. Support for Android 32-bit is available since RAD Studio XE5. Support for Linux 64-bit is available since RAD Studio 10.2 Tokyo. Support for macOS 64-bit is available since RAD Studio 10.3 Rio. Support for Android 64-bit is available since RAD Studio 10.3.3 Rio.

Supported GUI Frameworks

- FireMonkey (FMX)
- Visual Component Library (VCL)
- Lazarus Component Library (LCL)

Devart Data Access Components Compatibility

All DAC products are compatible with each other.

But, to install several DAC products to the same IDE, it is necessary to make sure that all DAC products have the same common engine (BPL files) version. The latest versions of

DAC products or versions with the same release date always have the same version of the common engine and can be installed to the same IDE.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.5 Using Several DAC Products in One IDE

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

Packages:

- dacXX.bpl
- dacvclXX.bpl
- dcldacXX.bpl

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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
|       |-TCustomDACConnection
|           |-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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.8 Editions

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

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

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

You can get **Source Access** to PgDAC Professional Edition by purchasing a special PgDAC Professional Edition with Source Code, which includes the source code of all component

classes. The source code of DataSet Manager and Migration Wizard is not distributed.

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

PgDAC Edition Matrix

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 metadata 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	✓	✓	
Dataset that wraps arbitrary non-tabular data TVirtualDataSet	✓	✓	
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	×	SRC ¹	

¹ Available only in Professional Edition with Source Code.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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 a Trial version that can be distributed for free as original Devart's PgDAC Trial package.

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

2.7. You may not remove or alter any Devart's copyright, trademark, or other proprietary rights

notice contained in any portion of Devart units, source code, or other files that bear such a notice.

3. REDISTRIBUTION

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

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

4. TRANSFER

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

5. TERMINATION

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

6. EVALUATION

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

7. WARRANTY

The Software and documentation are provided "AS IS" without warranty of any kind. Devart

makes no warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or use.

8. SUBSCRIPTION AND SUPPORT

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

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

9. COPYRIGHT

The Software is confidential and proprietary copyrighted work of Devart and is protected by international copyright laws and treaty provisions. You may not remove the copyright notice from any copy of the Software or any copy of the written materials, accompanying the Software.

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

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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 determine which version of PgDAC I am using?

You can determine your PgDAC version number in several ways:

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

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

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

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

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

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

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

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

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

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

record syntax.

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

Use the `TPgConnection.GetDatabaseNames` method.

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

Use the `TPgConnection.GetTableNames` method.

6. Some questions about the visual part of PgDAC

The following situations usually arise from the same problem:

- 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 TCustomDADDataSet.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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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

C++Builder 2006, 2007:

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

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

Lazarus

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

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

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

DBMonitor

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.2 Connecting to PostgreSQL

This tutorial describes how to connect to PostgreSQL using the [TPgConnection](#) component.

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

Requirements

This tutorial assumes that you have installed PgDAC and run the database server and the

IDE. You need to know the server address, the port number (if you use a port other than the default port 5432), the database name, the schema, and the username and password. To connect at runtime, add the [PgAccess](#) unit to the `uses` clause for Delphi or include the `PgAccess.hpp` header file for C++ Builder.

General Information

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

Creating a Connection

Connecting at Design-Time

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

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

Using TPgConnection Editor

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

Using Object Inspector

1. Select the `PgConnection1` object on the form.
2. Set the `Database` property to the database name.
3. Set the `Password` property to the password (`postgres` by default).

4. If you use a port other than the default port 5432, set the `Port` property to the port.
5. Set `Schema` property to the database schema (`public` by default).
6. Set the `Server` property to the DNS name or IP address of the PostgreSQL server.
7. Set the `Username` property to the username (`postgres` by default).

Connecting at Runtime

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

Delphi

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

C++ Builder

```
TPgConnection* PgConnection1 = new TPgConnection(NULL);
try {
  // adds connection parameters
  PgConnection1->Server = "server";
  PgConnection1->Database = "database";
  PgConnection1->Username = "username";
  PgConnection1->Password = "password";
  PgConnection1->Port = 5432;
  // disables a login prompt
  PgConnection1->LoginPrompt = False;
  // opens a connection
  PgConnection1->Open();
}
finally {
  PgConnection1->Free();
}
```

Opening a Connection

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

Delphi

```
PgConnection1.Open;
```

C++Builder

```
PgConnection1->Open;
```

Another way to open a connection at runtime is to set the [Connected](#) property to `True`:

Delphi

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

C++ Builder

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

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

Modifying a Connection

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

Closing a Connection

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

Delphi

```
PgConnection1.Close;
```

or:

```
PgConnection1.Connected := False;
```

C++ Builder

```
PgConnection1->Close;
```

or:

```
PgConnection1->Connected = False;
```

Additional Information

PgDAC offers a wide set of features to achieve better performance, balance network load, and enable additional capabilities, for example:

- Local Failover
- Connection Pooling
- Disconnected Mode
- Support for Unicode
- Data Type Mapping

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.3 Deleting Data From Tables

This tutorial describes how to delete data from a table using the [TPgQuery](#) and [TPgTable](#) components.

1. [Requirements](#)
2. [General Information](#)
3. [Using the DataSet Functionality](#)

4. [Building DML Statements Manually](#)
 - 4.1 [Parameterized DML Statements](#)
 - 4.2 [Non-Parameterized DML Statements](#)
5. [Additional Information](#)

Requirements

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

General Information

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

Using the DataSet Functionality

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

Delphi

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

```
PgQuery1.Delete;  
finally  
    PgQuery1.Free;  
end;  
end;
```

C++Builder

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

Building DML Statements Manually

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

Parameterized DML Statements

Delphi

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

C++Builder

```
TIBCQuery* PgQuery1 = new TIBCQuery(NULL);  
try {
```

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

Non-Parameterized DML Statements

Delphi

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

C++Builder

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

Additional Information

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.4 Creating Database Objects

This tutorial describes how to create database objects in PostgreSQL using the [TPgSQL](#) and [TPgScript](#) components.

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

Requirements

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

General Information

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

Creating Tables

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

Design-Time

- Find the `TPgSQL` component in the PgDAC category on the Tool Palette.
- Double-click the component. A new object will appear on the form. If this is the first `TPgSQL` object in this project, it will be named `PgSQL1`. Note that the [Connection](#) property is automatically set to an existing connection.
- Double-click the `PgSQL1` object.
- Enter the following statements:

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

- Click the `Execute` button to create two tables.

Runtime

The same tables created at runtime:

Delphi

```
var  
  PgSQL1: TPgSQL;  
begin  
  PgSQL1 := TPgSQL.Create(nil);
```

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

C++Builder

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

Creating Stored Procedures

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

Design-Time

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

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

- Click the `Execute` button to create two stored procedures.

Runtime

The same stored procedures created at runtime:

Delphi

```
var  
    PgScript1: TPgScript;  
begin  
    PgScript1 := TPgScript.Create(nil);  
    try  
        // PgConnection1 was set up earlier  
        PgScript1.Connection := con;
```

```

//adds statements to create procedures
PgScript1.SQL.Add('CREATE FUNCTION TenMostHighPaidEmployees()');
PgScript1.SQL.Add('RETURNS SETOF Emp AS $$');
PgScript1.SQL.Add('  SELECT * FROM emp ORDER BY emp.sal DESC LIMIT 10');
PgScript1.SQL.Add('$$ LANGUAGE sql;');
PgScript1.SQL.Add('');
PgScript1.SQL.Add('CREATE FUNCTION GetEmpNumberInDept(');
PgScript1.SQL.Add('  IN pdeptno int,');
PgScript1.SQL.Add('  OUT pempnumb int)');
PgScript1.SQL.Add('RETURNS int AS $$');
PgScript1.SQL.Add('BEGIN');
PgScript1.SQL.Add('  pempnumb := (SELECT count(*) FROM emp WHERE deptno');
PgScript1.SQL.Add('END;');
PgScript1.SQL.Add(' $$ LANGUAGE plpgsql;');
// executes the statements
PgScript1.Execute;
finally
  PgScript1.Free;
end;
end;

```

C++Builder

```

TPgScript* PgScript1 = new TPgScript(NULL);
try {
  // PgConnection1 was set up earlier
  PgScript1->Connection = con;
  //adds statements to create procedures
  PgScript1->SQL->Add("CREATE FUNCTION TenMostHighPaidEmployees()");
  PgScript1->SQL->Add("RETURNS SETOF Emp AS $$");
  PgScript1->SQL->Add("  SELECT * FROM emp ORDER BY emp.sal DESC LIMIT 10");
  PgScript1->SQL->Add("$$ LANGUAGE sql;");
  PgScript1->SQL->Add("");
  PgScript1->SQL->Add("CREATE FUNCTION GetEmpNumberInDept(");
  PgScript1->SQL->Add("  IN pdeptno int,");
  PgScript1->SQL->Add("  OUT pempnumb int)");
  PgScript1->SQL->Add("RETURNS int AS $$");
  PgScript1->SQL->Add("BEGIN");
  PgScript1->SQL->Add("  pempnumb = (SELECT count(*) FROM emp WHERE deptno");
  PgScript1->SQL->Add("END;");
  PgScript1->SQL->Add(" $$ LANGUAGE plpgsql;");
  // executes the statements
  PgScript1->Execute;
}
finally {
  PgScript1->Free();
}

```

Additional Information

There are many ways to create database objects on the server. Any tool or component that is capable of running an SQL query can be used to manage database objects. For example, TPgSQL can be used to insert statements one by one, while TPgScript is intended to execute

multiple DDL/DML statements as a single SQL script. For more information on DDL statements, refer to the PostgreSQL documentation.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.5 Inserting Data Into Tables

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

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

Requirements

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

General Information

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

The dept table definition:

deptno	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

The emp table definition:

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	PRE SIDE NT	NULL	17- 11- 1981	5000	NULL	10
7844	TURN ER	SALE SMA N	7698	08- 09- 1981	1500	0	30
7876	ADA MS	CLE RK	7788	13- 07- 1987	1100	NULL	20
7900	JAM ES	CLE RK	7698	03- 12- 1981	950	NULL	30
7902	FOR D	ANAL YST	7566	03- 12- 1981	3000	NULL	20
7934	MILL ER	CLE RK	7782	23- 01- 1982	1300	NULL	10

Inserting at Design-Time

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

```
INSERT INTO dept VALUES (10, 'ACCOUNTING', 'NEW YORK');
```

- Click the `Execute` button to add a new record to the `dept` table.

Inserting at Runtime

Using the DataSet Functionality

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

demonstrates the use of these methods:

Delphi

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

C++Builder

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

Building DML Statements Manually

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

demonstrates both ways.

Parameterized DML Statements

Delphi

```

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

```

C++Builder

```

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

```

Non-Parameterized DML Statements

Delphi

```

var
  PgQuery1: TPgQuery;
begin
  PgQuery1 := TPgQuery.Create(nil);
  try

```

```
// PgConnection1 was set up earlier
PgQuery1.Connection := PgConnection1;
// adds a statement to insert a record
PgQuery1.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (10, 'ACCO
// executes the statement
PgQuery1.Execute;
finally
PgQuery1.Free;
end;
end;
```

C++Builder

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

Additional Information

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.6 Retrieving Data From Tables

This tutorial describes how to retrieve data from a table using the [TPgQuery](#) and [TPgTable](#) components.

1. [Requirements](#)
2. [General Information](#)

3. [TPgQuery](#)
4. [TPgTable](#)
5. [Additional Information](#)

Requirements

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

General Information

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

TPgQuery

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

Delphi

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

C++Builder

```
TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
```

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

TPgTable

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

Delphi

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

C++Builder

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

Additional Information

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.7 Modifying Data in Tables

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

1. [Requirements](#)
2. [General Information](#)
3. [Using the DataSet Functionality](#)
4. [Building DML Statements Manually](#)
 - 4.1 [Parameterized DML Statements](#)
 - 4.2 [Non-Parameterized DML Statements](#)
5. [Additional Information](#)

Requirements

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

General Information

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

10 ACCOUNTING NEW YORK

to change it to:

10 RESEARCH LOS ANGELES

Using the DataSet Functionality

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

Delphi

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

C++Builder

```
TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
  // PgConnection1 was set up earlier
  PgQuery1->Connection = PgConnection1;
  // adds a statement to retrieve data
  PgQuery1->SQL->Text = "SELECT * FROM dept";
  // opens the dataset
  PgQuery1->Open();
  // positions the cursor on the deptno=10 record
  PgQuery1->FindKey(ARRAYOFCONST((10)));
  // enables editing of data in the dataset
```

```

PgQuery1->Edit();
// searches fields by their names and assigns new values
PgQuery1->FieldByName("dname")->AsString = "RESEARCH";
PgQuery1->FieldByName("loc")->AsString = "LOS ANGELES";
// writes the modified record
PgQuery1->Post();
}
finally {
PgQuery1->Free();
}

```

Building DML Statements Manually

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

Parameterized DML Statements

Delphi

```

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

```

C++Builder

```

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

```

```
}  
__finally {  
    PgQuery1->Free();  
}
```

Non-Parameterized DML Statements

Delphi

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

C++Builder

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

Additional Information

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.8 Using Stored Procedures

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

1. [Requirements](#)
2. [General information](#)
3. [Input Parameters](#)
4. [Output Parameters](#)
5. [Input/Output Parameters](#)
6. [Using PostgreSQL Stored Functions](#)
7. [Returning Result Sets](#)

Requirements

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

General Information

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

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

Input parameters

`Input parameters` are used to pass values from the calling program to the stored procedure or

function. If the procedure changes the input value, the change has effect only within the procedure, and the input variable will preserve its original value when control is returned to the calling program. The following procedure inserts a new row into the table [dept](#):

```
CREATE PROCEDURE InsertDept(  
    p_deptno int,  
    p_dname varchar(14),  
    p_loc varchar(13)  
)  
AS $$  
    INSERT INTO dept(deptno, dname, loc) VALUES(:p_deptno, :p_dname, :p_loc);  
$$ LANGUAGE sql;
```

The procedure accepts three input arguments that correspond to the fields of the table, and can be executed as follows:

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

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

Delphi

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

C++Builder

```
TPgStoredProc* PgStoredProc1 = new TPgStoredProc(NULL);  
try {  
    // PgConnection1 was set up earlier  
    PgStoredProc1->Connection = PgConnection1;  
    // indicates the name of the stored procedure to call  
    PgStoredProc1->StoredProcName = "InsertDept";
```

```

// constructs a statement based on the Params and StoredProcName
// properties, and assigns it to the SQL property
PgStoredProc1->PrepareSQL();
// searches parameters by their names and assigns new values
PgStoredProc1->ParamByName("p_deptno")->AsInteger = 10;
PgStoredProc1->ParamByName("p_dname")->AsString = "ACCOUNTING";
PgStoredProc1->ParamByName("p_loc")->AsString = "NEW YORK";
// executes the stored procedure
PgStoredProc1->Execute();
}
finally {
    PgStoredProc1->Free();
}

```

Output Parameters

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

```

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

```

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

Delphi

```

var
    PgStoredProc1: TPgStoredProc;
begin
    PgStoredProc1 := TPgStoredProc.Create(nil);
    try
        // PgConnection1 was set up earlier
        PgStoredProc1.Connection := PgConnection1;
        // indicates the name of the stored procedure to call
        PgStoredProc1.StoredProcName := 'CountDept';
        // constructs a statement based on the Params and StoredProcName
        // properties, and assigns it to the SQL property
        PgStoredProc1.PrepareSQL();
        // executes the stored procedure
        PgStoredProc1.Execute();
        // shows the value of the output parameter
        ShowMessage(IntToStr(PgStoredProc1.ParamByName('result').AsString));
    finally
        PgStoredProc1.Free;
    end;
end;

```

C++Builder

```
TPGStoredProc* PGStoredProc1 = new TPGStoredProc(NULL);
try {
    // PGConnection1 was set up earlier
    PGStoredProc1->Connection = PGConnection1;
    // indicates the name of the stored procedure to call
    PGStoredProc1->StoredProcName = "CountDept";
    // constructs a statement based on the Params and StoredProcName
    // properties, and assigns it to the SQL property
    PGStoredProc1->PrepareSQL();
    // executes the stored procedure
    PGStoredProc1->Execute();
    // shows the value of the output parameter
    ShowMessage(IntToStr(sp->ParamByName("cnt")->AsInteger));
}
__finally {
    PGStoredProc1->Free();
}
```

Input/output parameters

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

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

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

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

Delphi

```
var
    PgStoredProc1: TPgStoredProc;
begin
    PgStoredProc1 := TPgStoredProc.Create(nil);
    try
        // PgConnection1 was set up earlier
        PgStoredProc1.Connection := PgConnection1;
        // indicates the name of the stored procedure to call
        PgStoredProc1.StoredProcName := 'GiveBonus';
        // constructs a statement based on the Params and StoredProcName
        // properties, and assigns it to the SQL property
```

```

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

```

C++Builder

```

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

```

Using PostgreSQL Stored Functions

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

```

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

```

This function can be executed as follows:

```

SELECT GiveBonusFunc(500.5);

```

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

Delphi

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

C++Builder

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

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

Returning Result Sets

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.9 Using Stored Functions with Result Sets

This tutorial describes how to retrieve and modify result sets obtained from stored functions using the [TPgStoredProc](#) component

Requirements

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

General Information

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

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

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

Using the DataSet Functionality

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

Delphi

```
var  
    PgStoredProc11: TPgStoredProc;  
begin
```

```
PgStoredProc1 := TPgStoredProc.Create(nil);
try
    // Pgconnection1 was set up earlier
    PgStoredProc1.Connection := PgConnection1;
    // indicates the name of the stored procedure to call
    PgStoredProc1.StoredProcName := 'selectDept';
    // constructs a statement based on the Params and StoredProcName
    // properties, and assigns it to the SQL property
    PgStoredProc1.PrepareSQL;
    // opens the dataset
    PgStoredProc1.Open;
    // adds a new empty record at the end of the dataset
    PgStoredProc1.Append;
    // searches fields by their names and assigns new values
    PgStoredProc1.FieldName('deptno').AsInteger := 50;
    PgStoredProc1.FieldName('dname').AsString := 'SALES';
    PgStoredProc1.FieldName('loc').AsString := 'NEW YORK';
    // writes the modified record
    PgStoredProc1.Post;
    // adds a new empty record in the current cursor position
    PgStoredProc1.Insert;
    PgStoredProc1.FieldName('deptno').AsInteger := 60;
    PgStoredProc1.FieldName('dname').AsString := 'ACCOUNTING';
    PgStoredProc1.FieldName('loc').AsString := 'LOS ANGELES';
    PgStoredProc1.Post;
    // positions the cursor on the deptno=10 record
    PgStoredProc1.FindKey([10]);
    // enables editing of data in the dataset
    PgStoredProc1.Edit;
    PgStoredProc1.FieldName('dname').AsString := 'RESEARCH';
    PgStoredProc1.FieldName('loc').AsString := 'LOS ANGELES';
    PgStoredProc1.Post;
finally
    PgStoredProc1.Free;
end;
end;
```

C++Builder

```
TPgStoredProc* PgStoredProc1 = new TPgStoredProc(NULL);
try {
    // PgConnection1 was set up earlier
    PgStoredProc1->Connection = PgConnection1;
    // indicates the name of the stored procedure to call
    PgStoredProc1->StoredProcName = "selectDept";
    // constructs a statement based on the Params and StoredProcName
    // properties, and assigns it to the SQL property
    PgStoredProc1->PrepareSQL();
    // opens the dataset
    PgStoredProc1->Open();
    // adds a new empty record at the end of the dataset
    PgStoredProc1->Append();
    // searches fields by their names and assigns new values
    PgStoredProc1->FieldName("deptno")->AsInteger = 50;
    PgStoredProc1->FieldName("dname")->AsString = "SALES";
    PgStoredProc1->FieldName("loc")->AsString = "NEW YORK";
    // writes the modified record
```

```
PgStoredProc1->Post();
// adds a new empty record in the current cursor position
PgStoredProc1->Insert();
PgStoredProc1->FieldByName("deptno")->AsInteger = 60;
PgStoredProc1->FieldByName("dname")->AsString = "ACCOUNTING";
PgStoredProc1->FieldByName("loc")->AsString = "LOS ANGELES";
PgStoredProc1->Post();
// positions the cursor on the deptno=10 record
PgStoredProc1->FindKey(ARRAYOFCONST((10)));
// enables editing of data in the dataset
PgStoredProc1->Edit();
PgStoredProc1->FieldByName("dname")->AsString = "RESEARCH";
PgStoredProc1->FieldByName("loc")->AsString = "LOS ANGELES";
PgStoredProc1->Post();
}
__finally {
    PgStoredProc1->Free();
}
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.10 Demo Projects

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

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

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

Where are the PgDAC demo projects located?

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

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

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

For most new IDEs the structure will be as follows.

Demos

```
|-PgDacDemo [The main PgDAC demo project]
|-TechnologySpecific
|  |- SecureBridge [A component and a demo for integration with
the SecureBridge library]
|-ThirdParty
|  |- [A collection of demo projects on integration with third-
party components]
|-Miscellaneous
    |- [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 PostgreSQL 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	<p>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.</p> <p>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.</p> <p>For more information see the <i>Readme.html</i> file in the demo directory.</p>
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>

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

3.11 Deployment

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

Deploying Windows applications built without run-time

packages

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

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

Trial Limitation Warning

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

dacXX.bpl	always
pgdacXX.bpl	always

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

rtlXX.bpl	always
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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.1 Connecting to Heroku Postgres

Connecting to Heroku Postgres using PostgreSQL Data Access Components

Heroku is a cloud-based multilingual platform-as-a-service (PaaS) that is designed for hosting applications and web services, working with loaded applications, reducing the need for complex work with the server, and rapid scaling of applications. It also simplifies and speeds up the development cycle.

Requirements

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

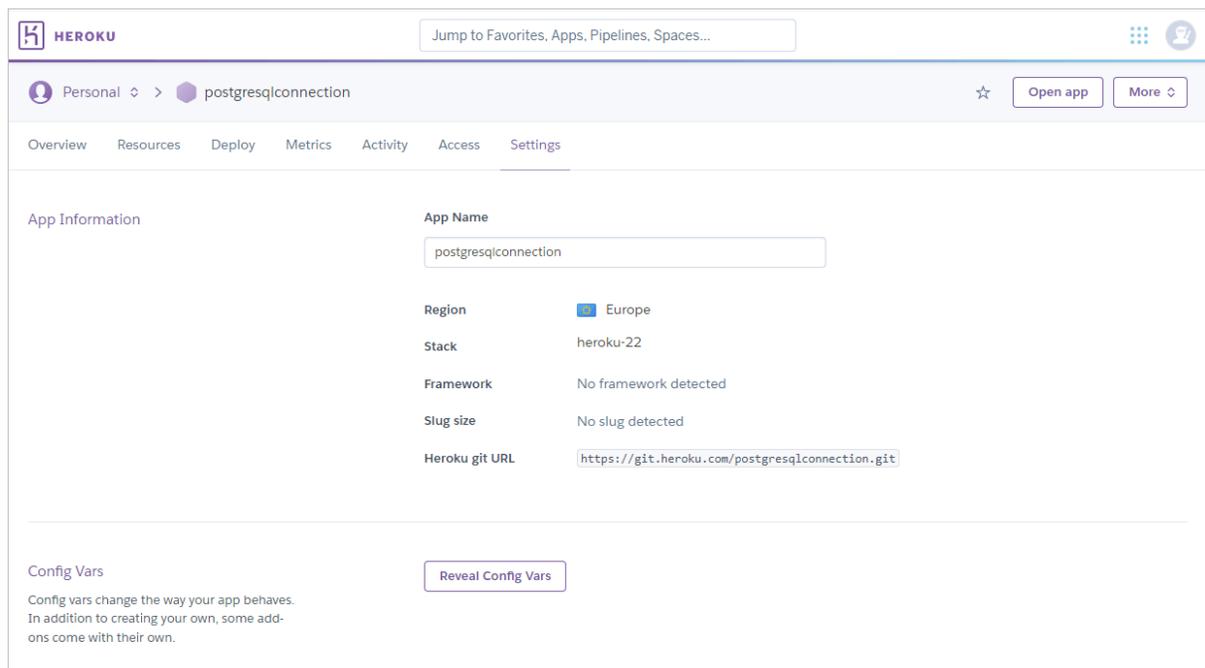
General Information

To establish a connection to your Heroku PostgreSQL database, set up the properties of the [TPgConnection](#) component: [Server](#), [Port](#), [Database](#), [Schema](#), [Username](#), and [Password](#). You can also specify all connection parameters in the [ConnectionString](#) property.

To connect to the Heroku Postgres, you need to get the URL of your database. You may get it through the **Heroku Dashboard**. This method is easier if you are using the platform for the first time.

1. Go to your application on Heroku you want to connect to.
2. Switch to the **Settings** tab and press the **Reveal Config Vars** button. The list of configuration variables will appear.
3. In the **Key** field, you will see the key value in the following format:

```
< postgres: //< username >: < password >@< hostname/server >/< databasename
```



Creating a Connection

Connecting at Design-Time

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

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

Using `TPgConnection` Editor

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

7. Specify the schema (`public` by default) in the `Schema` edit box.

Using Object Inspector

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

Connecting at Runtime

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

Delphi

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

C++ Builder

```
TPgConnection* PgConnection1 = new TPgConnection(NULL);
try {
  // adds connection parameters
  PgConnection1->Server = "server";
  PgConnection1->Database = "database";
  PgConnection1->Username = "username";
  PgConnection1->Password = "password";
  PgConnection1->Port = 5432;
```

```
// disables a login prompt
PgConnection1->LoginPrompt = False;
// opens a connection
PgConnection1->Open();
}
finally {
PgConnection1->Free();
}
```

Now, all you have to do is establish a connection to Heroku Postgres. It is assumed that you have installed the Heroku Postgres Data Access Components on your computer.

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

4.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.4 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.5 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, ftSma
// rule for numeric(10,0)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 5, 10, 0, 0, ftInt
// rule for numeric(15,0)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 11, r1Any, 0, 0, ftLar
// rule for numeric(5,2)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, 9, 1, r1Any, ftFlo
// 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, ftFMT

```

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.AddFieldNameRule('guid', ftGuid)
```

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

Ignoring conversion errors

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

For example:

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.6 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.AddFieldNameRule ('ename', ftString);  
PgQuery.DataTypeMap.AddFieldNameRule ('hiredate', ftDateTime);  
PgQuery.DataTypeMap.AddFieldNameRule ('sal', ftFloat);  
PgQuery.Open;
```

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.7 Working in an Unstable Network

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

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

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

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

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

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

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

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

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

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

their reexecuting yourself.

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

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.8 Secure Connections

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

- [Connecting via SSL](#)
- [Connecting via SSH](#)
- [HTTP/HTTPS Network Tunneling](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.8.1 Connecting via SSL

Connecting to PostgreSQL Through SSL in Delphi

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

Devart offers a solution called [SecureBridge](#), which allows you to embed an SSL client into a

Delphi or C++ Builder application to establish a secure connection to PostgreSQL server. This tutorial demonstrates how to create a sample Delphi application that connects to PostgreSQL using SSL as the encryption method.

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

Sample Delphi app that connects to PostgreSQL using SSL

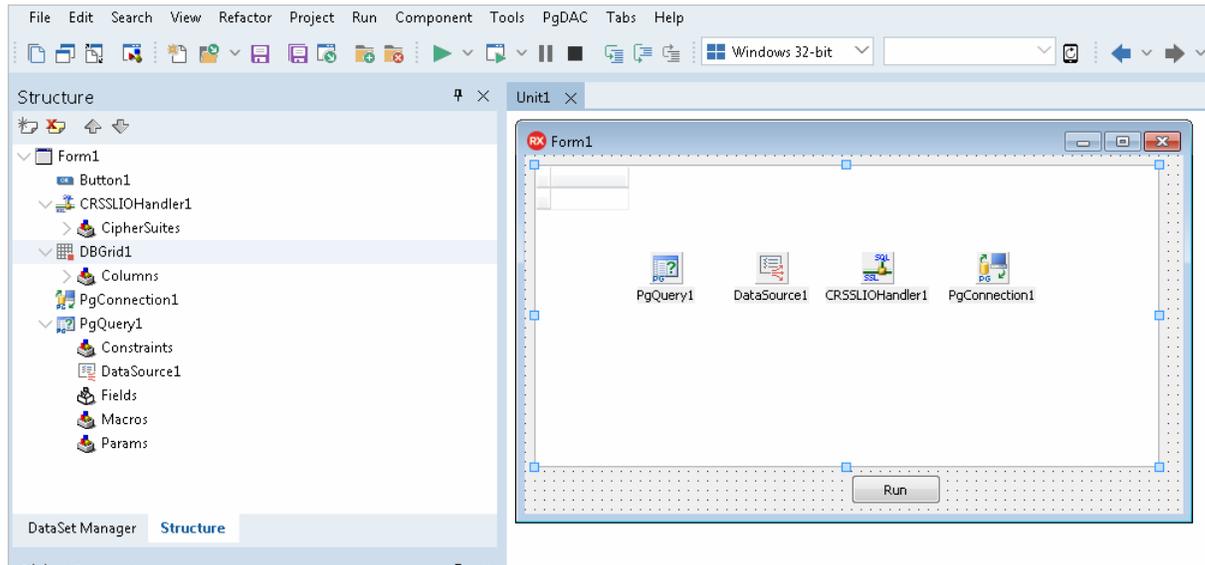
To create an SSL connection to PostgreSQL, PgDAC provides several [values](#) for the SSLOptions property. For this tutorial, the Mode property is set to smRequire , since it forces the application to only connect via SSL connection - if a connection attempt fails, an exception is raised.

Example of SSLOptions property set to smRequire:

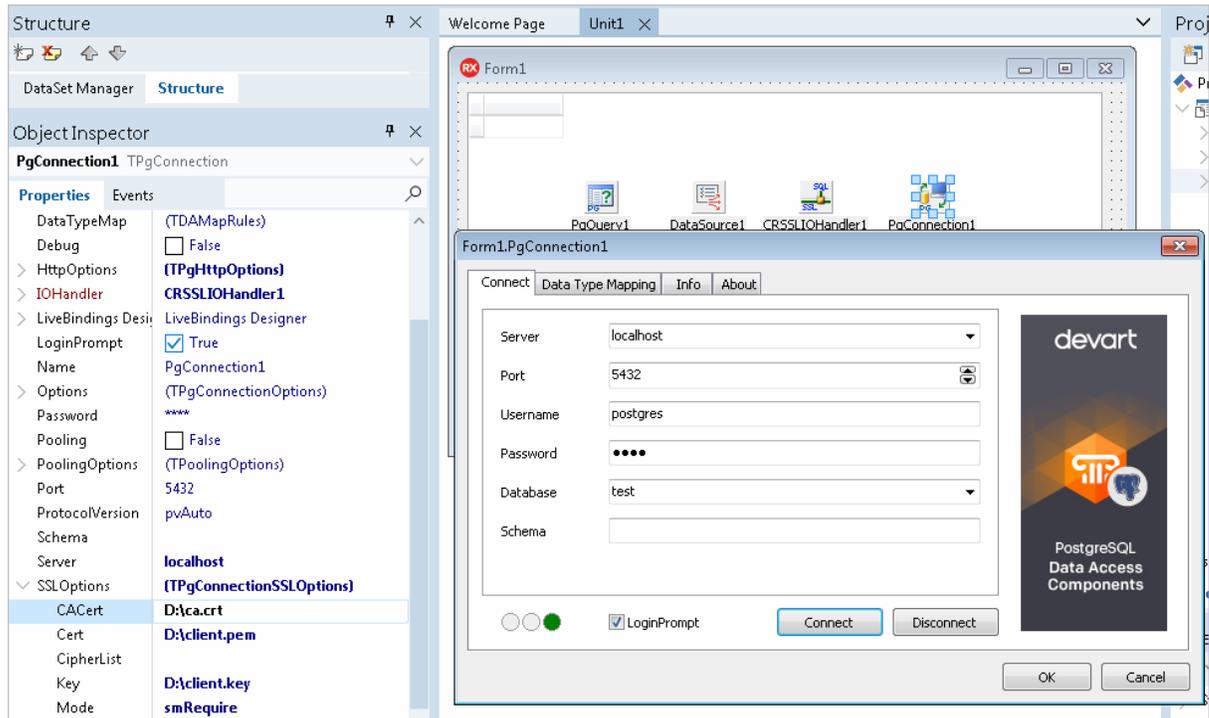
```
PgConnection.SSLOptions.Mode := smRequire;
```

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

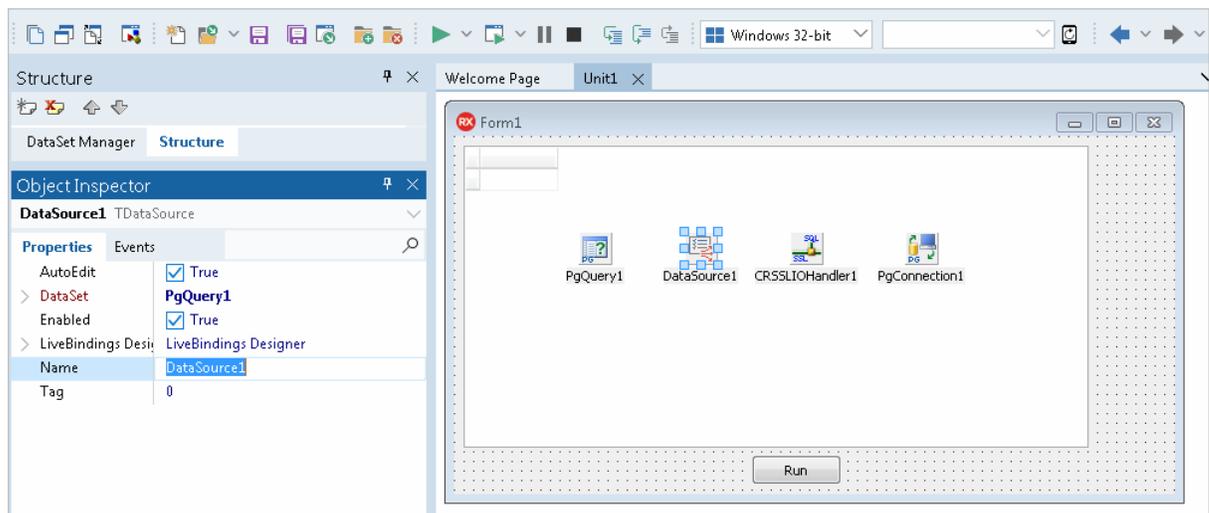
1. Run RAD Studio and select 'File -> New -> VCL Forms Application – Delphi'.
2. Place the **TCRSSLIOHandler** component, which allows PgDAC to connect to PostgreSQL server through SSL, onto the form. Also add the **TPgConnection**, **TPgQuery**, **TDataSource**, **TDBGrid**, and **TButton** components to the form - they are required to create a sample application that connects to the PostgreSQL server via SSL, runs a selection operation against the database, and displays the obtained rows in a data grid.



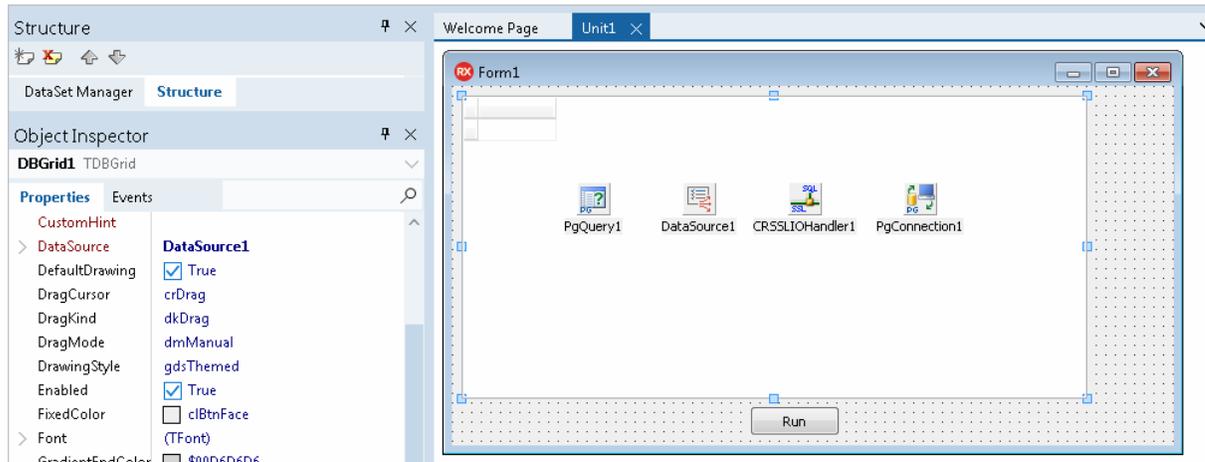
3. Select the **TPgConnection** component and assign the **TCRSSLIHandler** object to the **IOHandler** property in the Object Inspector.
4. Expand **SSLOptions** in the Object Inspector and specify the server certificate in the **CACert** property, the client certificate in the **Cert** property, and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



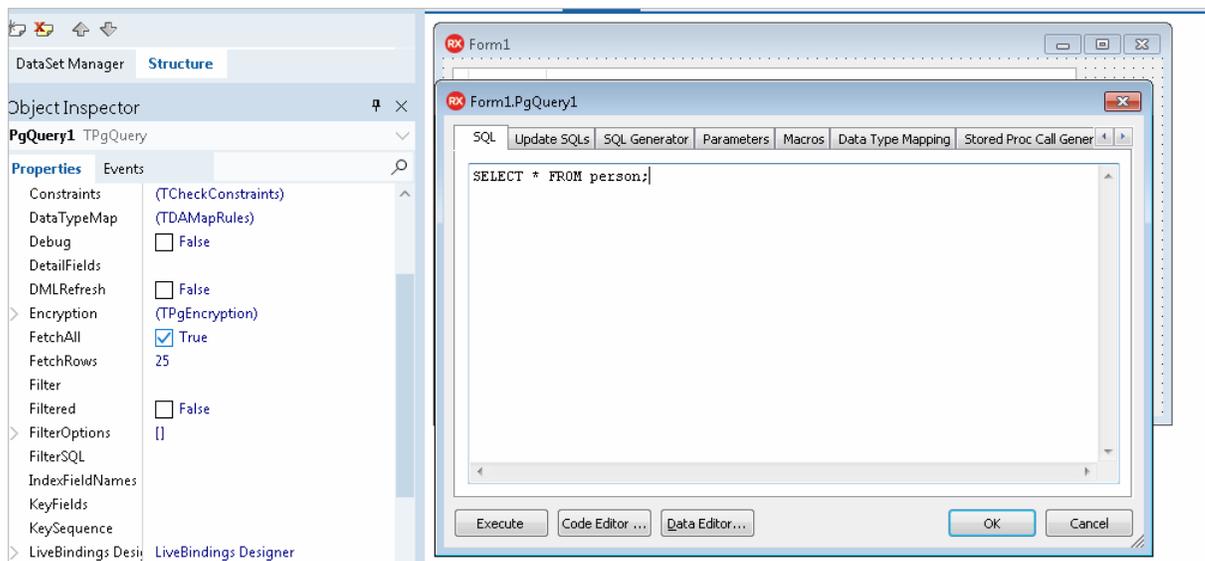
6. Select the **TDataSource** component and assign the **PgQuery1** object to the **DataSet** property.



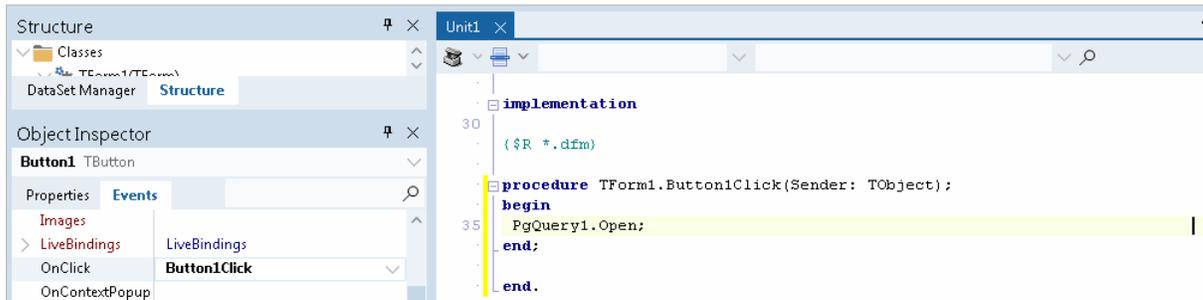
7. Assign the **DataSource1** object to the **DataSource** property in the **TDBGrid** component.



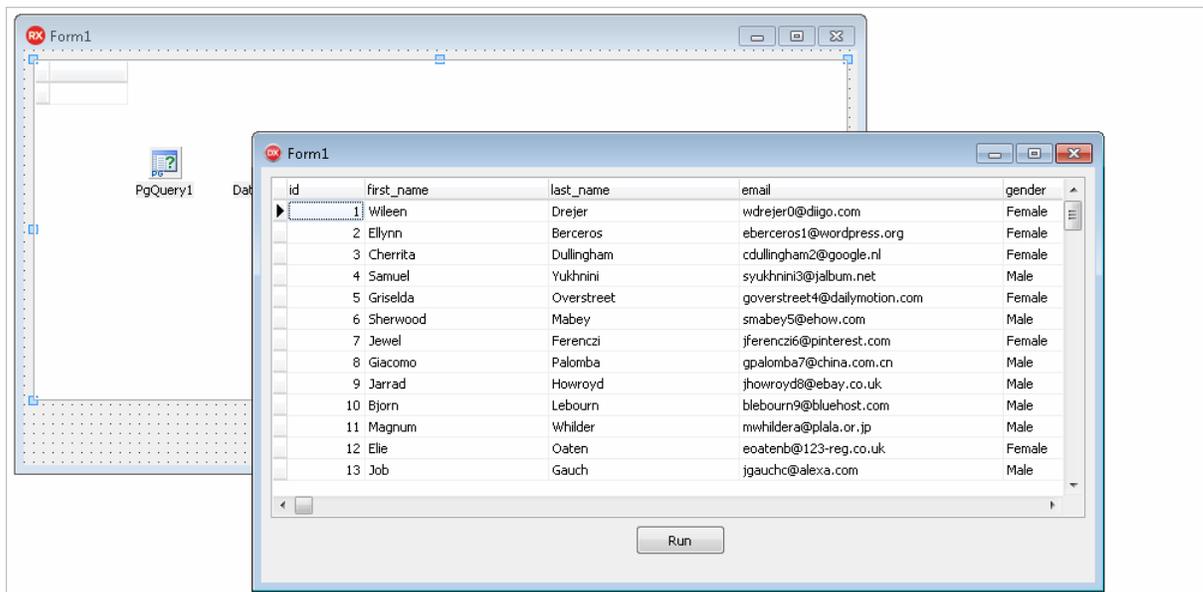
8. Double-click the **TPgQuery** component and add a SQL query that will be run against the PostgreSQL database.



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



10. Press F9 to compile and run the application. Click the button on the form to execute the query and display data in the grid.



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

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

The following options must be set for SSL connection:

- `SSLCACert` - the pathname to the certificate authority file.

- SSLCert - the pathname to the certificate file.
- SSLKey - the pathname to the key file.
- SSLCipherList - a list of allowable ciphers to use for SSL encryption.

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.8.2 Connecting via SSH

Connecting to PostgreSQL via SSH in Delphi

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

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

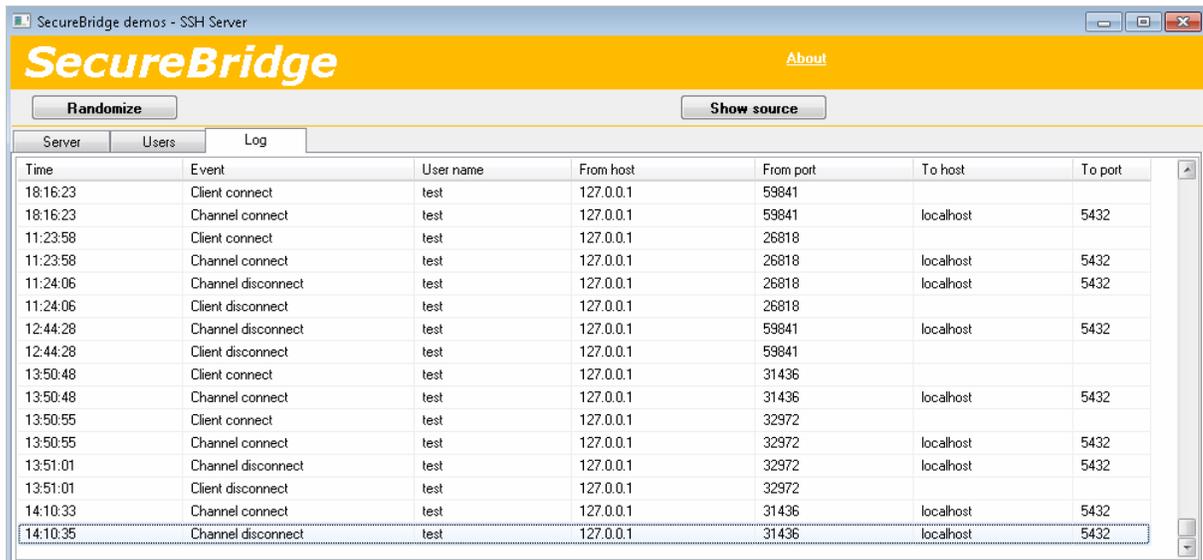
Devart offers a solution called [SecureBridge](#) that allows you to create a Delphi SSH client and a server. You can embed the SSH client into your application and install the SSH server on a remote machine where your PostgreSQL server resides. The SSH client connects to the SSH server, which sends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

SSH key-based authentication is done by public and private keys that a client uses to authenticate itself when logging into an SSH server. The server key is used by the client to authenticate the SSH server and is specified in the `TScSSHClient.HostKeyName`

property. The client key is used by the SSH server to authenticate the client and is specified in the TScSSHClient.PrivateKeyName property. Note that the private key contains the public key. See SecureBridge [tutorial](#) on configuring the SSH server.

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

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



The screenshot shows the 'SecureBridge demos - SSH Server' application window. The title bar includes 'SecureBridge' and 'About'. Below the title bar are buttons for 'Randomize' and 'Show source'. The main area contains a log table with columns: Time, Event, User name, From host, From port, To host, and To port. The log shows a series of connection and disconnection events for a user named 'test' on host '127.0.0.1'.

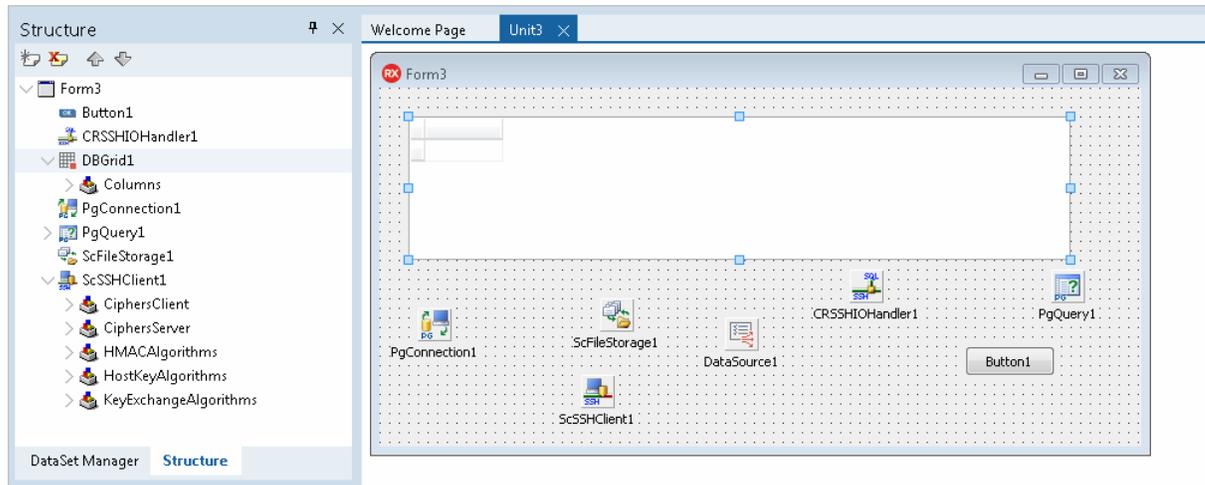
Time	Event	User name	From host	From port	To host	To port
18:16:23	Client connect	test	127.0.0.1	59841		
18:16:23	Channel connect	test	127.0.0.1	59841	localhost	5432
11:23:58	Client connect	test	127.0.0.1	26818		
11:23:58	Channel connect	test	127.0.0.1	26818	localhost	5432
11:24:06	Channel disconnect	test	127.0.0.1	26818	localhost	5432
11:24:06	Client disconnect	test	127.0.0.1	26818		
12:44:28	Channel disconnect	test	127.0.0.1	59841	localhost	5432
12:44:28	Client disconnect	test	127.0.0.1	59841		
13:50:48	Client connect	test	127.0.0.1	31436		
13:50:48	Channel connect	test	127.0.0.1	31436	localhost	5432
13:50:55	Client connect	test	127.0.0.1	32972		
13:50:55	Channel connect	test	127.0.0.1	32972	localhost	5432
13:51:01	Channel disconnect	test	127.0.0.1	32972	localhost	5432
13:51:01	Client disconnect	test	127.0.0.1	32972		
14:10:33	Channel connect	test	127.0.0.1	31436	localhost	5432
14:10:35	Channel disconnect	test	127.0.0.1	31436	localhost	5432

Sample Delphi app that connects to PostgreSQL using SSH

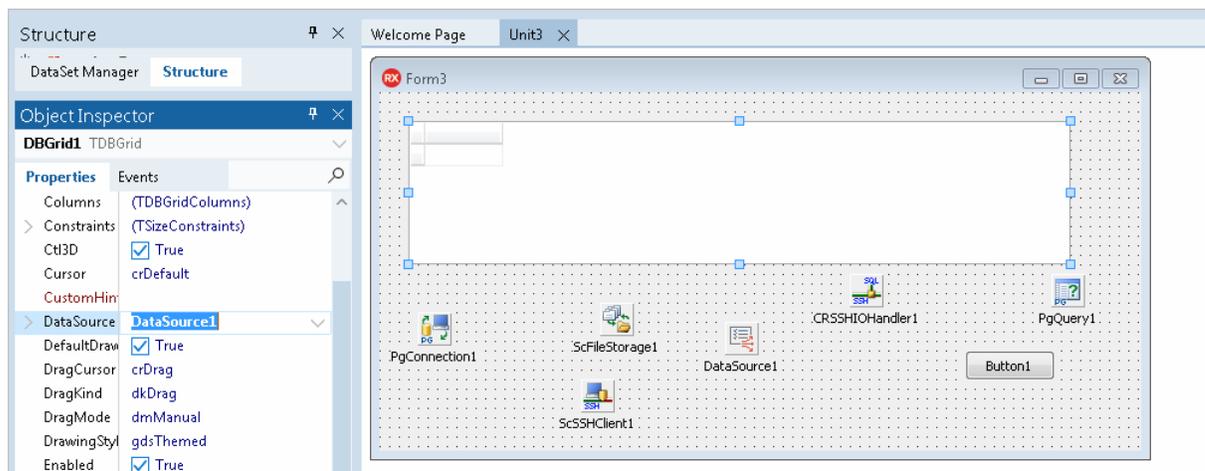
1. Run RAD Studio and select 'File -> New -> VCL Forms Application - Delphi'.
2. Place the following components on the form: **TCRSSHIOHandler**, **TPgConnection**, **TPgQuery**, **TScFileStorage**, **TScSSHClient**, **TDataSource**, **TDBGrid**, and **TButton**.

The sample application will connect to the PostgreSQL server via SSH, run a selection

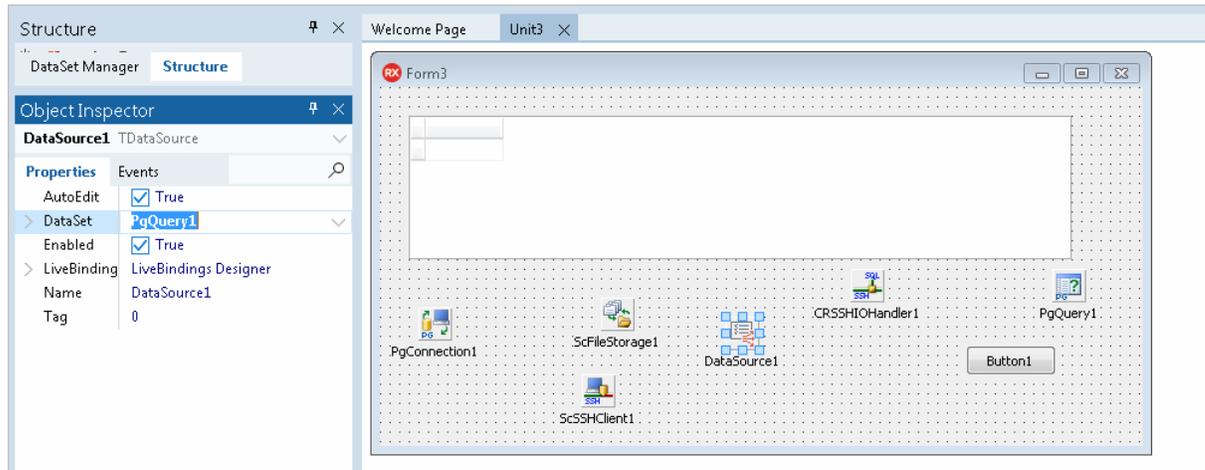
operation against the database, and display the obtained rows in the grid.



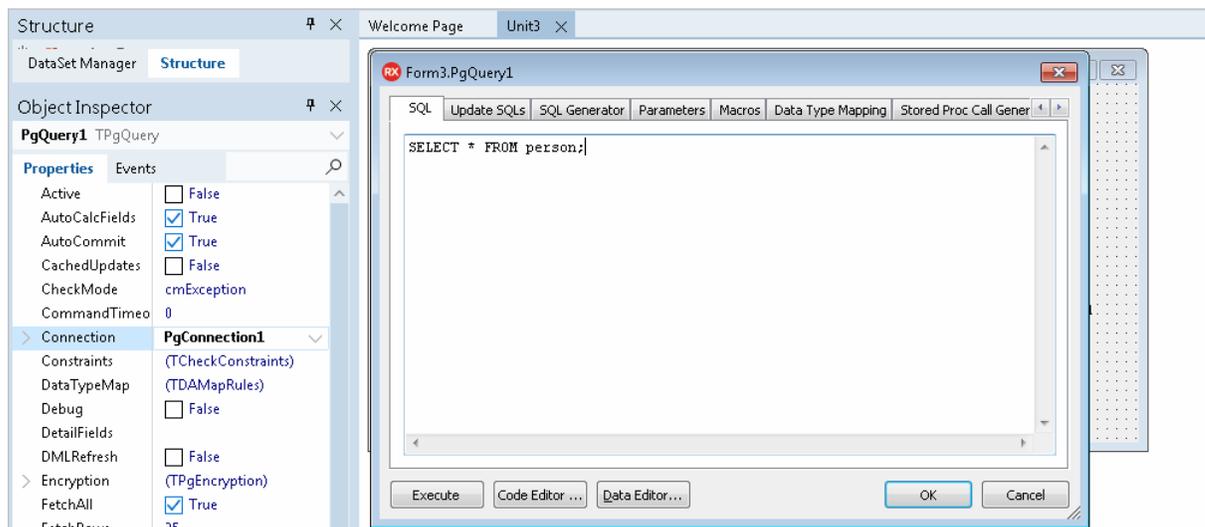
3. Select the **TDBGrid** and set the **DataSource** property to **DataSource1**.



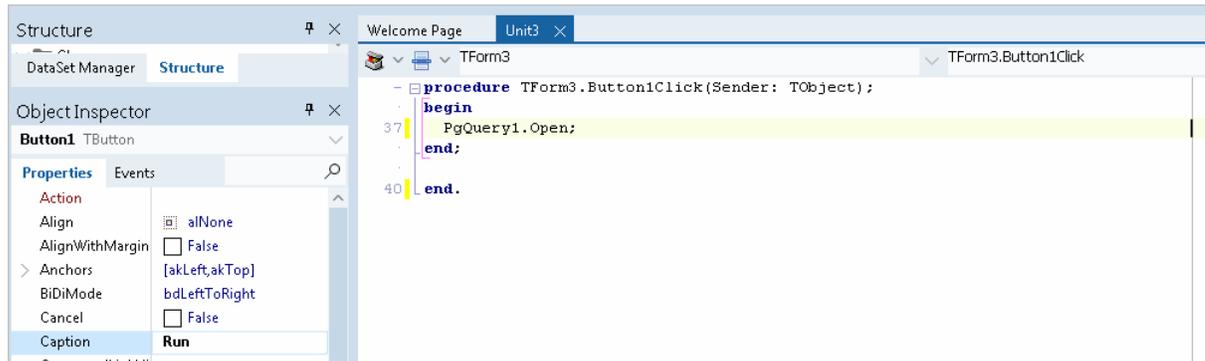
4. In the **TDataSource** component, assign **PgQuery1** to the **DataSet** property.



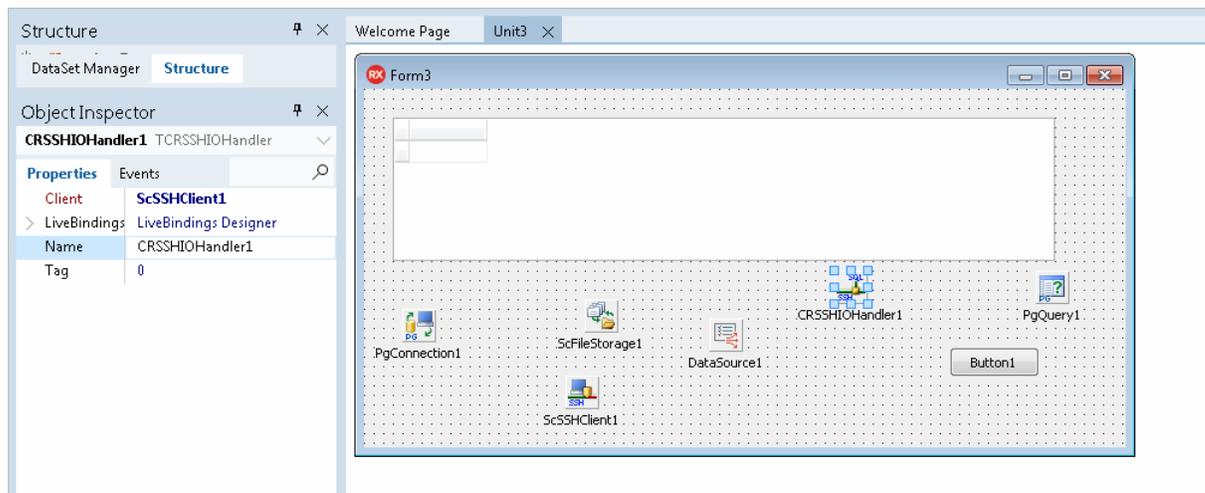
5. Select the **TPgQuery** and set the **Connection** property to **PgConnection1**. Double-click the component and enter a SQL statement to be executed against the PostgreSQL database.



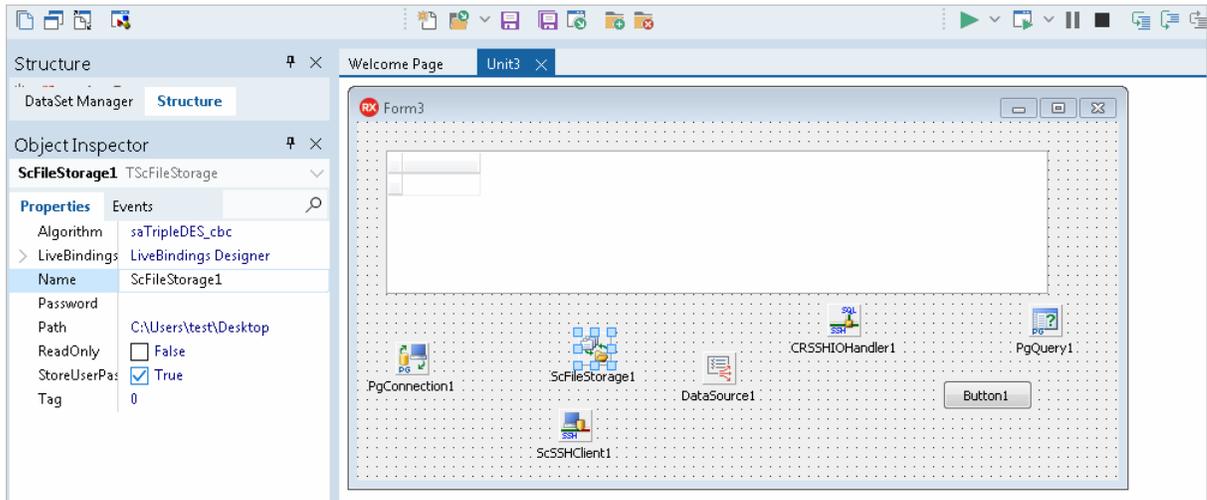
6. Double-click the **TButton** to switch to the unit view. Add the code to call the **Open** method on the **PgQuery1** object to activate the dataset when the button is clicked.



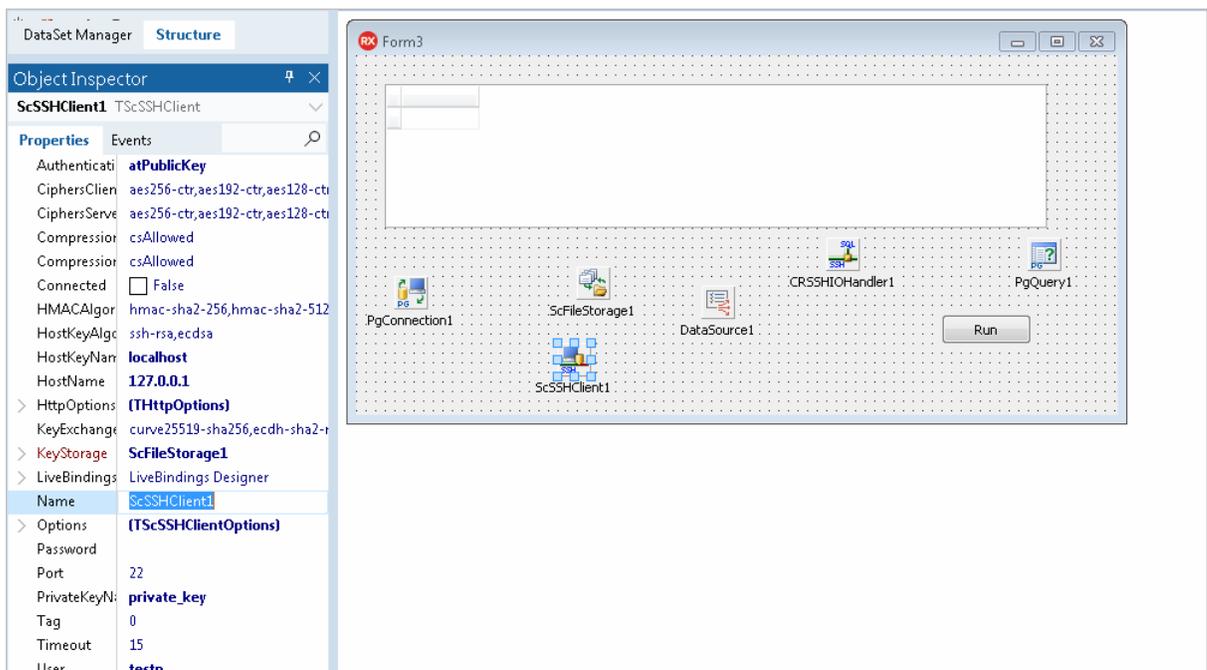
7. In the **TCRSSHIOHandler** component, assign **ScSSHClient1** to the **Client** property.



8. Select the **TScFileStorage** component and specify in the **Path** property the directory where keys are stored on your system. Double-click the component and [generate](#) a pair of keys for authenticating the server by the client.

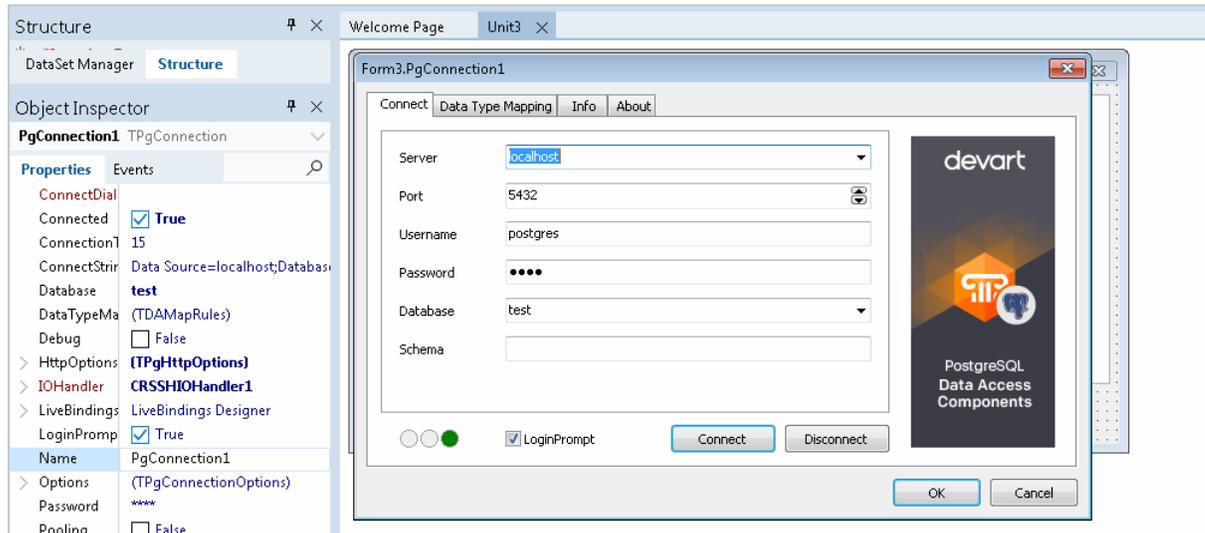


9. Set the **Authentication** property to **atPublicKey** in the **TScSSHClient** component. In **HostKeyName**, specify the server public key. In **PrivateKeyName**, specify the client private key. The **Hostname** property holds the address of your server. Assign **ScFileStorage1** to the **KeyStorage** property. Enter your username on the server in the **User** property. Specify the SSH port in the **Port** property.

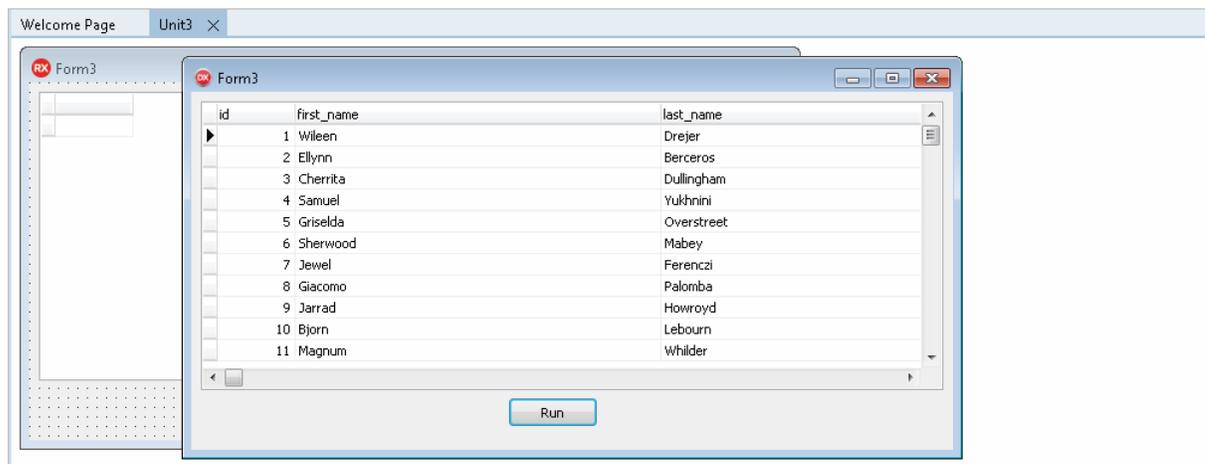


10. Double-click the **TPgConnection** component. Specify your server address, port,

database name (optionally), and username and password for the PostgreSQL user. Set the **IOHandler** property to **CRSSHIOHandler1**. Click Connect to check connection to the PostgreSQL server.



11. Press **F9** to compile and run the project. Click the button to run the query against the database and display the data in the form.



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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.8.3 Network Tunneling

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

- [Direct Connection](#)
- [Connection Through HTTP Tunnel](#)
 - [Connection Through Proxy and HTTP Tunnel](#)
- [Sample Delphi App That Connects to PostgreSQL Using HTTP/HTTPS Tunneling](#)
- [Additional Information](#)

Direct Connection

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

The following code illustrates the simplicity:

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

Connection Through HTTP Tunnel

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

An example of the web script tunneling usage can be the following: you have a remote website, and access to its database through the port of the database server is forbidden. Only

access through HTTP port 80 is allowed, and you need to access the database from a remote computer, like when using usual direct connection.

You need to deploy the tunnel.php script, which is included into the provider package, on the web server. It allows access to the database server via HTTP tunneling. The script must be accessible through the HTTP protocol. You can verify if it is accessible with a web browser. The script can be found in the HTTP subfolder of the installed provider folder, e. g. %Program Files%\Devart\PgDac for Delphi XX\HTTP\tunnel.php. The only requirement to the server is PHP 5 support.

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

Property	Mandatory	Meaning
HttpOptions.Url	Yes	URL of the tunneling PHP script. For example, if the script is in the server root, the URL can be the following: http://localhost/tunnel.php.
HttpOptions.UserName, HttpOptions.Password	No	Set these properties if the access to the website folder with the script is available only for registered users authenticated with user name and password.

Connecting Through Proxy and HTTP Tunnel

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

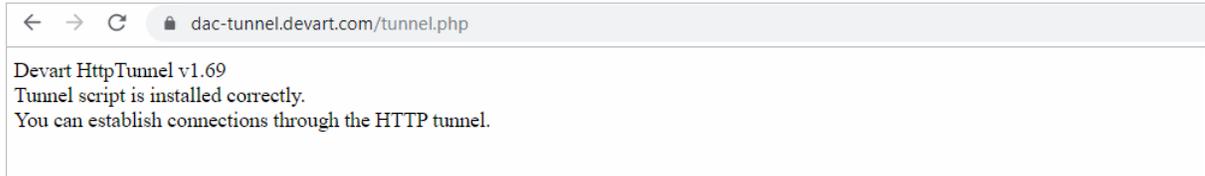
```
PgConnection := TPgConnection.Create(self);  
PgConnection.Server := '192.168.0.10';  
PgConnection.Port := 5433;  
PgConnection.Username := 'postgres';  
PgConnection.Password := 'postgres';  
PgConnection.HttpOptions.Enabled := True;  
PgConnection.HttpOptions.Url := 'http://server/tunnel.php';
```

```
PgConnection.HttpOptions.ProxyOptions.Hostname := '10.0.0.1';  
PgConnection.HttpOptions.ProxyOptions.Port := 808;  
PgConnection.HttpOptions.ProxyOptions.Username := 'ProxyUser';  
PgConnection.HttpOptions.ProxyOptions.Password := 'ProxyPassword';  
PgConnection.Connect;
```

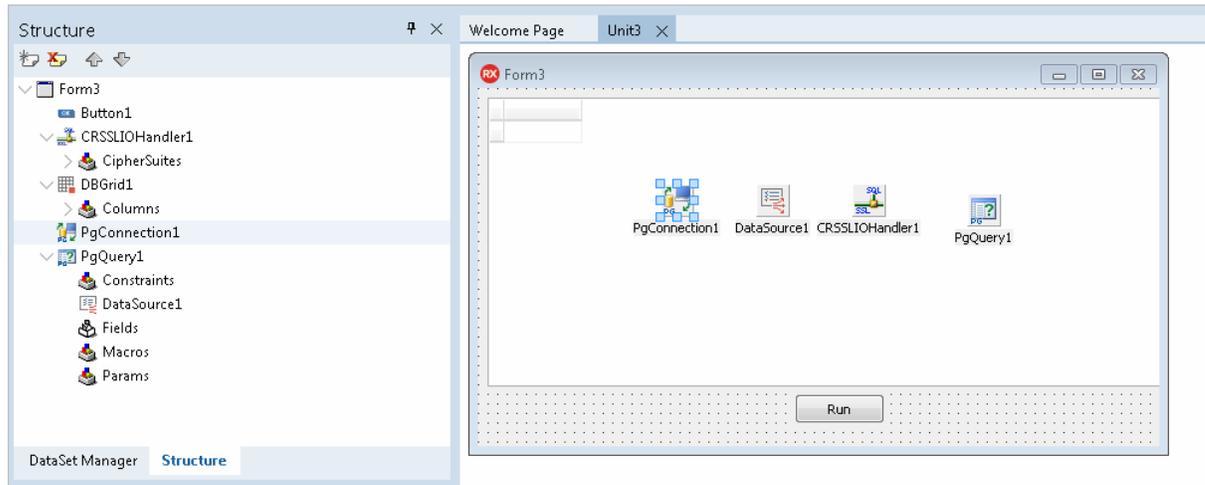
Note that setting parameters of `PgConnection.HttpOptions.ProxyOptions` automatically enables the use of the proxy server.

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

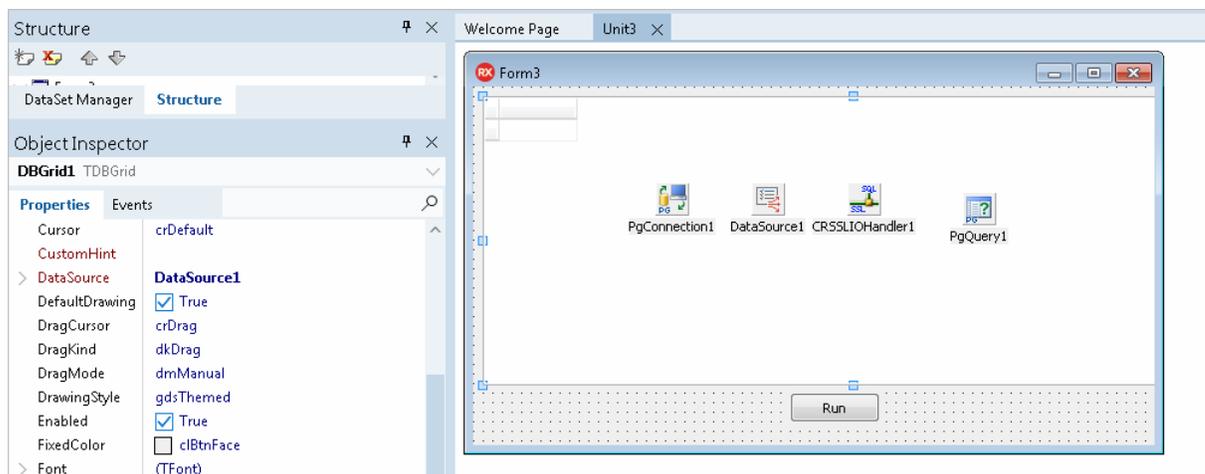
- Open your browser and visit the URL of the `tunnel.php` script on your server to verify that the script has been properly installed.



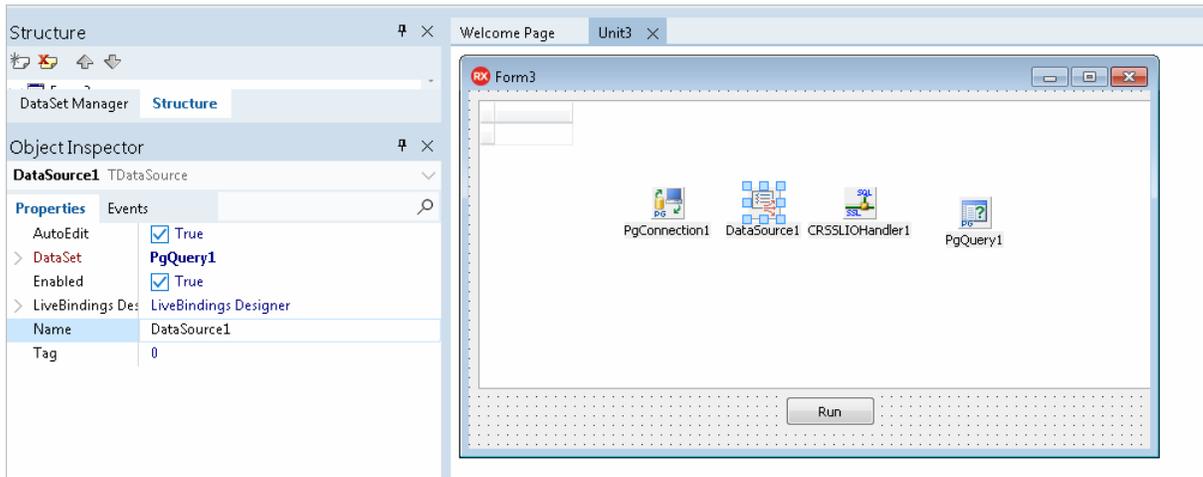
- Run RAD Studio and select 'File -> New -> VCL Forms Application - Delphi'.
- Place the following components on the form: **TPgConnection**, **TPgQuery**, **TDataSource**, **TDBGrid**, **TButton**, and **TCRSSLIOHandler**. The last component is required when you connect via HTTPS. Note that the **TCRSSLIOHandler**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



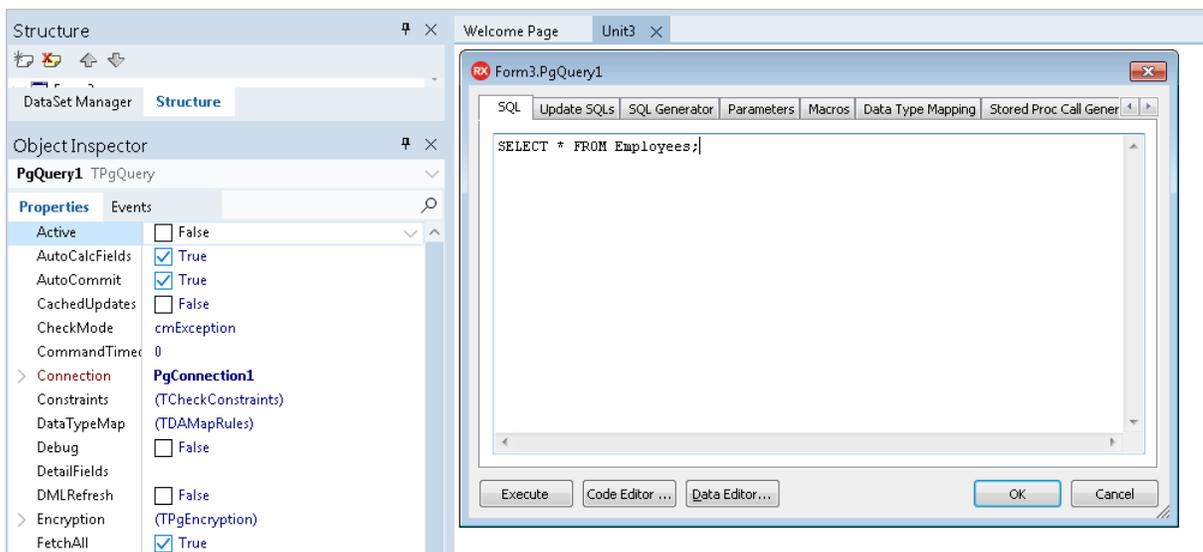
- Select **TDBGrid** and set the **DataSource** property to **DataSource1**.



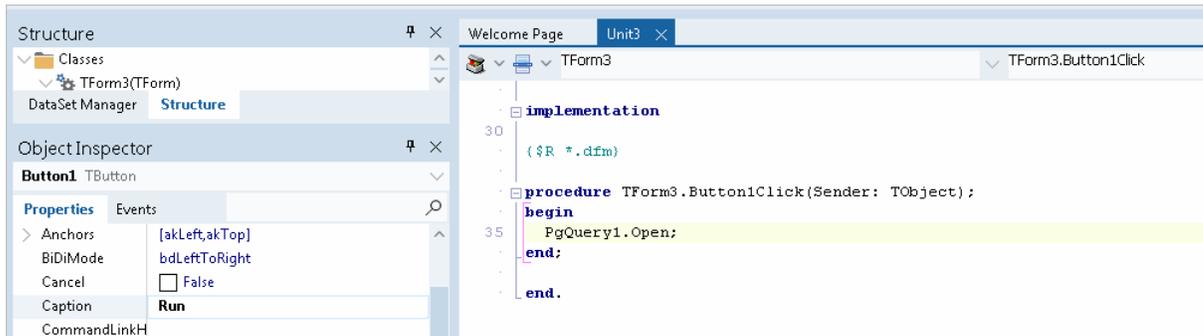
- In the **TDataSource** component, assign **PgQuery1** to the **DataSet** property.



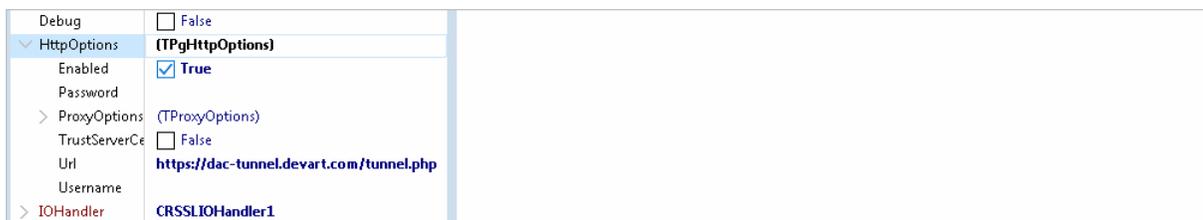
- Select **TPgQuery** and set the **Connection** property to **PgConnection1**. Double-click the component and enter an SQL statement to be executed against the PostgreSQL database.



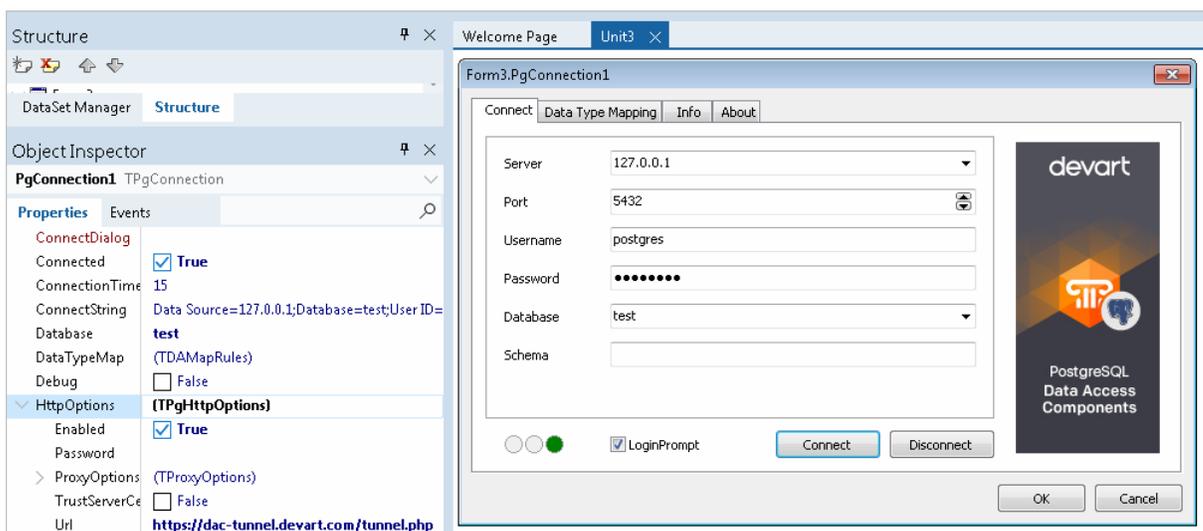
- Double-click **TButton** to switch to the unit view. Add code that invokes the **Open** method on the **PgQuery1** object to activate the dataset when the button is clicked.



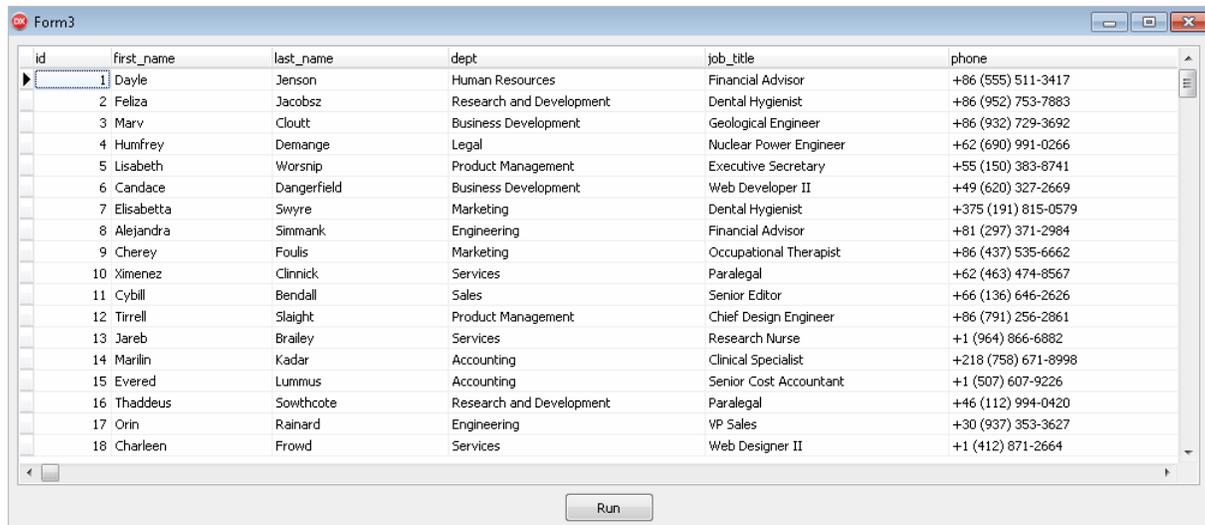
- Select the **TPgConnection** component. If you use the secure tunnel, set the **IOHandler** property to **CRSSLIHandler1**. Expand the **HttpOptions** and enter the URL of the tunnel.php script on your server. Set the **Enabled** property in **HttpOptions** to **True**.



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



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



id	first_name	last_name	dept	job_title	phone
1	Dayle	Jenson	Human Resources	Financial Advisor	+86 (555) 511-3417
2	Feliza	Jacobsz	Research and Development	Dental Hygienist	+86 (952) 753-7883
3	Marv	Cloutt	Business Development	Geological Engineer	+86 (932) 729-3692
4	Humfrey	Demange	Legal	Nuclear Power Engineer	+62 (690) 991-0266
5	Lisabeth	Worsnip	Product Management	Executive Secretary	+55 (150) 383-8741
6	Candace	Dangerfield	Business Development	Web Developer II	+49 (620) 327-2669
7	Elisabetta	Swyre	Marketing	Dental Hygienist	+375 (191) 815-0579
8	Alejandra	Simbank	Engineering	Financial Advisor	+81 (297) 371-2984
9	Chery	Foulis	Marketing	Occupational Therapist	+86 (437) 535-6662
10	Ximenez	Clinnick	Services	Paralegal	+62 (463) 474-8567
11	Cybill	Bendall	Sales	Senior Editor	+66 (136) 646-2626
12	Tirrell	Slaight	Product Management	Chief Design Engineer	+86 (791) 256-2861
13	Jareb	Brailey	Services	Research Nurse	+1 (964) 866-6882
14	Marlin	Kadar	Accounting	Clinical Specialist	+218 (758) 671-8998
15	Evered	Lummus	Accounting	Senior Cost Accountant	+1 (507) 607-9226
16	Thaddeus	Southcote	Research and Development	Paralegal	+46 (112) 994-0420
17	Orin	Rainard	Engineering	VP Sales	+30 (937) 353-3627
18	Charleen	Frowd	Services	Web Designer II	+1 (412) 871-2664

Additional Information

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.9 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 [TCustomDACConnection.Pooling](#).

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

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

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

```
TDataSet.CachedUpdates = True
TCustomDADataset.FetchAll = True
TCustomDADataset.Options.LocalMasterDetail = True
```

These settings minimize the number of requests to the server.

Disconnected mode features

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

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

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

See Also

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

- [Working in unstable network](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.10 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.**

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

4.11 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 [TCustomDAUpdateSQL](#). The performance gain achieved this way can be anywhere from several percent to several times, depending on the situation.

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

If the [TCustomDADataset.Options.StrictUpdate](#) option is set to False, the [RowsAffected](#)

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

Fetch

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

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

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

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

Navigate

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

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

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

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

Update

If your application updates datasets in the CachedUpdates mode, then setting the

[TCustomDADataset.Options.UpdateBatchSize](#) option to more than 1 can improve performance several hundred times more by reducing the number of requests to the server.

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

Insert

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.12 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

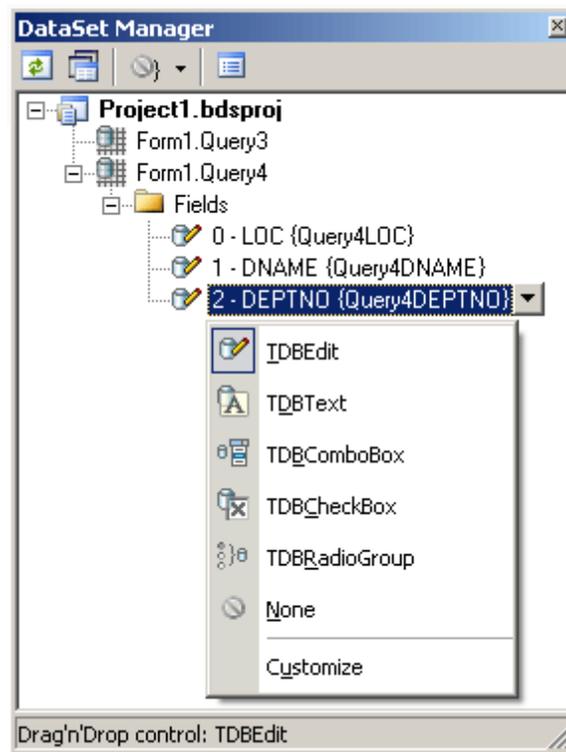
[DAC Forum](#)

[Provide Feedback](#)

4.13 DataSet Manager

DataSet Manager window

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



Using the DataSet Manager window, you can:

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

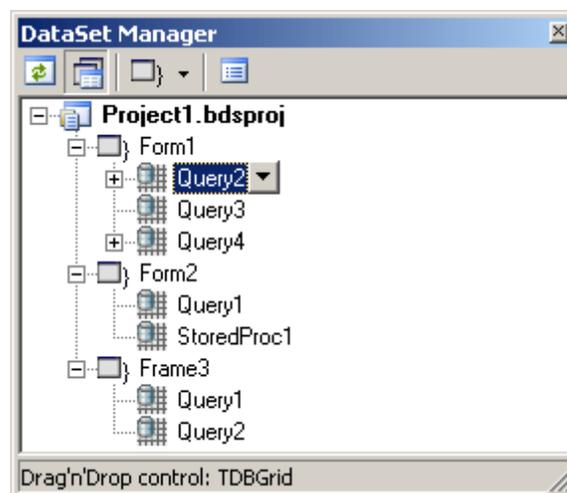
Opening the DataSet Manager window

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

Observing project DataSets in the DataSet Manager Window

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

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

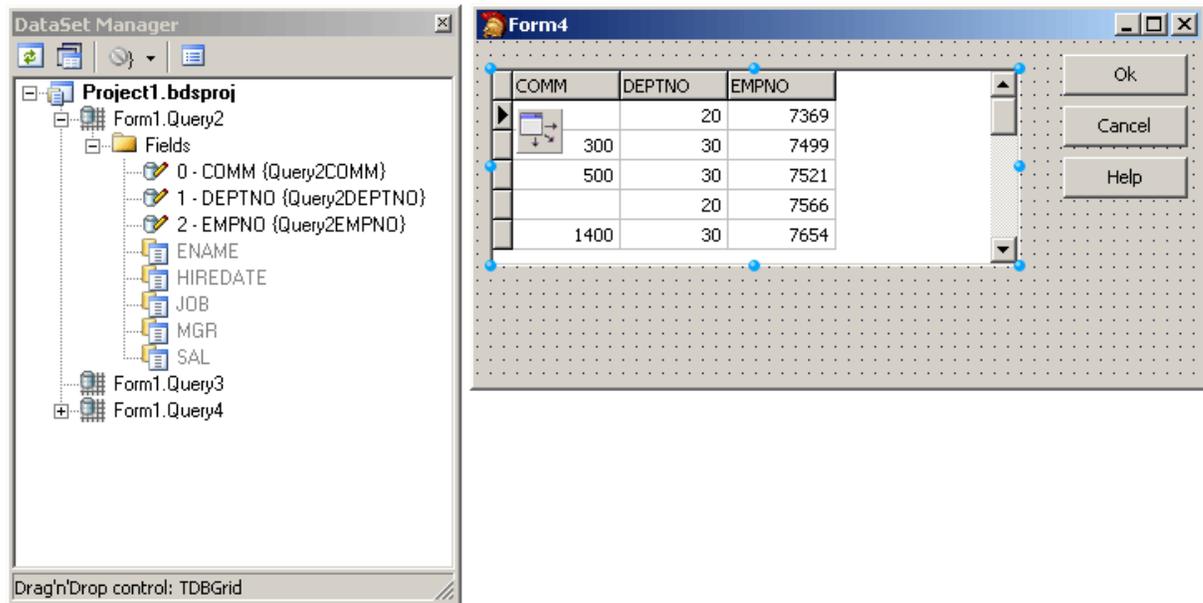


Creating Data-bound Controls

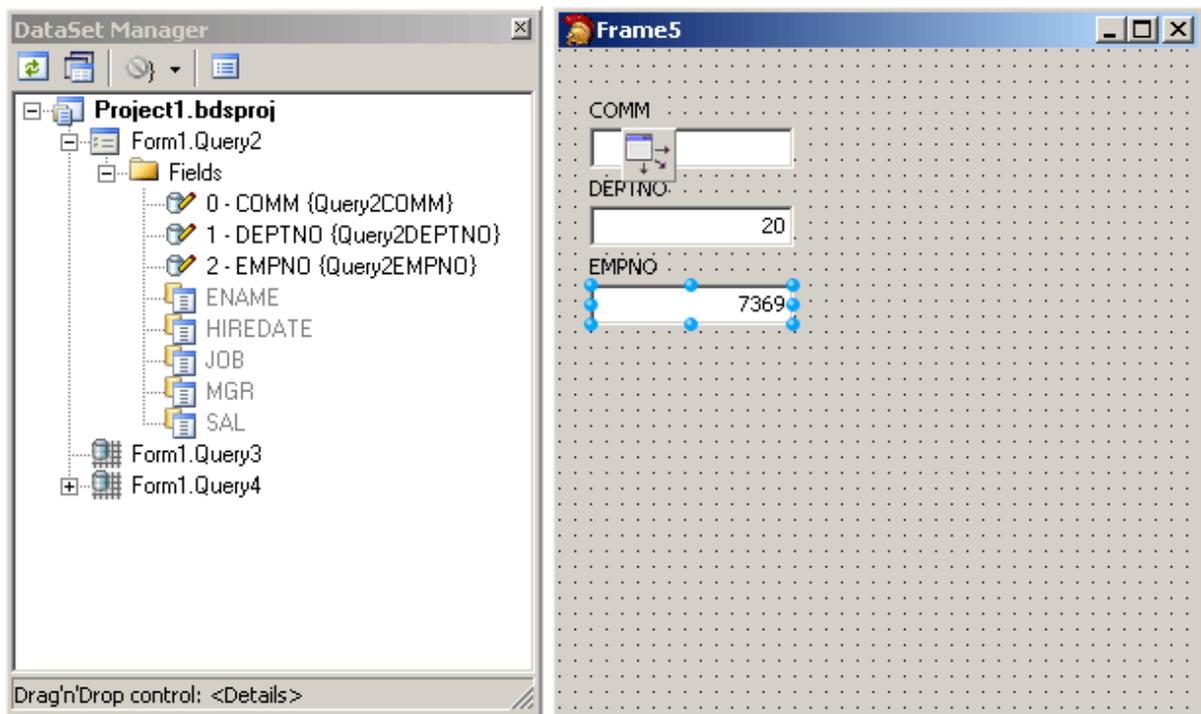
You can drag an item from the DataSet Manager window onto a form to create a new data-bound control. Each node in the DataSet Manager window allows you to choose the type of control that will be created when you drag it onto a form. You must choose between a Grid layout, where all columns or properties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

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

drop-down control list.

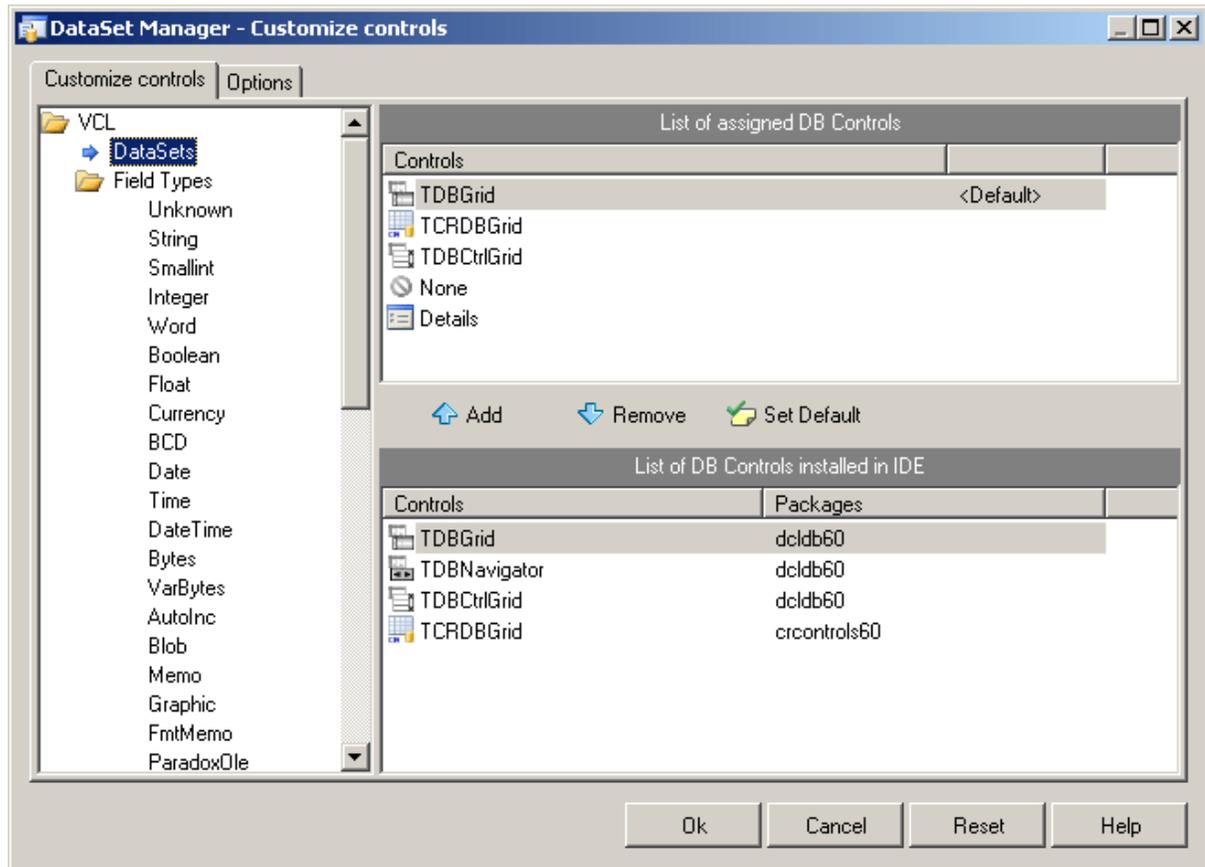


To use Details layout choose Details from the DataSet node drop-down control list in the DataSet Manager window. Then select required controls in the drop-down control list for each DataSet field. DataSet fields must be created. After setting required options you can drag the DataSet to the form from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



Adding custom controls to the DataSet Manager window

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



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

Working with TField objects

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

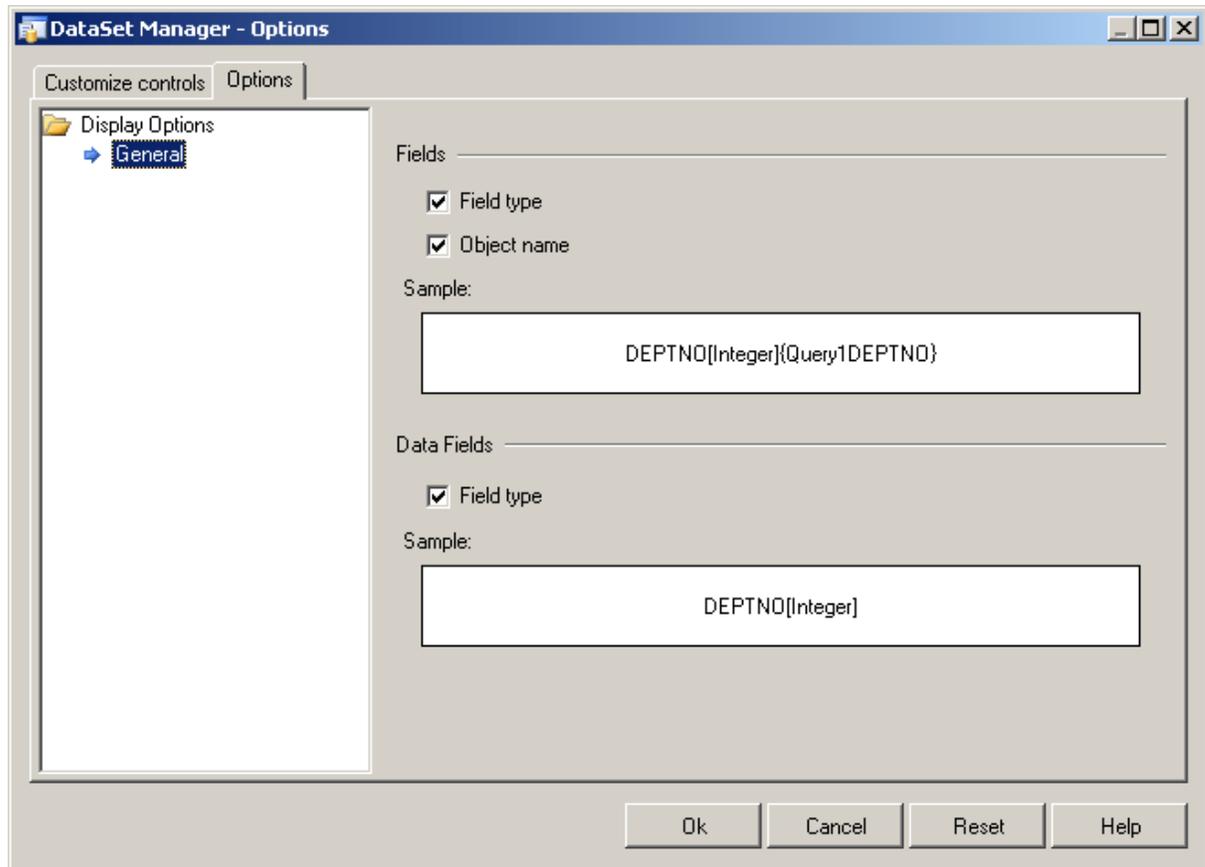
To create a field based on the database table column right-click the Fields node and select *Create Field* from the popup menu or press <Insert>. Note that after you add at least one field manually, DataSet fields corresponding to data fields will not be generated automatically when you drag the DataSet on the form, and you can not drag such fields on the form. To add all available fields right-click the Fields node and select *Add all fields* from the popup menu.

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

OK button.

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

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



You can 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.14 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.15 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.ReadBlob;
  LargeObject.Read(0, 10, &AData);
  LargeObject.CloseObject;
finally
  LargeObject.Free;
end;
...
```

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

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

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

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

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

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

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.16 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;
```



© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.17 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.18 Connection Pooling

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

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

To use connection pooling set the Pooling property of the [TCustomDACConnection](#) component to True. Also you should set the [PoolingOptions](#) of the [TCustomDACConnection](#). These options include [MinPoolSize](#), [MaxPoolSize](#), [Validate](#), [ConnectionLifeTime](#). Connections belong to the same pool if they have identical values for the following parameters: [MinPoolSize](#), [MaxPoolSize](#), [Validate](#), [ConnectionLifeTime](#), [Server](#), [Username](#), [Password](#), [Database](#), [Port](#), [ProtocolVersion](#), [Charset](#), [UseUnicode](#), [Schema](#), [ConnectionTimeout](#), [SSLOptions](#). When a connection component disconnects from the database the connection actually remains active and is placed into the pool. When this or another connection component connects to the database it takes a connection from the pool. Only when there are no connections in the pool, new connection is established.

Connections in the pool are validated to make sure that a broken connection will not be returned for the [TCustomDACConnection](#) component when it connects to the database. The pool validates connection when it is placed to the pool (e. g. when the [TCustomDACConnection](#)

component disconnects). If connection is broken it is not placed to the pool. Instead the pool frees this connection. Connections that are held in the pool are validated every 30 seconds. All broken connections are freed. If you set the [PoolingOptions.Validate](#) to True, a connection also will be validated when the [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

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

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.19 DBMonitor

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

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

DBMonitor has the following features:

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

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

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.20 Writing GUI Applications with PgDAC

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

Delphi and C++Builder

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.21 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 DataSetChange event 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.DoNotRaiseExcecionOnUaFail:

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 DoNotRaiseExcecionOnUaFail to True.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.22 64-bit Development with Embarcadero RAD Studio XE2

RAD Studio XE2 Overview

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

Its main features are the following:

- Windows 64-bit platform support;
- Mac OS support;
- FireMonkey application development platform;
- Live data bindings with visual components;

- VCL styles for Windows applications.

Changes in 64-bit Application Development

64-bit platform support implies several important changes that each developer must keep in mind prior to the development of a new application or the modernization of an old one.

General

RAD Studio XE2 IDE is a 32-bit application. It means that it cannot load 64-bit packages at design-time. So, all design-time packages in RAD Studio XE2 IDE are 32-bit.

Therefore, if you develop your own components, you should remember that for the purpose of developing components with the 64-bit platform support, you have to compile run-time packages both for the 32- and 64-bit platforms, while design-time packages need to be compiled only for the 32-bit platform. This might be a source of difficulties if your package is simultaneously both a run-time and a design-time package, as it is more than likely that this package won't be compiled for the 64-bit platform. In this case, you will have to separate your package into two packages, one of which will be used as run-time only, and the other as design-time only.

For the same reason, if your design-time packages require that certain DLLs be loaded, you should remember that design-time packages can be only 32-bit and that is why they can load only 32-bit versions of these DLLs, while at run-time 64-bit versions of the DLLs will be loaded. Correspondingly, if there are only 64-bit versions of the DLL on your computer, you won't be able to use all functions at design-time and, vice versa, if you have only 32-bit versions of the DLLs, your application won't be able to work at run-time.

Extended type

For this type in a 64-bit applications compiler generates SSE2 instructions instead of FPU, and that greatly improves performance in applications that use this type a lot (where data accuracy is needed). For this purpose, the size and precision of Extended type is reduced:

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/Tokyo/en/Running_the_Platform_Assistant_on_Windows

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.23 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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	This unit contains the TCRDBGrid component.
CRVio	This unit contains classes for HTTP connections.
CRXml	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.
PgDataTypeMap	This unit contains the implementation of mapping between PostgreSQL and Delphi data types.
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.
PgTransaction	Description is not available at the moment.

VirtualDataSet	This unit contains implementation of the TVirtualDataSet component.
VirtualTable	This unit contains implementation of the TVirtualTable component.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

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 TCustomDADataset.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

© 1997-2024

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

Devart. All Rights Reserved.

5.1.1 Classes

Classes in the **CRAccess** unit.

Classes

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.1.1 TCRCursor Class

A base class for classes that work with database cursors.

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

Unit

[CRAccess](#)

Syntax

```
TCRCursor = class(TSharedObject);
```

Remarks

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

Inheritance Hierarchy

[TSharedObject](#)

TCRCursor

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.2 Types

Types in the **CRAccess** unit.

Types

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.2.1 TBeforeFetchProc Procedure Reference

This type is used for the [TCustomDADDataSet.BeforeFetch](#) event.

Unit

[CRAccess](#)

Syntax

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

Parameters

Cancel

True, if the current fetch operation should be aborted.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.3.1 TCRIsolationLevel Enumeration

Specifies how to handle transactions containing database modifications.

Unit

[CRAccess](#)

Syntax

```
TCRIsolationLevel = (ilReadCommitted, ilReadUnCommitted,
```

```
ilRepeatableRead, ilIsolated, ilSnapshot, ilCustom);
```

Values

Value	Meaning
ilCustom	The parameters of the transaction are set manually in the Params property.
ilIsolated	The most restricted level of transaction isolation. Database server isolates data involved in current transaction by putting additional processing on range locks. Used to put aside all undesired effects observed in the concurrent accesses to the same set of data, but may lead to a greater latency at times of a congested database environment.
ilReadCommitted	Sets isolation level at which transaction cannot see changes made by outside transactions until they are committed. Only dirty reads (changes made by uncommitted transactions) are eliminated by this state of the isolation level. The default value.
ilReadUnCommitted	The most unrestricted level of the transaction isolation. All types of data access interferences are possible. Mainly used for browsing database and to receive instant data with prospective changes.
ilRepeatableRead	Prevents concurrent transactions from modifying data in the current uncommitted transaction. This level eliminates dirty reads as well as nonrepeatable reads (repeatable reads of the same data in one transaction before and after outside transactions may have started and committed).
ilSnapshot	Uses row versioning. Provides transaction-level read consistency. A data snapshot is taken when the snapshot transaction starts, and remains consistent for the duration of a transaction.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.3.2 TCRTransactionAction Enumeration

Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Unit

[CRAccess](#)

Syntax

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

Values

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.1 Classes

Classes in the **CRBatchMove** unit.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.1.1 TCRBatchMove Class

Transfers records between datasets.

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

Unit

[CRBatchMove](#)

Syntax

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

Remarks

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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

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.
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.

Published

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.
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.
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.1.1.2.1 AbortOnKeyViol Property

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

Class

[TCRBatchMove](#)

Syntax

```
property AbortOnKeyViol: boolean default True;
```

Remarks

Use the AbortOnKeyViol property to specify whether the batch operation is terminated immediately after key or integrity violation.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.2.1.1.2.8 Mappings Property

Used to set field matching between source and destination datasets for the batch operation.

Class

[TCRBatchMove](#)

Syntax

```
property Mappings: TStrings;
```

Remarks

Use the Mappings property to set field matching between the source and destination datasets for the batch operation. By default fields matching is based on their position in the datasets.

To map the column ColName in the source dataset to the column with the same name in the destination dataset, use:

ColName

Example

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

```
DestColName=SourceColName
```

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.2 Types

Types in the **CRBatchMove** unit.

Types

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

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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 .

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.3.1.1 TCEncryptor 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 [TCEncryptor](#) members.

Unit

[CEncryption](#)

Syntax

```
TCEncryptor = class(TComponent);
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.3.1.1.1 Members

[TCEncryptor](#) 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.3.1.1.2.1 DataHeader Property

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

Class

[TCREncryptor](#)

Syntax

```
property DataHeader: TCREncDataHeader default ehTagAndHash;
```

Remarks

Use DataHeader to specify whether the additional information is stored with the encrypted data. Default value is [ehTagAndHash](#).

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.3.1.1.2.2 EncryptionAlgorithm Property

Specifies the algorithm of data encryption.

Class

[TCREncryptor](#)

Syntax

```
property EncryptionAlgorithm: TCREncryptionAlgorithm default  
eaBlowfish;
```

Remarks

Use EncryptionAlgorithm to specify the algorithm of data encryption. Default value is [eaBlowfish](#).

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#).

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4 CRVio

This unit contains classes for HTTP connections.

Classes

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

THttpOptions	This class is used to establish an HTTP connection.
TProxyOptions	This class is used to establish an HTTP connection through a proxy server.

Enumerations

Name	Description
TIPVersion	Specifies Internet Protocol version.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1 Classes

Classes in the **CRVio** unit.

Classes

Name	Description
THttpOptions	This class is used to establish an HTTP connection.
TProxyOptions	This class is used to establish an HTTP connection through a proxy server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.1 THttpOptions Class

This class is used to establish an HTTP connection.

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

Unit

[CRVio](#)

Syntax

```
THttpOptions = class(TPersistent);
```

Remarks

The THttpOptions class is used to establish an HTTP connection.

For more information about HTTP tunneling, see [Network Tunneling](#) .

See Also

- [Network Tunneling](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.1.1 Members

[THttpOptions](#) class overview.

Properties

Name	Description
Enabled	Enables an HTTP connection.
Password	Holds the password for HTTP authorization.
ProxyOptions	Holds a TProxyOptions object that contains settings for a proxy connection.
TrustServerCertificate	Verifies the server certificate during an SSL handshake.
Url	Holds the URL of the PHP script for HTTP tunneling.
Username	Holds the username for HTTP authorization.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.1.2 Properties

Properties of the **THttpRequestOptions** class.

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

Public

Name	Description
Enabled	Enables an HTTP connection.
ProxyOptions	Holds a TProxyOptions object that contains settings for a proxy connection.

Published

Name	Description
Password	Holds the password for HTTP authorization.
TrustServerCertificate	Verifies the server certificate during an SSL handshake.
Url	Holds the URL of the PHP script for HTTP tunneling.
Username	Holds the username for HTTP authorization.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.1.2.1 Enabled Property

Enables an HTTP connection.

Class

[THttpRequestOptions](#)

Syntax

```
property Enabled: boolean default False;
```

Remarks

The Enabled property specifies that a connection is established through HTTP.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.1.2.2 Password Property

Holds the password for HTTP authorization.

Class

[THttpOptions](#)

Syntax

```
property Password: string;
```

Remarks

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

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.1.2.3 ProxyOptions Property

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

Class

[THttpOptions](#)

Syntax

```
property ProxyOptions: TProxyOptions;
```

Remarks

The ProxyOptions property holds a TProxyOptions object that contains settings for a proxy connection.

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.1.2.4 TrustServerCertificate Property

Verifies the server certificate during an SSL handshake.

Class

[THttpOptions](#)

Syntax

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

Remarks

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.1.2.5 Url Property

Holds the URL of the PHP script for HTTP tunneling.

Class

[THttpOptions](#)

Syntax

```
property Url: string;
```

Remarks

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.1.2.6 Username Property

Holds the username for HTTP authorization.

Class

[THttpOptions](#)

Syntax

```
property Username: string;
```

Remarks

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.2 TProxyOptions Class

This class is used to establish an HTTP connection through a proxy server.

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

Unit

[CRVio](#)

Syntax

```
TProxyOptions = class(TPersistent);
```

Remarks

The TProxyOptions class is used to establish an HTTP connection through a proxy server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.2.1 Members

[TProxyOptions](#) class overview.

Properties

Name	Description
Hostname	Holds the hostname or IP address of the proxy server.
Password	Holds the proxy password.
Port	Holds the port number of the proxy server.
Username	Holds the proxy username.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.2.2 Properties

Properties of the **TProxyOptions** class.

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

Published

Name	Description
Hostname	Holds the hostname or IP address of the proxy server.
Password	Holds the proxy password.
Port	Holds the port number of the proxy server.
Username	Holds the proxy username.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.2.2.1 Hostname Property

Holds the hostname or IP address of the proxy server.

Class

[TProxyOptions](#)

Syntax

```
property Hostname: string;
```

Remarks

The Hostname property holds the hostname or IP address of the proxy server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.2.2.2 Password Property

Holds the proxy password.

Class

[TProxyOptions](#)

Syntax

```
property Password: string;
```

Remarks

The Password property holds the proxy password.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.2.2.3 Port Property

Holds the port number of the proxy server.

Class

[TProxyOptions](#)

Syntax

```
property Port: integer default 0;
```

Remarks

Use the Port property to specify the port number of the proxy server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.1.2.2.4 Username Property

Holds the proxy username.

Class

[TProxyOptions](#)

Syntax

```
property Username: string;
```

Remarks

The Username property holds the proxy username.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.2 Enumerations

Enumerations in the **CRVio** unit.

Enumerations

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

TIPVersion	Specifies Internet Protocol version.
----------------------------	--------------------------------------

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.4.2.1 TIPVersion Enumeration

Specifies Internet Protocol version.

Unit

[CRVio](#)

Syntax

```
TIPVersion = (ivIPv4, ivIPv6, ivIPBoth);
```

Values

Value	Meaning
ivIPBoth	Specifies that either IPv6 or IPv4 Internet Protocol version is used
ivIPv4	Specifies that the IPv4 Internet Protocol version is used
ivIPv6	Specifies that the IPv6 Internet Protocol version is used

Remarks

Note: When the TIPVersion property is set to **ivIPBoth** , a connection attempt is made via IPv6 if it is enabled in the operating system settings. If the connection attempt fails, a new connection attempt is made via IPv4.

See Also

- [TPgConnectionOptions.IPVersion](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.5 CRXml

5.5.1 Structs

Structs in the **CRXml** unit.

Structs

Name	Description
TAttribute	TAttribute is not used in PgDAC.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.5.1.1 TAttribute Record

TAttribute is not used in PgDAC.

Unit

CRXml

Syntax

```
TAttribute = record;
```

Fields

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1 Classes

Classes in the **DAAlerter** unit.

Classes

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

[TDAAlerter](#)

A base class that defines functionality for database event notification.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1 TDAAlerter Class

A base class that defines functionality for database event notification.

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

Unit

[DAAlerter](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.1 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.2 Properties

Properties of the **TDAAlerter** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.2.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.3 Methods

Methods of the **TDAAlerter** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.3.1 SendEvent Method

Sends an event with Name and content Message.

Class

[TDAAlerter](#)

Syntax

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

Parameters

EventName

Holds the event name.

Message

Holds the content Message of the event.

Remarks

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.3.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.3.3 Stop Method

Stops waiting process.

Class

[TDAAlerter](#)

Syntax

```
procedure Stop;
```

Remarks

Call Stop method to end waiting process.

See Also

- [Start](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.4 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](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.1.1.4.1 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.2 Types

Types in the **DAAlerter** unit.

Types

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.6.2.1 TAlerterErrorEvent Procedure Reference

This type is used for the TDAlerter.OnError event.

Unit

[DAAlerter](#)

Syntax

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

Parameters

Sender

An object that raised the event.

E

Exception object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1 TDADump Class

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

a script.

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

Unit

[DADump](#)

Syntax

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

Remarks

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.1 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 the statement that is being executed and the values and types of its 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 , <code>M:Devart.Dac.TDADump.BackupToFile(System.String)</code> or <code>M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream)</code> 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.2 Properties

Properties of the **TDADump** class.

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

Public

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 the statement that is being executed and the values and types of its 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](#)

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.7.1.1.2.1 Connection Property

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

Class

[TDADump](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

See Also

- [TCustomDAConnection](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.2.2 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

[TDADump](#)

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

See Also

- [TCustomDADDataSet.Debug](#)
- [TCustomDASQL.Debug](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.2.3 Options Property

Used to specify the behaviour of a TDADump component.

Class

[TDADump](#)

Syntax

```
property options: TDADumpOptions;
```

Remarks

Use the Options property to specify the behaviour of a TDADump component.

Descriptions of all options are in the table below.

Option Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.

GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.2.4 SQL Property

Used to set or get the dump script.

Class

[TDADump](#)

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to get or set the dump script. The SQL property stores script that is executed by the [Restore](#) method. This property will store the result of [Backup](#) and [BackupQuery](#). At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

See Also

- [Restore](#)
- [Backup](#)
- [BackupQuery](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.2.5 TableNames Property

Used to set the names of the tables to dump.

Class

[TDADump](#)

Syntax

```
property TableNames: string;
```

Remarks

Use the TableNames property to set the names of the tables to dump. Table names must be separated with semicolons. If the property is empty, the [Backup](#) method will dump all available tables.

See Also

- [Backup](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.3 Methods

Methods of the **TDADump** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.3.1 Backup Method

Dumps database objects to the [SQL](#) property.

Class

[TDADump](#)

Syntax

```
procedure Backup;
```

Remarks

Call the Backup method to dump database objects. The result script will be stored in the [SQL](#) property.

See Also

- [SQL](#)
- [Restore](#)
- [BackupToFile](#)
- [BackupToStream](#)
- [BackupQuery](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.3.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.3.3 BackupToFile Method

Dumps database objects to the specified file.

Class

[TDADump](#)

Syntax

```
procedure BackupToFile(const FileName: string; const Query:  
string = '');
```

Parameters

FileName

Holds the file name to dump database objects to.

Query

Your query to receive the data for dumping.

Remarks

Call the BackupToFile method to dump database objects to the specified file.

See Also

- [RestoreFromStream](#)
- [Backup](#)
- [BackupToStream](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.3.4 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.3.5 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.3.6 RestoreFromFile Method

Executes a script from a file.

Class

[TDADump](#)

Syntax

```
procedure RestoreFromFile(const FileName: string);  
overload; procedure RestoreFromFile(const FileName: string;
```

```
Encoding: TEncoding); overload;
```

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.3.7 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.4 Events

Events of the **TDADump** class.

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

Published

Name	Description
OnBackupProgress	Occurs to indicate the TDADump.Backup , <code>M:Devart.Dac.TDADump.BackupToFile(System.String)</code> or <code>M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream)</code> 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.4.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.4.2 OnError Event

Occurs when PostgreSQL raises some error on [Restore](#).

Class

[TDADump](#)

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

The OnError event occurs when PostgreSQL raises some error on [Restore](#).

Action indicates the action to take when the OnError handler exits. On entry into the handler, Action is always set to eaException.

Note: You should add the DAScript module to the 'uses' list to use the OnError event handler.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.1.4.3 OnRestoreProgress Event

Occurs to indicate the [Restore](#), [RestoreFromFile](#), or [RestoreFromStream](#) method execution progress.

Class

[TDADump](#)

Syntax

```
property OnRestoreProgress: TDARestoreProgressEvent;
```

Remarks

The OnRestoreProgress event occurs several times during the dumping process of the [Restore](#), [RestoreFromFile](#), or [RestoreFromStream](#) method execution and indicates its progress. The Percent parameter of the OnRestoreProgress event handler indicates the percentage of the whole restore script execution.

See Also

- [Restore](#)
- [RestoreFromFile](#)
- [RestoreFromStream](#)

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.7.1.2 TDADumpOptions Class

This class allows setting up the behaviour of the TDADump class.

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

Unit

[DADump](#)

Syntax

```
TDADumpOptions = class(TPersistent);
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.2.1 Members

[TDADumpOptions](#) class overview.

Properties

Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.2.2 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.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.2.2.1 AddDrop Property

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

Class

[TDADumpOptions](#)

Syntax

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

Remarks

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.2.2.2 CompleteInsert Property

Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.

Class

[TDADumpOptions](#)

Syntax

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

Remarks

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

```
INSERT INTO dept(deptno, dname, loc) VALUES ('10', 'ACCOUNTING', 'NEW YORK');
```

If False, it won't include the field names, for example:

```
INSERT INTO dept VALUES ('10', 'ACCOUNTING', 'NEW YORK');
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.2.2.3 GenerateHeader Property

Used to add a comment header to a script.

Class

[TDADumpOptions](#)

Syntax

```
property GenerateHeader: boolean default True;
```

Remarks

Use the GenerateHeader property to add a comment header to a script. It contains script generation date, DAC version, and some other information.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.1.2.2.4 QuoteNames Property

Used for TDADump to quote all database object names in generated SQL statements.

Class

[TDADumpOptions](#)

Syntax

```
property QuoteNames: boolean default False;
```

Remarks

If the QuoteNames property is True, TDADump quotes all database object names in generated SQL statements.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.7.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.7.2.1 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.

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.7.2.2 TDARestoreProgressEvent Procedure Reference

This type is used for the [TDADump.OnRestoreProgress](#) event.

Unit

[DADump](#)

Syntax

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

Parameters

Sender

An object that raised the event.

Percent

The percentage of the whole restore script execution.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.1 TDAColumn Class

Represents the attributes for column loading.

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

Unit

[DALoader](#)

Syntax

```
TDAColumn = class(TCollectionItem);
```

Remarks

Each [TDALoader](#) uses [TDAColumns](#) to maintain a collection of TDAColumn objects.

TDAColumn object represents the attributes for column loading. Every TDAColumn object corresponds to one of the table fields with the same name as its [TDAColumn.Name](#) property.

To create columns at design-time use the column editor of the [TDALoader](#) component.

See Also

- [TDALoader](#)

- [TDAColumns](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.1.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.1.2 Properties

Properties of the **TDAColumn** class.

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

Published

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.1.2.1 FieldType Property

Used to specify the types of values that will be loaded.

Class

[TDAColumn](#)

Syntax

```
property FieldType: TFieldType default ftString;
```

Remarks

Use the FieldType property to specify the types of values that will be loaded. Field types for columns may not match data types for the corresponding fields in the database table.

[TDALoader](#) will cast data values to the types of their fields.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.1.2.2 Name Property

Used to specify the field name of loading table.

Class

[TDAColumn](#)

Syntax

```
property Name: string;
```

Remarks

Each TDAColumn corresponds to one field of the loading table. Use the Name property to specify the name of this field.

See Also

- [FieldType](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.2 TDAColumns Class

Holds a collection of [TDAColumn](#) objects.

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

Unit

[DALoader](#)

Syntax

```
TDAColumns = class(TOwnedCollection);
```

Remarks

Each TDAColumns holds a collection of [TDAColumn](#) objects. TDAColumns maintains an index of the columns in its Items array. The Count property contains the number of columns in the collection. At design-time, use the Columns editor to add, remove, or modify columns.

See Also

- [DALoader](#)
- [TDAColumn](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.2.1 Members

[TDAColumns](#) class overview.

Properties

Name	Description
Items	Used to access individual columns.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.2.2 Properties

Properties of the **TDAColumns** class.

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

Public

Name	Description
Items	Used to access individual columns.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.2.2.1 Items Property(Indexer)

Used to access individual columns.

Class

[TDAColumns](#)

Syntax

```
property Items[Index: integer]: TDAColumn; default;
```

Parameters

Index

Holds the Index of [TDAColumn](#) to refer to.

Remarks

Use the Items property to access individual columns. The value of the Index parameter corresponds to the Index property of [TDAColumn](#).

See Also

- [TDAColumn](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3 TDALoader Class

This class allows loading external data into database.

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

Unit

[DALoader](#)

Syntax

```
TDALoader = class (TComponent);
```

Remarks

TDALoader allows loading external data into database. To specify the name of loading table set the [TDALoader.TableName](#) property. Use the [TDALoader.Columns](#) property to access individual columns. Write the [TDALoader.OnGetColumnData](#) or [TDALoader.OnPutData](#) event handlers to read external data and pass it to the database. Call the [TDALoader.Load](#) method to start loading data.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.1 Members

[TDALoader](#) class overview.

Properties

Name	Description
Columns	Used to add a TDAColumn object for each field that will be loaded.
Connection	property. 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.2 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	property. 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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.2.1 Columns Property

Used to add a [TDAColumn](#) object for each field that will be loaded.

Class

[TDALoader](#)

Syntax

```
property Columns: TDAColumns stored IsColumnsStored;
```

Remarks

Use the Columns property to add a [TDAColumn](#) object for each field that will be loaded.

See Also

- [TDAColumns](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.2.2 Connection Property

property. Used to specify TCustomDACConnection in which TDALoader will be executed.

Class

[TDALoader](#)

Syntax

```
property Connection: TCustomDACConnection;
```

Remarks

Use the Connection property to specify TCustomDACConnection in which TDALoader will be executed. If Connection is not connected, the [Load](#) method calls [TCustomDACConnection.Connect](#).

See Also

- [TCustomDACConnection](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.2.3 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](#)

- [TCustomDACConnection.GetTableNames](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.3 Methods

Methods of the **TDALoader** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.3.1 CreateColumns Method

Creates [TDAColumn](#) objects for all fields of the table with the same name as [TableName](#).

Class

[TDALoader](#)

Syntax

```
procedure CreateColumns;
```

Remarks

Call the CreateColumns method to create [TDAColumn](#) objects for all fields of the table with the same name as [TableName](#). If columns were created before, they will be recreated. You can call CreateColumns from the component popup menu at design-time. After you can customize column loading by setting properties of TDAColumn objects.

See Also

- [TDAColumn](#)
- [TableName](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.3.2 Load Method

Starts loading data.

Class

[TDALoader](#)

Syntax

```
procedure Load; virtual;
```

Remarks

Call the Load method to start loading data. At first it is necessary to [create columns](#) and write one of the [OnPutData](#) or [OnGetColumnData](#) event handlers.

See Also

- [OnGetColumnData](#)
- [OnPutData](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.3.3 LoadFromDataSet Method

Loads data from the specified dataset.

Class

[TDALoader](#)

Syntax

```
procedure LoadFromDataSet(DataSet: TDataSet);
```

Parameters

DataSet

Holds the dataset to load data from.

Remarks

Call the LoadFromDataSet method to load data from the specified dataset. There is no need to create columns and write event handlers for [OnPutData](#) and [OnGetColumnData](#) before calling this method.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.3.4 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; Row: integer; const Value: variant)	Puts the value of individual columns by the column name.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.4 Events

Events of the **TDALoader** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.4.1 OnGetColumnData Event

Occurs when it is needed to put column values.

Class

[TDALoader](#)

Syntax

```
property OnGetColumnData: TGetColumnDataEvent;
```

Remarks

Write the OnGetColumnData event handler to put column values. [TDALoader](#) calls the OnGetColumnData event handler for each column in the loop. Column points to a [TDAColumn](#) object that corresponds to the current loading column. Use its Name or Index property to identify what column is loading. The Row parameter indicates the current loading record. TDALoader increments the Row parameter when all the columns of the current record are loaded. The first row is 1. Set EOF to True to stop data loading. Fill the Value parameter by column values. To start loading call the [Load](#) method.

Another way to load data is using the [OnPutData](#) event.

Example

This handler loads 1000 rows.

```
procedure 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.4.2 OnProgress Event

Occurs if handling data loading progress of the [LoadFromDataSet](#) method is needed.

Class

[TDALoader](#)

Syntax

```
property OnProgress: TLoaderProgressEvent;
```

Remarks

Add a handler to this event if you want to handle data loading progress of the [LoadFromDataSet](#) method.

See Also

- [LoadFromDataSet](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.3.4.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.4 TDALoaderOptions Class

Allows loading external data into database.

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

Unit

[DALoader](#)

Syntax

```
TDALoaderOptions = class(TPersistent);
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.4.1 Members

[TDALoaderOptions](#) class overview.

Properties

Name	Description
UseBlankValues	Forces PgDAC to fill the buffer with null values after loading a row to the database.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.4.2 Properties

Properties of the **TDALoaderOptions** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.1.4.2.1 UseBlankValues Property

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

Class

[TDALoaderOptions](#)

Syntax

```
property useBlankValues: boolean default True;
```

Remarks

Used to force PgDAC to fill the buffer with null values after loading a row to the database.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.2.1 TDAPutDataEvent Procedure Reference

This type is used for the [TDALoader.OnPutData](#) event.

Unit

[DLoader](#)

Syntax

```
TDAPutDataEvent = procedure (Sender: TDALoader) of object;
```

Parameters*Sender*

An object that raised the event.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.2.2 TGetColumnDataEvent Procedure Reference

This type is used for the [TDALoader.OnGetColumnData](#) event.

Unit

[DLoader](#)

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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.8.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1 Classes

Classes in the **DAScript** unit.

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1 TDAScript Class

Makes it possible to execute several SQL statements one by one.

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

Unit

[DAScript](#)

Syntax

```
TDAScript = class(TComponent);
```

Remarks

Often it is necessary to execute several SQL statements one by one. This can be performed using a lot of components such as [TCustomDASQL](#) descendants. Usually it isn't the best solution. With only one TDAScript descendant component you can execute several SQL statements as one. This sequence of statements is called script. To separate single statements use semicolon (;) or slash (/) and for statements that can contain semicolon, only slash. Note that slash must be the first character in line.

Errors that occur during execution can be processed in the [TDAScript.OnError](#) event handler. By default, on error TDAScript shows exception and continues execution.

See Also

- [TCustomDASQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.1 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.
FindMacro	Finds a macro with the specified name.
MacroByName	Finds a macro with the specified 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2 Properties

Properties of the **TDAScript** class.

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

Public

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.1 Connection Property

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

Class

[TDAScript](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify the connection in which the script will be executed. If Connection is not connected, the [Execute](#) method calls the Connect method of Connection.

Set at design-time by selecting from the list of provided [TCustomDAConnection](#) objects.

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

See Also

- [TCustomDAConnection](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.2 DataSet Property

Refers to a dataset that holds the result set of query execution.

Class

[TDAScript](#)

Syntax

```
property DataSet: TCustomDADataSet;
```

Remarks

Set the DataSet property to retrieve the results of the SELECT statements execution inside a script.

See Also

- [ExecuteNext](#)
- [Execute](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.3 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 the statement that is being executed and the values and types of its parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.5 EndLine Property

Used to get the current statement last line number in a script.

Class

[TDAScript](#)

Syntax

```
property EndLine: Int64;
```

Remarks

Use the EndLine property to get the current statement last line number in a script.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.6 EndOffset Property

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

Class

[TDAScript](#)

Syntax

```
property EndOffset: Int64;
```

Remarks

Use the EndOffset property to get the offset in the last line of the current statement.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.7 EndPos Property

Used to get the end position of the current statement.

Class

[TDAScript](#)

Syntax

```
property EndPos: Int64;
```

Remarks

Use the EndPos property to get the end position of the current statement (the position of the last character in the statement) in a script.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.8 Macros Property

Used to change SQL script text in design- or run-time easily.

Class

[TDAScript](#)

Syntax

```
property Macros: TMacros stored False;
```

Remarks

With the help of macros you can easily change SQL script text in design- or run-time. Macros extend abilities of parameters and allow changing conditions in the WHERE clause or sort order in the ORDER BY clause. You just insert &MacroName in a SQL query text and change value of macro by the Macro property editor in design-time or the MacroByName function in

run-time. In time of opening query macro is replaced by its value.

See Also

- [TMacro](#)
- [MacroByName](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.9 SQL Property

Used to get or set script text.

Class

[TDAScript](#)

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to get or set script text.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.10 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.11 StartOffset Property

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

Class

[TDAScript](#)

Syntax

```
property StartOffset: Int64;
```

Remarks

Use the StartOffset property to get the offset in the first line of the current statement.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.12 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.2.13 Statements Property

Contains a list of statements obtained from the SQL property.

Class

[TDAscript](#)

Syntax

```
property Statements: TDAStatements;
```

Remarks

Contains a list of statements that are obtained from the SQL property. Use the Access Statements property to view SQL statement, set parameters or execute the specified statement. Statements is a zero-based array of statement records. Index specifies the array element to access.

For example, consider the following script:

```
CREATE TABLE A (FIELD1 INTEGER);  
INSERT INTO A VALUES(1);  
INSERT INTO A VALUES(2);  
INSERT INTO A VALUES(3);  
CREATE TABLE B (FIELD1 INTEGER);  
INSERT INTO B VALUES(1);  
INSERT INTO B VALUES(2);  
INSERT INTO B VALUES(3);
```

Note: The list of statements is created and filled when the value of Statements property is requested. That's why the first access to the Statements property can take a long time.

Example

You can use the Statements property in the following way:

```
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;
```

See Also

- [TDAStatements](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.3 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.
FindMacro	Finds a macro with the specified name.
MacroByName	Finds a macro with the specified name.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.3.1 BreakExec Method

Stops script execution.

Class

[TDAScript](#)

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to stop script execution.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.3.2 ErrorOffset Method

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

Class

[TDAScript](#)

Syntax

```
function ErrorOffset: Int64;
```

Return Value

offset of an error.

Remarks

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

See Also

- [OnError](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.3.3 Execute Method

Executes a script.

Class

[TDAScript](#)

Syntax

```
procedure Execute; virtual;
```

Remarks

Call the Execute method to execute a script. If PostgreSQL raises an error, the OnError event occurs.

See Also

- [ExecuteNext](#)
- [OnError](#)
- [ErrorOffset](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.3.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.3.5 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.3.6 ExecuteStream Method

Executes SQL statements contained in a stream object.

Class

[TDAScript](#)

Syntax

```
procedure ExecuteStream(Stream: TStream);
```

Parameters

Stream

Holds the stream object from which the statements will be executed.

Remarks

Call the ExecuteStream method to execute SQL statements contained in a stream object. Reading from the stream and execution is performed by blocks of 64k size.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.3.7 FindMacro Method

Finds a macro with the specified name.

Class

[TDAScript](#)

Syntax

```
function FindMacro(Name: string): TMacro;
```

Parameters

Name

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the [TMacros.Items](#) property to avoid depending on the order of the items.

See Also

- [TMacro](#)

- [Macros](#)
- [MacroByName](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.3.8 MacroByName Method

Finds a macro with the specified name.

Class

[TDAScript](#)

Syntax

```
function MacroByName(Name: string): TMacro;
```

Parameters

Name

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the [TMacros.Items](#) property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the [TMacro.Value](#) property.

See Also

- [TMacro](#)
- [Macros](#)
- [FindMacro](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.4 Events

Events of the **TDAScript** class.

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

Published

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.4.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.4.2 BeforeExecute Event

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

Class

[TDAScript](#)

Syntax

```
property BeforeExecute: TBeforeStatementExecuteEvent;
```

Remarks

Write the BeforeExecute event handler to take specific action before executing the current SQL statement. SQL holds text of the current SQL statement. Write SQL to change the statement that will be executed. Set Omit to True to skip statement execution.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.1.4.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2 TDAScript 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 [TDAScript](#) members.

Unit

[TDAScript](#)

Syntax

```
TDAScript = class(TCollectionItem);
```

Remarks

TDAScript contains SQL statements, represented as TDAScript objects. The TDAScript class has attributes and methods for controlling single SQL statement of a script.

See Also

- [TDAScript](#)
- [TDAScripts](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.1 Members

[TDAScript](#) 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2 Properties

Properties of the **TDAStatement** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.9.1.2.2.1 EndLine Property

Used to determine the number of the last statement line in a script.

Class

[TDASTatement](#)

Syntax

```
property EndLine: integer;
```

Remarks

Use the EndLine property to determine the number of the last statement line in a script.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2.2 EndOffset Property

Used to get the offset in the last line of the statement.

Class

[TDASstatement](#)

Syntax

```
property EndOffset: integer;
```

Remarks

Use the EndOffset property to get the offset in the last line of the statement.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2.3 EndPos Property

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

Class

[TDASstatement](#)

Syntax

```
property EndPos: integer;
```

Remarks

Use the EndPos property to get the end position of the statement (the position of the last

character in the statement) in a script.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2.4 Omit Property

Used to avoid execution of a statement.

Class

[TDASstatement](#)

Syntax

```
property omit: boolean;
```

Remarks

Set the Omit property to True to avoid execution of a statement.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2.5 Params Property

Contains parameters for an SQL statement.

Class

[TDASstatement](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2.6 Script Property

Used to determine the TDAScript object the SQL Statement belongs to.

Class

[TDASstatement](#)

Syntax

```
property script: TDAScript;
```

Remarks

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2.7 SQL Property

Used to get or set the text of an SQL statement.

Class

[TDASstatement](#)

Syntax

```
property SQL: string;
```

Remarks

Use the SQL property to get or set the text of an SQL statement.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2.8 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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2.9 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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.2.10 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.2.3.1 Execute Method

Executes a statement.

Class

[TDASstatement](#)

Syntax

```
procedure Execute;
```

Remarks

Use the Execute method to execute a statement.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.3 TDASentences Class

Holds a collection of [TDASentence](#) objects.

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

Unit

[DAScript](#)

Syntax

```
TDASentences = class(TCollection);
```

Remarks

Each TDASentences holds a collection of [TDASentence](#) objects. TDASentences maintains an index of the sentences in its Items array. The Count property contains the number of sentences in the collection. Use TDASentences class to manipulate script SQL statements.

See Also

- [DAScript](#)
- [TDASentence](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.3.1 Members

[TDASentences](#) class overview.

Properties

Name	Description
Items	Used to access separate script statements.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.3.2 Properties

Properties of the **TDAStatements** class.

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

Public

Name	Description
Items	Used to access separate script statements.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.1.3.2.1 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 [TDASStatement](#).

See Also

- [TDASStatement](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.2.1 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.2.2 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.2.3 TOnErrorEvent Procedure Reference

This type is used for the [TDAScript.OnError](#) event.

Unit

[DAScript](#)

Syntax

```
TOnErrorEvent = procedure (Sender: TObject; E: Exception; SQL:
string; var Action: TErrorAction) of object;
```

Parameters

Sender

An object that raised the event.

E

The error code.

SQL

Holds the passed SQL statement.

Action

The action to take when the OnError handler exits.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.3 Enumerations

Enumerations in the **DAScript** unit.

Enumerations

Name	Description
TErrorAction	Indicates the action to take when the OnError handler exits.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.9.3.1 TErrorAction Enumeration

Indicates the action to take when the OnError handler exits.

Unit

[DAScript](#)

Syntax

```
TErrorAction = (eaAbort, eaFail, eaException, eaContinue);
```

Values

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10 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 displayed.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.1 TCustomDASQLMonitor Class

A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.

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

Unit

[DASQLMonitor](#)

Syntax

```
TCustomDASQLMonitor = class(TComponent);
```

Remarks

TCustomDASQLMonitor is a base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively. TCustomDASQLMonitor provides two ways of displaying debug information. It monitors either by dialog window or by Borland's proprietary SQL Monitor. Furthermore to receive debug information use the [TCustomDASQLMonitor.OnSQL](#) event.

In applications use descendants of TCustomDASQLMonitor.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.1.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.1.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.1.2.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.1.2.2 DBMonitorOptions Property

Used to set options for dbMonitor.

Class

[TCustomDASQLMonitor](#)

Syntax

```
property DBMonitorOptions: TDBMonitorOptions;
```

Remarks

Use DBMonitorOptions to set options for dbMonitor.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.1.2.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.1.2.4 TraceFlags Property

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

Class

[TCustomDASQLMonitor](#)

Syntax

```
property TraceFlags: TDATraceFlags default [tfQPrepare, tfQExecute, tfError, tfConnect, tfTransact, tfParams, tfMisc];
```

Remarks

Use the TraceFlags property to specify which database operations the monitor should track in an application at runtime.

See Also

- [OnSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.1.3 Events

Events of the **TCustomDASQLMonitor** class.

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

Public

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

[OnSQL](#)

Occurs when tracing of SQL activity on database components is needed.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.1.3.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.2 TDBMonitorOptions Class

This class holds options for dbMonitor.

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

Unit

[DASQLMonitor](#)

Syntax

```
TDBMonitorOptions = class(TPersistent);
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.2.2 Properties

Properties of the **TDBMonitorOptions** class.

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

Published

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.2.2.1 Host Property

Used to set the host name or IP address of the computer where dbMonitor application runs.

Class

[TDBMonitorOptions](#)

Syntax

```
property Host: string;
```

Remarks

Use the Host property to set the host name or IP address of the computer where dbMonitor application runs.

dbMonitor supports remote monitoring. You can run dbMonitor on a different computer than monitored application runs. In this case you need to set the Host property to the corresponding computer name.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.2.2.2 Port Property

Used to set the port number for connecting to dbMonitor.

Class

[TDBMonitorOptions](#)

Syntax

```
property Port: integer default DBMonitorPort;
```

Remarks

Use the Port property to set the port number for connecting to dbMonitor.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.1.2.2.3 ReconnectTimeout Property

Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.

Class

[TDBMonitorOptions](#)

Syntax

```
property ReconnectTimeout: integer default  
DefaultReconnectTimeout;
```

Remarks

Use the ReconnectTimeout property to set the minimum time (in milliseconds) that should be spent before allowing reconnecting to dbMonitor. If an error occurs when the component sends an event to dbMonitor (dbMonitor is not running), next events are ignored and the component does not restore the connection until ReconnectTimeout is over.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.10.1.2.2.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.2.1 TDATraceFlags Set

Represents the set of [TDATraceFlag](#).

Unit

[DASQLMonitor](#)

Syntax

```
TDATraceFlags = set of TDATraceFlag;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.2.2 TMonitorOptions Set

Represents the set of [TMonitorOption](#).

Unit

[DASQLMonitor](#)

Syntax

```
TMonitorOptions = set of TMonitorOption;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.2.3 TOnSQLEvent Procedure Reference

This type is used for the [TCustomDASQLMonitor.OnSQL](#) event.

Unit

[DASQLMonitor](#)

Syntax

```
TOnSQLEvent = procedure (Sender: TObject; Text: string; Flag: TDATraceFlag) of object;
```

Parameters

Sender

An object that raised the event.

Text

Holds the detected SQL statement.

Flag

Use the Flag parameter to make selective processing of SQL in the handler body.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.3.1 TDATraceFlag Enumeration

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

Unit

[DASQLMonitor](#)

Syntax

```
TDATraceFlag = (tfQPrepare, tfQExecute, tfQFetch, tfError, tfStmt,
tfConnect, tfTransact, tfBlob, tfService, tfMisc, tfParams,
tfObjDestroy, tfPool);
```

Values

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.10.3.2 TMonitorOption Enumeration

Used to define where information from SQLMonitor will be displayed.

Unit

[DASQLMonitor](#)

Syntax

```
TMonitorOption = (moDialog, moSQLMonitor, moDBMonitor, moCustom, moHandled);
```

Values

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11 DBAccess

This unit contains base classes for most of the components.

Classes

Name	Description
EDAError	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.
TCustomDAConnection	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.
TDACConnectionOptions	This class allows setting up the behaviour of the TDACConnection class.
TDACConnectionSSLOptions	This class is used to set up the SSL options.
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.
------------------------------------	---

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.OnConnectionLost 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.AfterUpdateExecute and TCustomDADataset.BeforeUpdateExecute events.

Enumerations

Name	Description
TLabelSet	Sets the language of labels in the connect dialog.
TLockMode	Specifies the lock mode.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1 Classes

Classes in the **DBAccess** unit.

Classes

Name	Description
EDAError	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.
TDAConnectionSSLOptions	This class is used to set up the SSL options.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.1 EDAError Class

A base class for exceptions that are raised when an error occurs on the server side.

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

Unit

[DBAccess](#)

Syntax

```
EDAEError = class(EDatabaseError);
```

Remarks

EDAEError is a base class for exceptions that are raised when an error occurs on the server side.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.1.1 Members

[EDAEError](#) class overview.

Properties

Name	Description
Component	Contains the component that caused the error.
ErrorCode	Determines the error code returned by the server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.1.2 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](#)

- [EDAError Class Members](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.1.2.1 Component Property

Contains the component that caused the error.

Class

[EDAError](#)

Syntax

```
property Component: TObject;
```

Remarks

The Component property contains the component that caused the error.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.1.2.2 ErrorCode Property

Determines the error code returned by the server.

Class

[EDAError](#)

Syntax

```
property ErrorCode: integer;
```

Remarks

Use the ErrorCode property to determine the error code returned by PostgreSQL. This value is always positive.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.2 TCRDataSource Class

Provides an interface between a DAC dataset components and data-aware controls on a form.

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

Unit

[DBAccess](#)

Syntax

```
TCRDataSource = class(TDataSource);
```

Remarks

TCRDataSource provides an interface between a DAC dataset components and data-aware controls on a form.

TCRDataSource inherits its functionality directly from the TDataSource component.

At design time assign individual data-aware components' DataSource properties from their drop-down listboxes.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.2.1 Members

[TCRDataSource](#) class overview.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3 TCustomConnectDialog Class

A base class for the connect dialog components.

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

Unit

[DBAccess](#)

Syntax

```
TCustomConnectDialog = class(TComponent);
```

Remarks

TCustomConnectDialog is a base class for the connect dialog components. It provides functionality to show a dialog box where user can edit username, password and server name before connecting to a database. You can customize captions of buttons and labels by their properties.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2 Properties

Properties of the **TCustomConnectDialog** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.1.3.2.2 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.3 ConnectButton Property

Used to specify the label for the Connect button.

Class

[TCustomConnectDialog](#)

Syntax

```
property ConnectButton: string;
```

Remarks

Use the ConnectButton property to specify the label for the Connect button.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.4 DialogClass Property

Used to specify the class of the form that will be displayed to enter login information.

Class

[TCustomConnectDialog](#)

Syntax

```
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.5 LabelSet Property

Used to set the language of buttons and labels captions.

Class

[TCustomConnectDialog](#)

Syntax

```
property LabelSet: TLabelSet default IsEnglish;
```

Remarks

Use the LabelSet property to set the language of labels and buttons captions.

The default value is IsEnglish.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.6 PasswordLabel Property

Used to specify a prompt for password edit.

Class

[TCustomConnectDialog](#)

Syntax

```
property PasswordLabel: string;
```

Remarks

Use the PasswordLabel property to specify a prompt for password edit.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.7 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.8 SavePassword Property

Used for the password to be displayed in ConnectDialog in asterisks.

Class

[TCustomConnectDialog](#)

Syntax

```
property SavePassword: boolean default False;
```

Remarks

If True, and the Password property of the connection instance is assigned, the password in ConnectDialog is displayed in asterisks.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.9 ServerLabel Property

Used to specify a prompt for the server name edit.

Class

[TCustomConnectDialog](#)

Syntax

```
property ServerLabel: string;
```

Remarks

Use the ServerLabel property to specify a prompt for the server name edit.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.10 StoreLogInfo Property

Used to specify whether the login information should be kept in system registry after a connection was established.

Class

[TCustomConnectDialog](#)

Syntax

```
property StoreLogInfo: boolean default True;
```

Remarks

Use the StoreLogInfo property to specify whether to keep login information in system registry after a connection was established using provided username, password and servername.

Set this property to True to store login information.

The default value is True.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.2.11 UsernameLabel Property

Used to specify a prompt for username edit.

Class

[TCustomConnectDialog](#)

Syntax

```
property UsernameLabel: string;
```

Remarks

Use the UsernameLabel property to specify a prompt for username edit.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.3 Methods

Methods of the **TCustomConnectDialog** class.

For a complete list of the **TCustomConnectDialog** class members, see the

[TCustomConnectDialog Members](#) topic.

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.3.1 Execute Method

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.

Class

[TCustomConnectDialog](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.3.3.2 GetServerList Method

Retrieves a list of available server names.

Class

[TCustomConnectDialog](#)

Syntax

```
procedure GetServerList(List: TStrings); virtual;
```

Parameters

List

Holds a list of available server names.

Remarks

Call the GetServerList method to retrieve a list of available server names. It is particularly relevant for writing custom login form.

See Also

- [DialogClass](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4 TCustomDACConnection Class

A base class for components used to establish connections.

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

Unit

[DBAccess](#)

Syntax

```
TCustomDACConnection = class(TCustomConnection);
```

Remarks

TCustomDACConnection is a base class for components that establish connection with database, provide customised login support, and perform transaction control.

Do not create instances of TCustomDACConnection. To add a component that represents a connection to a source of data, use descendants of the TCustomDACConnection class.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.11.1.4.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2 Properties

Properties of the **TCustomDACConnection** class.

For a complete list of the **TCustomDACConnection** class members, see the [TCustomDACConnection 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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.1 ConnectDialog Property

Allows to link a [TCustomConnectDialog](#) component.

Class

[TCustomDACConnection](#)

Syntax

```
property ConnectDialog: TCustomConnectDialog;
```

Remarks

Use the ConnectDialog property to assign to connection a [TCustomConnectDialog](#) component.

See Also

- [TCustomConnectDialog](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.2 ConnectString Property

Used to specify the connection information, such as: UserName, Password, Server, etc.

Class

[TCustomDAConnection](#)

Syntax

```
property ConnectString: string stored False;
```

Remarks

PgDAC recognizes an ODBC-like syntax in provider string property values. Within the string, elements are delimited by using a semicolon. Each element consists of a keyword, an equal sign character, and the value passed on initialization. For example:

```
Server=London1;User ID=nancyd
```

Connection parameters

The following connection parameters can be used to customize connection:

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.
P:Devart.Pgdac.TPgConnectionSSLOptions.CACert	Holds the pathname to the certificate authority file.
P:Devart.Pgdac.TPgConnectionSSLOptions.Cert	Holds the pathname to the certificate file.
P:Devart.Pgdac.TPgConnectionSSLOptions.CipherList	Holds the list of allowed ciphers to use for SSL encryption.
P:Devart.Pgdac.TPgConnectionSSLOptions.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](#)

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.4 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.5 LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Class

[TCustomDACConnection](#)

Syntax

```
property LoginPrompt default DefValLoginPrompt;
```

Remarks

Specifies whether a login dialog appears immediately before opening a new connection. If [ConnectDialog](#) is not specified, the default connect dialog will be shown. The connect dialog will appear only if the PgDacVcl unit appears to the uses clause.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.6 Options Property

Specifies the connection behavior.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.7 Password Property

Serves to supply a password for login.

Class

[TCustomDAConnection](#)

Syntax

```
property Password: string stored False;
```

Remarks

Use the Password property to supply a password to handle server's request for a login.

Warning: Storing hard-coded user name and password entries as property values or in code

for the OnLogin event handler can compromise server security.

See Also

- [Username](#)
- [Server](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.8 Pooling Property

Enables or disables using connection pool.

Class

[TCustomDACConnection](#)

Syntax

```
property Pooling: boolean default DefValPooling;
```

Remarks

Normally, when TCustomDACConnection 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. TCustomDACConnection 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.9 PoolingOptions Property

Specifies the behaviour of connection pool.

Class

[TCustomDACConnection](#)

Syntax

```
property PoolingOptions: TPoolingOptions;
```

Remarks

Set the properties of PoolingOptions to specify the behaviour of connection pool.

Descriptions of all options are in the table below.

Option Name	Description
ConnectionLifetime	Used to specify the maximum time during which an open 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.
PoolId	Used to specify an ID for a connection pool.
Validate	Used for a connection to be validated when

it is returned from the pool.

See Also

- [Pooling](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.10 Server Property

Serves to supply the server name for login.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.2.11 Username Property

Used to supply a user name for login.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.1 ApplyUpdates Method

Applies changes in datasets.

Class

[TCustomDAConnection](#)

Overload List

Name	Description
ApplyUpdates	Applies changes from all active datasets.
ApplyUpdates(const DataSets: array of TCustomDADataSet)	Applies changes from the specified datasets.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.3 Connect Method

Establishes a connection to the server.

Class

[TCustomDAConnection](#)

Syntax

```
procedure Connect; overload; procedure Connect(const
ConnectString: string); overload;
```

Remarks

Call the Connect method to establish a connection to the server. Connect sets the Connected property to True. If LoginPrompt is True, Connect prompts user for login information as required by the server, or otherwise tries to establish a connection using values provided in the [Username](#), [Password](#), and [Server](#) properties.

See Also

- [Disconnect](#)
- [Username](#)
- [Password](#)
- [Server](#)
- [ConnectDialog](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.4 CreateSQL Method

Creates a component for queries execution.

Class

[TCustomDACConnection](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.5 Disconnect Method

Performs disconnect.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.6 ExecProc Method

Allows to execute stored procedure or function providing its name and parameters.

Class

[TCustomDAConnection](#)

Syntax

```
function ExecProc(const Name: string; const Params: array of  
variant): variant; virtual;
```

Parameters

Name

Holds the name of the stored procedure or function.

Params

Holds the parameters of the stored procedure or function.

Return Value

the result of the stored procedure.

Remarks

Allows to execute stored procedure or function providing its name and parameters.

Use the following Name value syntax for executing specific overloaded routine:

"StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.

Assign parameters' values to the Params array in exactly the same order and number as they

appear in the stored procedure declaration. Out parameters of the procedure can be accessed with the ParamByName procedure.

If the value of an input parameter was not included to the Params array, parameter default value is taken. Only parameters at the end of the list can be unincluded to the Params array. If the parameter has no default value, the NULL value is sent.

Note: Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. The stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For further examples of parameter usage see [ExecSQL](#), [ExecSQLEx](#).

Example

For example, having stored function declaration presented in Example 1), you may execute it and retrieve its result with commands presented in Example 2):

```
Example 1)
CREATE procedure MY_SUM (
    A INTEGER,
    B INTEGER)
RETURNS (
    RESULT INTEGER)
as
begin
    Result = a + b;
end;
Example 2)
Label1.Caption:= MyPgConnection1.ExecProc('My_Sum', [10, 20]);
Label2.Caption:= MyPgConnection1.ParamByName('Result').AsString;
```

See Also

- [ExecProcEx](#)
- [ExecSQL](#)
- [ExecSQLEx](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.7 ExecProcEx Method

Allows to execute a stored procedure or function.

Class

[TCustomDAConnection](#)

Syntax

```
function ExecProcEx(const Name: string; const Params: array of  
variant): variant; virtual;
```

Parameters

Name

Holds the stored procedure name.

Params

Holds an array of pairs of parameters' names and values.

Return Value

the result of the stored procedure.

Remarks

Allows to execute a stored procedure or function. Provide the stored procedure name and its parameters to the call of ExecProcEx.

Use the following Name value syntax for executing specific overloaded routine:

"StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.

Assign pairs of parameters' names and values to a Params array so that every name comes before its corresponding value when an array is being indexed.

Out parameters of the procedure can be accessed with the ParamByName procedure. If the value for an input parameter was not included to the Params array, the parameter default value is taken. If the parameter has no default value, the NULL value is sent.

Note: Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. Stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For an example of parameters usage see [ExecSQLEx](#).

Example

If you have some stored procedure accepting four parameters, and you want to provide values only for the first and fourth parameters, you should call ExecProcEx in the following way:

```
Connection.ExecProcEx('Some_Stored_Procedure', ['Param_Name1', 'Param_Value1
```

See Also

- [ExecSQL](#)
- [ExecSQLEx](#)
- [ExecProc](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.8 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.9 ExecSQLEx Method

Executes any SQL statement outside the TQuery or TSQL components.

Class

[TCustomDAConnection](#)

Syntax

```
function ExecSQLEx(const Text: string; const Params: array of
variant): variant; virtual;
```

Parameters

Text

a SQL statement to be executed.

Params

Array of parameter values arranged in the same order as they appear in SQL statement.

Return Value

Out parameter with the name Result will hold the result of a function having data type dtString. Otherwise returns Null.

Remarks

Call the ExecSQLEx method to execute any SQL statement outside the TQuery or TSQL components. Supply the Params array with values arranged in pairs of parameter name and its value. This way each parameter name in the array is found on even index values whereas

parameter value is on odd index value but right after its parameter name. The parameter pairs must be arranged according to their occurrence in a SQL statement which itself is passed in the Text string parameter.

The Params array must contain all IN and OUT parameters defined in the SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the ExecSQLEx method.

Out parameter with the name Result will hold the result of a function having data type dtString. If neither of the parameters in the Text statement is named Result, ExecSQLEx will return Null.

To get the values of OUT parameters use the ParamByName function.

Example

```
PgConnection.ExecSQLEX('begin :A:= :B + :C; end;',  
    ['A', 0, 'B', 5, 'C', 3]);  
A:= PgConnection.ParamByName('A').AsInteger;
```

See Also

- [ExecSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.10 GetDatabaseNames Method

Returns a database list from the server.

Class

[TCustomDAConnection](#)

Syntax

```
procedure GetDatabaseNames(List: TStrings); virtual;
```

Parameters

List

A TStrings descendant that will be filled with database names.

Remarks

Populates a string list with the names of databases.

Note: Any contents already in the target string list object are eliminated and overwritten by data produced by `GetDatabaseNames`.

See Also

- [GetTableNames](#)
- [GetStoredProcNames](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.11 GetKeyFieldNames Method

Provides a list of available key field names.

Class

[TCustomDAConnection](#)

Syntax

```
procedure GetKeyFieldNames(const TableName: string; List: TStrings); virtual;
```

Parameters

TableName

Holds the table name

List

The list of available key field names

Return Value

Key field name

Remarks

Call the `GetKeyFieldNames` method to get the names of available key fields. Populates a string list with the names of key fields in tables.

See Also

- [GetTableNames](#)

- [GetStoredProcNames](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.12 GetStoredProcNames Method

Returns a list of stored procedures from the server.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.11.1.4.3.13 GetTableNames Method

Provides a list of available tables names.

Class

[TCustomDAConnection](#)

Syntax

```
procedure GetTableNames(List: TStrings; AllTables: boolean = False; OnlyTables: boolean = False); virtual;
```

Parameters

List

A TStrings descendant that will be filled with table names.

AllTables

True, if procedure returns all table names including the names of system tables to the List parameter.

OnlyTables

Remarks

Call the GetTableNames method to get the names of available tables. Populates a string list with the names of tables in the database. If AllTables = True, procedure returns all table names including the names of system tables to the List parameter, otherwise List will not contain the names of system tables. If AllTables = True, the procedure returns to the List parameter the names of the tables that belong to all schemas; otherwise, List will contain the names of the tables that belong to the current schema.

Note: Any contents already in the target string list object are eliminated and overwritten by the data produced by GetTableNames.

See Also

- [GetDatabaseNames](#)
- [GetStoredProcNames](#)

Reserved.

5.11.1.4.3.14 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](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.15 Ping Method

Used to check state of connection to the server.

Class

[TCustomDAConnection](#)

Syntax

```
procedure Ping;
```

Remarks

The method is used for checking server connection state.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.16 RemoveFromPool Method

Marks the connection that should not be returned to the pool after disconnect.

Class

[TCustomDAConnection](#)

Syntax

```
procedure RemoveFromPool;
```

Remarks

Call the RemoveFromPool method to mark the connection that should be deleted after disconnect instead of returning to the connection pool.

See Also

- [Pooling](#)
- [PoolingOptions](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.17 Rollback Method

Discards all current data changes and ends transaction.

Class

[TCustomDAConnection](#)

Syntax

```
procedure Rollback; virtual;
```

Remarks

Call the Rollback method to discard all updates, insertions, and deletions of data associated

with the current transaction to the database server and then end the transaction. The current transaction is the last transaction started by calling [StartTransaction](#).

See Also

- [Commit](#)
- [StartTransaction](#)
- [TCustomPgDataSet.FetchAll](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.3.18 StartTransaction Method

Begins a new user transaction.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.4 Events

Events of the **TCustomDACConnection** class.

For a complete list of the **TCustomDACConnection** class members, see the [TCustomDACConnection 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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.4.1 OnConnectionLost Event

This event occurs when connection was lost.

Class

[TCustomDACConnection](#)

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 TCustomDACConnection.Options.LocalFailover property to True.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.4.4.2 OnError Event

This event occurs when an error has arisen in the connection.

Class

[TCustomDACConnection](#)

Syntax

```
property OnError: TDAConnectionErrorEvent;
```

Remarks

Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5 TCustomDADataSet Class

Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.

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

Unit

[DBAccess](#)

Syntax

```
TCustomDADataset = class(TMemDataSet);
```

Remarks

TCustomDADataset encapsulates general set of properties, events, and methods for working with data accessed through various database engines. All database-specific features are supported by descendants of TCustomDADataset.

Applications should not use TCustomDADataset objects directly.

Inheritance Hierarchy

[TMemDataSet](#)

TCustomDADataset

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.1 Members

[TCustomDADataset](#) class overview.

Properties

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 the statement that is being executed and the values and types of its 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.
FinalSQL	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	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 find out whether TCustomDADataset has 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	Finds a macro with the specified name.
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 specified 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2 Properties

Properties of the **TCustomDADataset** class.

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

Public

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 the statement that is being executed and the values and types of its 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.
FinalSQL	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.1 BaseSQL Property

Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.

Class

[TCustomDADataset](#)

Syntax

```
property BaseSQL : string;
```

Remarks

Use the BaseSQL property to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL, only macros are expanded. SQL text with all these changes can be returned by [FinalSQL](#).

See Also

- [FinalSQL](#)
- [AddWhere](#)
- [SaveSQL](#)
- [SQLSaved](#)
- [RestoreSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.2 Conditions Property

Used to add WHERE conditions to a query

Class

[TCustomDADataset](#)

Syntax

```
property Conditions: TDAConditions stored False;
```

See Also

- [TDAConditions](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.3 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

[TCustomDADataset](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.4 DataTypeMap Property

Used to set data type mapping rules

Class

[TCustomDADataset](#)

Syntax

```
property DataTypeMap: TDAMapRules stored IsMapRulesStored;
```

See Also

- [TDAMapRules](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.5 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

[TCustomDADataset](#)

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.6 DetailFields Property

Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

Class

[TCustomDADataset](#)

Syntax

```
property DetailFields: string;
```

Remarks

Use the DetailFields property to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship. DetailFields is a string containing one or more field names in the detail table. Separate field names with semicolons.

Use Field Link Designer to set the value in design time.

See Also

- [MasterFields](#)
- [MasterSource](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.7 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.8 FetchRows Property

Used to define the number of rows to be transferred across the network at the same time.

Class

[TCustomDADataset](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.9 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.10 FinalSQL Property

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Class

[TCustomDADataset](#)

Syntax

```
property FinalSQL: string;
```

Remarks

Use FinalSQL to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros. This is the exact statement that will be passed on to the database server.

See Also

- [FinalSQL](#)
- [AddWhere](#)
- [SaveSQL](#)
- [SQLSaved](#)
- [RestoreSQL](#)
- [BaseSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.11 IsQuery Property

Used to check whether SQL statement returns rows.

Class

[TCustomDADataset](#)

Syntax

```
property IsQuery: boolean;
```

Remarks

After the TCustomDADataset component is prepared, the IsQuery property returns True if SQL statement is a SELECT query.

Use the IsQuery property to check whether the SQL statement returns rows or not.

IsQuery is a read-only property. Reading IsQuery on unprepared dataset raises an exception.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.12 KeyFields Property

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.

Class

[TCustomDADataset](#)

Syntax

```
property KeyFields: string;
```

Remarks

TCustomDADataset uses the KeyFields property to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database. For this feature KeyFields may hold a list of semicolon-delimited field names. If KeyFields is not defined before opening a dataset, TCustomDADataset requests information about primary keys by sending an additional query to the server.

See Also

- [SQLDelete](#)
- [SQLInsert](#)
- [SQLRefresh](#)
- [SQLUpdate](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.13 MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

[TCustomDADataset](#)

Syntax

```
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also

- [Macros](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.14 Macros Property

Makes it possible to change SQL queries easily.

Class

[TCustomDADataset](#)

Syntax

```
property Macros: TMacros stored False;
```

Remarks

With the help of macros you can easily change SQL query text at design- or runtime. Macros 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.Text := 'SELECT * FROM Dept ORDER BY &Order';  
PgQuery.MacroByName('Order').Value := 'DeptNo';  
PgQuery.Open;
```

See Also

- [TMacro](#)
- [MacroByName](#)
- [Params](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.15 MasterFields Property

Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

Class

[TCustomDADataset](#)

Syntax

```
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.16 MasterSource Property

Used to specify the data source component which binds current dataset to the master one.

Class

[TCustomDADataset](#)

Syntax

```
property MasterSource: TDataSource;
```

Remarks

The MasterSource property specifies the data source component which binds current dataset to the master one.

TCustomDADataset uses MasterSource to extract foreign key fields values from the master dataset when building master/detail relationship between two datasets. MasterSource must point to another dataset; it cannot point to this dataset component.

When MasterSource is not **nil** dataset fills parameter values with corresponding field values from the current record of the master dataset.

Note: Do not set the DataSource property when building master/detail relationships. Although it points to the same object as the MasterSource property, it may lead to undesirable results.

See Also

- [MasterFields](#)
- [DetailFields](#)
- [Master/Detail Relationships](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.17 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.18 ParamCheck Property

Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Class

[TCustomDADataset](#)

Syntax

```
property ParamCheck: boolean default True;
```

Remarks

Use the ParamCheck property to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Set ParamCheck to True to let dataset automatically generate the Params property for the dataset based on a SQL statement.

Setting ParamCheck to False can be used if the dataset component passes to a server the

DDL statements that contain, for example, declarations of stored procedures which themselves will accept parameterized values. The default value is True.

See Also

- [Params](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.19 ParamCount Property

Used to indicate how many parameters are there in the Params property.

Class

[TCustomDADataset](#)

Syntax

```
property ParamCount: word;
```

Remarks

Use the ParamCount property to determine how many parameters are there in the Params property.

See Also

- [Params](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.20 Params Property

Used to view and set parameter names, values, and data types dynamically.

Class

[TCustomDADataset](#)

Syntax

```
property Params: TDAParams stored False;
```

Remarks

Contains the parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information).

Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

See Also

- [ParamByName](#)
- [Macros](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.21 ReadOnly Property

Used to prevent users from updating, inserting, or deleting data in the dataset.

Class

[TCustomDADataset](#)

Syntax

```
property ReadOnly: boolean default False;
```

Remarks

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is False, meaning that users can potentially alter data stored in the dataset.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to True.

When ReadOnly is True, the dataset's CanModify property is False.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.22 RefreshOptions Property

Used to indicate when the editing record is refreshed.

Class

[TCustomDADataset](#)

Syntax

```
property RefreshOptions: TRefreshOptions default [];
```

Remarks

Use the RefreshOptions property to determine when the editing record is refreshed.

Refresh is performed by the [RefreshRecord](#) method.

It queries the current record and replaces one in the dataset. Refresh record is useful when the table has triggers or the table fields have default values. Use roBeforeEdit to get actual data before editing.

The default value is [].

See Also

- [RefreshRecord](#)

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.23 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

[TCustomDADataset](#)

Syntax

```
property RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.24 SQL Property

Used to provide a SQL statement that a query component executes when its Open method is called.

Class

[TCustomDADataset](#)

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to provide a SQL statement that a query component executes when its Open method is called. At the design time the SQL property can be edited by invoking the String List editor in Object Inspector.

When SQL is changed, TCustomDADataset calls Close and UnPrepare.

See Also

- [SQLInsert](#)
- [SQLUpdate](#)
- [SQLDelete](#)
- [SQLRefresh](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.25 SQLDelete Property

Used to specify a SQL statement that will be used when applying a deletion to a record.

Class

[TCustomDADataset](#)

Syntax

```
property SQLDelete: TStrings;
```

Remarks

Use the SQLDelete property to specify the SQL statement that will be used when applying a deletion to a record. Statements can be parameterized queries.

To create a SQLDelete statement at design-time, use the query statements editor.

Example

```
DELETE FROM Orders  
WHERE  
    OrderID = :Old_OrderID
```

See Also

- [SQL](#)
- [SQLInsert](#)
- [SQLUpdate](#)
- [SQLRefresh](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.26 SQLInsert Property

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Class

[TCustomDADataset](#)

Syntax

```
property SQLInsert: TStrings;
```

Remarks

Use the SQLInsert property to specify the SQL statement that will be used when applying an insertion to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. Parameters prefixed with OLD_ allow using current values of fields prior to the actual operation.

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.27 SQLLock Property

Used to specify a SQL statement that will be used to perform a record lock.

Class

[TCustomDADataset](#)

Syntax

```
property SQLLock: TStrings;
```

Remarks

Use the SQLLock property to specify a SQL statement that will be used to perform a record lock. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

To create a SQLLock statement at design-time, the use query statement editor.

See Also

- [SQL](#)
- [SQLInsert](#)
- [SQLUpdate](#)
- [SQLDelete](#)
- [SQLRefresh](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.28 SQLRecCount Property

Used to specify the SQL statement that is used to get the record count when opening a dataset.

Class

[TCustomDADataset](#)

Syntax

```
property SQLRecCount: TStrings;
```

Remarks

Use the SQLRecCount property to specify the SQL statement that is used to get the record count when opening a dataset. The SQL statement is used if the TDADatasetOptions.QueryRecCount property is True, and the TCustomDADataset.FetchAll property is False. Is not used if the FetchAll property is True.

To create a SQLRecCount statement at design-time, use the query statements editor.

See Also

- [SQLInsert](#)
- [SQLUpdate](#)
- [SQLDelete](#)

- [SQLRefresh](#)
- [TDADatasetOptions](#)
- [FetchingAll](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.29 SQLRefresh Property

Used to specify a SQL statement that will be used to refresh current record by calling the [RefreshRecord](#) procedure.

Class

[TCustomDADataset](#)

Syntax

```
property SQLRefresh: TStrings;
```

Remarks

Use the SQLRefresh property to specify a SQL statement that will be used to refresh current record by calling the [RefreshRecord](#) procedure.

Different behavior is observed when the SQLRefresh property is assigned with a single WHERE clause that holds frequently altered search condition. In this case the WHERE clause from SQLRefresh is combined with the same clause of the SELECT statement in a SQL property and this final query is then sent to the database server.

To create a SQLRefresh statement at design-time, use the query statements editor.

Example

```
SELECT Shipname FROM Orders
WHERE
    OrderID = :OrderID
```

See Also

- [RefreshRecord](#)
- [SQL](#)

- [SQLInsert](#)
- [SQLUpdate](#)
- [SQLDelete](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.30 SQLUpdate Property

Used to specify a SQL statement that will be used when applying an update to a dataset.

Class

[TCustomDADataset](#)

Syntax

```
property SQLUpdate: TStrings;
```

Remarks

Use the SQLUpdate property to specify a SQL statement that will be used when applying an update to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.2.31 UniDirectional Property

Used if an application does not need bidirectional access to records in the result set.

Class

[TCustomDADataset](#)

Syntax

```
property UniDirectional: boolean default False;
```

Remarks

Traditionally SQL cursors are unidirectional. They can travel only forward through a dataset. TCustomDADataset, however, permits bidirectional travelling by caching records. If an application does not need bidirectional access to the records in the result set, set UniDirectional to True. When UniDirectional is True, an application requires less memory and performance is improved. However, UniDirectional datasets cannot be modified. In FetchAll=False mode data is fetched on demand. When UniDirectional is set to True, data is fetched on demand as well, but obtained rows are not cached except for the current row. In case if the Unidirectional property is True, the FetchAll property will be automatically set to False. And if the FetchAll property is True, the Unidirectional property will be automatically set to False. The default value of UniDirectional is False, enabling forward and backward navigation.

Note: Pay attention to the specificity of using the FetchAll property=False

See Also

- [TPgQuery.FetchAll](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3 Methods

Methods of the **TCustomDADataset** class.

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

Public

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 find out whether TCustomDADataset has 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	Finds a macro with the specified name.
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 specified 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](#)

Devart. All Rights Reserved.

5.11.1.5.3.1 AddWhere Method

Adds condition to the WHERE clause of SELECT statement in the SQL property.

Class

[TCustomDADataset](#)

Syntax

```
procedure AddWhere(const Condition: string);
```

Parameters

Condition

Holds the condition that will be added to the WHERE clause.

Remarks

Call the AddWhere method to add a condition to the WHERE clause of SELECT statement in the SQL property.

If SELECT has no WHERE clause, AddWhere creates it.

Note: the AddWhere method is implicitly called by [RefreshRecord](#). The AddWhere method works for the SELECT statements only.

Note: the AddWhere method adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

See Also

- [DeleteWhere](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.2 BreakExec Method

Breaks execution of the SQL statement on the server.

Class

[TCustomDADataset](#)

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to break execution of the SQL statement on the server. It makes sense to only call BreakExec from another thread.

See Also

- [TCustomDADataset.Execute](#)
- [TCustomDASQL.BreakExec](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.3 CreateBlobStream Method

Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.

Class

[TCustomDADataset](#)

Syntax

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode):  
TStream; override;
```

Parameters

Field

Holds the BLOB field for reading data from or writing data to from a stream.

Mode

Holds the stream mode, for which the stream will be used.

Return Value

The BLOB Stream.

Remarks

Call the `CreateBlobStream` method to obtain a stream for reading data from or writing data to a BLOB field, specified by the `Field` parameter. It must be a `TBlobField` component. You can specify whether the stream will be used for reading, writing, or updating the contents of the field with the `Mode` parameter.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.4 DeleteWhere Method

Removes WHERE clause from the SQL property and assigns the BaseSQL property.

Class

[TCustomDADataset](#)

Syntax

```
procedure Deletewhere;
```

Remarks

Call the `DeleteWhere` method to remove WHERE clause from the the SQL property and assign BaseSQL.

See Also

- [AddWhere](#)
- [BaseSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.5 Execute Method

Executes a SQL statement on the server.

Class

[TCustomDADataset](#)

Overload List

Name	Description
Execute	Executes a SQL statement on the server.
Execute(Iters: integer; Offset: integer)	Used to perform Batch operations .

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Executes a SQL statement on the server.

Class

[TCustomDADataset](#)

Syntax

```
procedure Execute; overload; virtual;
```

Remarks

Call the Execute method to execute an SQL statement on the server. If SQL statement is a SELECT query, Execute calls the Open method.

Execute implicitly prepares SQL statement by calling the [TCustomDADataset.Prepare](#) method if the [TCustomDADataset.Options](#) option is set to True and the statement has not been prepared yet. To speed up the performance in case of multiple Execute calls, an application should call Prepare before calling the Execute method for the first time.

See Also

- [TCustomDADataset.AfterExecute](#)
- [TCustomDADataset.Executing](#)
- [TCustomDADataset.Prepare](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.6 Executing Method

Indicates whether SQL statement is still being executed.

Class

[TCustomDADataset](#)

Syntax

```
function Executing: boolean;
```

Return Value

True, if SQL statement is still being executed.

Remarks

Check Executing to learn whether TCustomDADataset is still executing SQL statement.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.7 Fetched Method

Used to find out whether TCustomDADataset has fetched all rows.

Class

[TCustomDADataset](#)

Syntax

```
function Fetched: boolean; virtual;
```

Return Value

True, if all rows have been fetched.

Remarks

Call the Fetched method to find out whether TCustomDADataset has fetched all rows.

See Also

- [Fetching](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.8 Fetching Method

Used to learn whether TCustomDADataset is still fetching rows.

Class

[TCustomDADataset](#)

Syntax

```
function Fetching: boolean;
```

Return Value

True, if TCustomDADataset is still fetching rows.

Remarks

Check Fetching to learn whether TCustomDADataset is still fetching rows. Use the Fetching method if NonBlocking is True.

See Also

- [Executing](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.9 FetchingAll Method

Used to learn whether TCustomDADataset is fetching all rows to the end.

Class

[TCustomDADataset](#)

Syntax

```
function FetchingAll: boolean;
```

Return Value

True, if TCustomDADataset is fetching all rows to the end.

Remarks

Check FetchingAll to learn whether TCustomDADataset is fetching all rows to the end.

See Also

- [Executing](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.10 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.11 FindMacro Method

Finds a macro with the specified name.

Class

[TCustomDADataset](#)

Syntax

```
function FindMacro(const value: string): TMacro;
```

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the [TMacros.Items](#) property to avoid depending on the order of the items.

See Also

- [TMacro](#)
- [Macros](#)

- [MacroByName](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.12 FindNearest Method

Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.

Class

[TCustomDADataset](#)

Syntax

```
procedure FindNearest(const KeyValues: array of System.TVarRec);
```

Parameters

KeyValues

Holds the values of the record key fields to which the cursor should be moved.

Remarks

Call the FindNearest method to move the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter. If there are no records that match or exceed the specified criteria, the cursor will not move.

This function is provided for BDE compatibility only. It is recommended to use functions [TMemDataSet.Locate](#) and [TMemDataSet.LocateEx](#) for the record search.

See Also

- [TMemDataSet.Locate](#)
- [TMemDataSet.LocateEx](#)
- [FindKey](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.13 FindParam Method

Determines if a parameter with the specified name exists in a dataset.

Class

[TCustomDADataset](#)

Syntax

```
function FindParam(const value: string): TDAParam;
```

Parameters

Value

Holds the name of the param for which to search.

Return Value

the TDAParam object for the specified Name. Otherwise it returns nil.

Remarks

Call the FindParam method to determine if a specified param component exists in a dataset. Name is the name of the param for which to search. If FindParam finds a param with a matching name, it returns a TDAParam object for the specified Name. Otherwise it returns nil.

See Also

- [Params](#)
- [ParamByName](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.14 GetDataTypes Method

Returns internal field types defined in the MemData and accompanying modules.

Class

[TCustomDADataset](#)

Syntax

```
function GetDataTypes(const FieldName: string): integer; virtual;
```

Parameters

FieldName

Holds the name of the field.

Return Value

internal field types defined in MemData and accompanying modules.

Remarks

Call the GetDataType method to return internal field types defined in the MemData and accompanying modules. Internal field data types extend the TFieldType type of VCL by specific database server data types. For example, ftString, ftFile, ftObject.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.15 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.16 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](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.17 GetFieldScale Method

Retrieves the scale of a number field.

Class

[TCustomDADataset](#)

Syntax

```
function GetFieldScale(const FieldName: string): integer;
```

Parameters

FieldName

Holds the existing field name.

Return Value

the scale of the number field.

Remarks

Call the GetFieldScale method to retrieve the scale of a number field. FieldName is the name of an existing field.

See Also

- [GetFieldPrecision](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.18 GetKeyFieldNames Method

Provides a list of available key field names.

Class

[TCustomDADataset](#)

Syntax

```
procedure GetKeyFieldNames(List: TStrings);
```

Parameters

List

The list of available key field names

Return Value

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.19 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.20 GotoCurrent Method

Sets the current record in this dataset similar to the current record in another dataset.

Class

[TCustomDADataset](#)

Syntax

```
procedure GotoCurrent(DataSet: TCustomDADataset);
```

Parameters

DataSet

Holds the TCustomDADataset descendant to synchronize the record position with.

Remarks

Call the GotoCurrent method to set the current record in this dataset similar to the current record in another dataset. The key fields in both these DataSets must be coincident.

See Also

- [TMemDataSet.Locate](#)
- [TMemDataSet.LocateEx](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.21 Lock Method

Locks the current record.

Class

[TCustomDADataset](#)

Syntax

```
procedure Lock; virtual;
```

Remarks

Call the Lock method to lock the current record by executing the statement that is defined in the SQLLock property.

The Lock method sets the savepoint with the name LOCK_ + <component_name>.

See Also

- [UnLock](#)

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.22 MacroByName Method

Finds a macro with the specified name.

Class

[TCustomDADataset](#)

Syntax

```
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the [TMacros.Items](#) property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the [TMacro.Value](#) property.

Example

```
PgQuery.SQL := 'SELECT * FROM Scott.Dept ORDER BY &Order';  
PgQuery.MacroByName('Order').Value := 'DeptNo';  
PgQuery.Open;
```

See Also

- [TMacro](#)
- [Macros](#)

- [FindMacro](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.23 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.1.5.3.24 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

[TCustomDADataset](#)

Syntax

```
procedure Prepare; override;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- [TMemDataSet.Prepared](#)
- [TMemDataSet.UnPrepare](#)
- [Options](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.25 RefreshRecord Method

Actualizes field values for the current record.

Class

[TCustomDADataset](#)

Syntax

```
procedure RefreshRecord;
```

Remarks

Call the RefreshRecord method to actualize field values for the current record.

RefreshRecord performs query to database and refetches new field values from the returned cursor.

See Also

- [RefreshOptions](#)
- [SQLRefresh](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.26 RestoreSQL Method

Restores the SQL property modified by AddWhere and SetOrderBy.

Class

[TCustomDADataset](#)

Syntax

```
procedure RestoreSQL;
```

Remarks

Call the RestoreSQL method to restore the SQL property modified by AddWhere and SetOrderBy.

See Also

- [AddWhere](#)
- [SetOrderBy](#)
- [SaveSQL](#)
- [SQLSaved](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.27 SaveSQL Method

Saves the SQL property value to BaseSQL.

Class

[TCustomDADataset](#)

Syntax

```
procedure SaveSQL;
```

Remarks

Call the SaveSQL method to save the SQL property value to the BaseSQL property.

See Also

- [SQLSaved](#)
- [RestoreSQL](#)
- [BaseSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.28 SetOrderBy Method

Builds an ORDER BY clause of a SELECT statement.

Class

[TCustomDADataset](#)

Syntax

```
procedure SetOrderBy(const Fields: string);
```

Parameters

Fields

Holds the names of the fields which will be added to the ORDER BY clause.

Remarks

Call the SetOrderBy method to build an ORDER BY clause of a SELECT statement. The

fields are identified by the comma-delimited field names.

Note: The `GetOrderBy` and `SetOrderBy` methods serve to process only quite simple queries and don't support, for example, subqueries.

Example

```
Query1.SetOrderBy('DeptNo;DName');
```

See Also

- [GetOrderBy](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.29 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.3.30 UnLock Method

Releases a record lock.

Class

[TCustomDADataset](#)

Syntax

```
procedure UnLock;
```

Remarks

Call the Unlock method to release the record lock made by the [Lock](#) method before.

Unlock is performed by rolling back to the savepoint set by the Lock method.

See Also

- [Lock](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.4 Events

Events of the **TCustomDADataset** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.4.1 AfterExecute Event

Occurs after a component has executed a query to database.

Class

[TCustomDADataset](#)

Syntax

```
property AfterExecute: TAfterExecuteEvent;
```

Remarks

Occurs after a component has executed a query to database.

See Also

- [TCustomDADataset.Execute](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.4.2 AfterFetch Event

Occurs after dataset finishes fetching data from server.

Class

[TCustomDADataset](#)

Syntax

```
property AfterFetch: TAfterFetchEvent;
```

Remarks

The AfterFetch event occurs after dataset finishes fetching data from server.

See Also

- [BeforeFetch](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.4.3 AfterUpdateExecute Event

Occurs after executing insert, delete, update, lock and refresh operations.

Class

[TCustomDADataset](#)

Syntax

```
property AfterUpdateExecute: TUpdateExecuteEvent;
```

Remarks

Occurs after executing insert, delete, update, lock, and refresh operations. You can use AfterUpdateExecute to set the parameters of corresponding statements.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.4.4 BeforeFetch Event

Occurs before dataset is going to fetch block of records from the server.

Class

[TCustomDADataset](#)

Syntax

```
property BeforeFetch: TBeforeFetchEvent;
```

Remarks

The BeforeFetch event occurs every time before dataset is going to fetch a block of records from the server. Set Cancel to True to abort current fetch operation.

See Also

- [AfterFetch](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.5.4.5 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6 TCustomDASQL Class

A base class for components executing SQL statements that do not return result sets.

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

Unit

[DBAccess](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.1 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 the statement that is being executed and the values and types of its 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
BreakExec	Breaks execution of an SQL statement on the server.
Execute	Overloaded. Executes a SQL statement on the server.
Executing	Checks whether TCustomDASQL still

	executes a SQL statement.
FindMacro	Finds a macro with the specified name.
FindParam	Finds a parameter with the specified name.
MacroByName	Finds a macro with the specified 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.

Events

Name	Description
AfterExecute	Occurs after a SQL statement has been executed.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2 Properties

Properties of the **TCustomDASQL** class.

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

Public

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 the statement that is being executed and the values and types of its 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.2 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

[TCustomDASQL](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDACConnection or its descendant class objects.

At runtime, link an instance of a TCustomDACConnection descendant to the Connection property.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.3 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

[TCustomDASQL](#)

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

See Also

- [TCustomDADataset.Debug](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.4 FinalSQL Property

Used to return a SQL statement with expanded macros.

Class

[TCustomDASQL](#)

Syntax

```
property FinalSQL: string;
```

Remarks

Read the FinalSQL property to return a SQL statement with expanded macros. This is the exact statement that will be passed on to the database server.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.5 MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

[TCustomDASQL](#)

Syntax

```
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also

- [Macros](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.6 Macros Property

Makes it possible to change SQL queries easily.

Class

[TCustomDASQL](#)

Syntax

```
property Macros: TMacros stored False;
```

Remarks

With the help of macros you can easily change SQL query text at design- or runtime. Macros 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.7 ParamCheck Property

Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

Class

[TCustomDASQL](#)

Syntax

```
property ParamCheck: boolean default True;
```

Remarks

Use the ParamCheck property to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

Set ParamCheck to True to let TCustomDASQL generate the Params property for the dataset based on a SQL statement automatically.

Setting ParamCheck to False can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of the stored procedures that will accept parameterized values themselves. The default value is True.

See Also

- [Params](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.8 ParamCount Property

Indicates the number of parameters in the Params property.

Class

[TCustomDASQL](#)

Syntax

```
property ParamCount: word;
```

Remarks

Use the ParamCount property to determine how many parameters are there in the Params property.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.9 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.10 ParamValues Property(Indexer)

Used to get or set the values of individual field parameters that are identified by name.

Class

[TCustomDASQL](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.11 Prepared Property

Used to indicate whether a query is prepared for execution.

Class

[TCustomDASQL](#)

Syntax

```
property Prepared: boolean;
```

Remarks

Check the Prepared property to determine if a query is already prepared for execution. True means that the query has already been prepared. As a rule prepared queries are executed faster, but the preparation itself also takes some time. One of the proper cases for using preparation is parametrized queries that are executed several times.

See Also

- [Prepare](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.12 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

[TCustomDASQL](#)

Syntax

```
property RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.2.13 SQL Property

Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

Class

[TCustomDASQL](#)

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called. At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

See Also

- [FinalSQL](#)
- [TCustomDASQL.Execute](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3 Methods

Methods of the **TCustomDASQL** class.

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

Public

Name	Description
BreakExec	Breaks execution of an SQL statement on the server.
Execute	Overloaded. Executes a SQL statement on the server.
Executing	Checks whether TCustomDASQL still executes a SQL statement.
FindMacro	Finds a macro with the specified name.

FindParam	Finds a parameter with the specified name.
MacroByName	Finds a macro with the specified 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.1 BreakExec Method

Breaks execution of an SQL statement on the server.

Class

[TCustomDASQL](#)

Syntax

```
procedure BreakExec;
```

Remarks

Call the BreakExec method to break execution of an SQL statement on the server. It makes sense to call BreakExec only from another thread. Useful when NonBlocking is True.

See Also

- [TCustomDASQL.Execute](#)

- [TCustomDADDataSet.BreakExec](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.2 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 .

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.3 Executing Method

Checks whether TCustomDASQL still executes a SQL statement.

Class

[TCustomDASQL](#)

Syntax

```
function Executing: boolean;
```

Return Value

True, if a SQL statement is still being executed by TCustomDASQL.

Remarks

Check Executing to find out whether TCustomDASQL still executes a SQL statement.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.4 FindMacro Method

Finds a macro with the specified name.

Class

[TCustomDASQL](#)

Syntax

```
function FindMacro(const value: string): TMacro;
```

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the [TMacros.Items](#) property to avoid depending on the order of the items.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.5 FindParam Method

Finds a parameter with the specified name.

Class

[TCustomDASQL](#)

Syntax

```
function FindParam(const value: string): TDAParm;
```

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.6 MacroByName Method

Finds a macro with the specified name.

Class

[TCustomDASQL](#)

Syntax

```
function MacroByName(const value: string): TMacro;
```

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the [TMacros.Items](#) property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the [TMacro.Value](#) property.

See Also

- [TMacro](#)
- [Macros](#)
- [FindMacro](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.7 ParamByName Method

Finds a parameter with the specified name.

Class

[TCustomDASQL](#)

Syntax

```
function ParamByName(const Value: string): TDAParam;
```

Parameters

Value

Holds the name of the parameter to search for.

Return Value

a TDAParam object, if a match was found. Otherwise, an exception is raised.

Remarks

Use the ParamByName method to find a parameter with the specified name. If no parameter with the specified name found, an exception is raised.

Example

```
PgSQL.Execute;  
Edit1.Text := PgSQL.ParamsByName('contact').AsString;
```

See Also

- [FindParam](#)

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.8 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

[TCustomDASQL](#)

Syntax

```
procedure Prepare; virtual;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- [Prepared](#)
- [UnPrepare](#)

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.9 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class

[TCustomDASQL](#)

Syntax

```
procedure UnPrepare; virtual;
```

Remarks

Call the UnPrepare method to free resources allocated for a previously prepared query on the server and client sides.

See Also

- [Prepare](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.3.10 WaitExecuting Method

Waits until TCustomDASQL executes a SQL statement.

Class

[TCustomDASQL](#)

Syntax

```
function waitExecuting(Timeout: integer = 0): boolean;
```

Parameters

Timeout

Holds the time in seconds to wait while TCustomDASQL executes the SQL statement. Zero means infinite time.

Return Value

True, if the execution of a SQL statement was completed in the preset time.

Remarks

Call the `WaitExecuting` method to wait until `TCustomDASQL` executes a SQL statement.

See Also

- [Executing](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.4 Events

Events of the `TCustomDASQL` class.

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

Public

Name	Description
AfterExecute	Occurs after a SQL statement has been executed.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.6.4.1 AfterExecute Event

Occurs after a SQL statement has been executed.

Class

[TCustomDASQL](#)

Syntax

```
property AfterExecute: TAfterExecuteEvent;
```

Remarks

Occurs after a SQL statement has been executed. This event may be used for descendant components which use multithreaded environment.

See Also

- [TCustomDASQL.Execute](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7 TCustomDAUpdateSQL Class

A base class for components that provide DML statements for more flexible control over data modifications.

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

Unit

[DBAccess](#)

Syntax

```
TCustomDAUpdateSQL = class(TComponent);
```

Remarks

TCustomDAUpdateSQL is a base class for components that provide DML statements for more flexible control over data modifications. Besides providing BDE compatibility, this component allows to associate a separate component for each update command.

See Also

- [TCustomPgDataSet.UpdateObject](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2 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
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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.1 DataSet Property

Used to hold a reference to the TCustomDADataset object that is being updated.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property DataSet: TCustomDADataset;
```

Remarks

The DataSet property holds a reference to the TCustomDADataset object that is being updated. Generally it is not used directly.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.1.7.2.2 DeleteObject Property

Provides ability to perform advanced adjustment of the delete operations.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property DeleteObject: TComponent;
```

Remarks

Assign SQL component or a TCustomPgDataSet descendant to this property to perform advanced adjustment of the delete operations. In some cases this can give some additional performance. Use the same principle to set the SQL property of an object as for setting the [DeleteSQL](#) property.

See Also

- [DeleteSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.3 DeleteSQL Property

Used when deleting a record.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property DeleteSQL: TStrings;
```

Remarks

Set the DeleteSQL property to a DELETE statement to use when deleting a record. Statements can be parameterized queries with parameter names corresponding to the

dataset field names.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.4 InsertObject Property

Provides ability to perform advanced adjustment of insert operations.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property InsertObject: TComponent;
```

Remarks

Assign SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of insert operations. In some cases this can give some additional performance. Set the SQL property of the object in the same way as used for the [InsertSQL](#) property.

See Also

- [InsertSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.5 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.6 LockObject Property

Provides ability to perform advanced adjustment of lock operations.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property LockObject: TComponent;
```

Remarks

Assign a SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of lock operations. In some cases that can give some additional performance. Set the SQL property of an object in the same way as used for the [LockSQL](#) property.

See Also

- [LockSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.7 LockSQL Property

Used to lock the current record.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property LockSQL: TStrings;
```

Remarks

Use the LockSQL property to lock the current record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.8 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.9 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.10 RefreshObject Property

Provides ability to perform advanced adjustment of refresh operations.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property RefreshObject: TComponent;
```

Remarks

Assign a SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of refresh operations. In some cases that can give some additional performance. Set the SQL property of the object in the same way as used for the [RefreshSQL](#) property.

See Also

- [RefreshSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.11 RefreshSQL Property

Used to specify an SQL statement that will be used for refreshing the current record by [TCustomDADataset.RefreshRecord](#) procedure.

Class

[TCustomDAUpdatesSQL](#)

Syntax

```
property RefreshSQL: TStrings;
```

Remarks

Use the RefreshSQL property to specify a SQL statement that will be used for refreshing the current record by the [TCustomDADataset.RefreshRecord](#) procedure.

You can assign to SQLRefresh a WHERE clause only. In such a case it is added to SELECT defined by the SQL property by [TCustomDADataset.AddWhere](#).

To create a RefreshSQL statement at design time, use the query statements editor.

See Also

- [TCustomDADataset.RefreshRecord](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.2.12 SQL Property(Indexer)

Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Class

[TCustomDAUpdatesSQL](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.3 Methods

Methods of the **TCustomDAUpdateSQL** class.

For a complete list of the **TCustomDAUpdateSQL** class members, see the

[TCustomDAUpdateSQL Members](#) topic.

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.3.1 Apply Method

Sets parameters for a SQL statement and executes it to update a record.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
procedure Apply(UpdateKind: TUpdateKind); virtual;
```

Parameters

UpdateKind

Specifies which of update SQL statements to execute.

Remarks

Call the Apply method to set parameters for a SQL statement and execute it to update a record. UpdateKind indicates which SQL statement to bind and execute.

Apply is primarily intended for manually executing update statements from an OnUpdateRecord event handler.

Note: If a SQL statement does not contain parameters, it is more efficient to call ExecSQL instead of Apply.

See Also

- [ExecSQL](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.7.3.2 ExecSQL Method

Executes a SQL statement.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
procedure ExecSQL(UpdateKind: TUpdateKind);
```

Parameters

UpdateKind

Specifies the kind of update statement to be executed.

Remarks

Call the ExecSQL method to execute a SQL statement, necessary for updating the records belonging to a read-only result set when cached updates is enabled. UpdateKind specifies the statement to execute.

ExecSQL is primarily intended for manually executing update statements from the

OnUpdateRecord event handler.

Note: To both bind parameters and execute a statement, call [Apply](#).

See Also

- [Apply](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.8 TDACCondition Class

Represents a condition from the [TDACConditions](#) list.

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

Unit

[DBAccess](#)

Syntax

```
TDACCondition = class(TCollectionItem);
```

Remarks

Manipulate conditions using [TDACConditions](#).

See Also

- [TDACConditions](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.8.1 Members

[TDACCondition](#) 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.8.2 Properties

Properties of the **TDACCondition** class.

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

Published

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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.8.2.1 Enabled Property

Indicates whether the condition is enabled or not

Class

[TDACondition](#)

Syntax

```
property Enabled: Boolean default True;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.8.2.2 Name Property

The name of the condition

Class

[TDACondition](#)

Syntax

```
property Name: string;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.8.2.3 Value Property

The value of the condition

Class

[TDACondition](#)

Syntax

```
property value: string;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.8.3 Methods

Methods of the **TDACCondition** class.

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

Public

Name	Description
Disable	Disables the condition
Enable	Enables the condition

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.8.3.1 Disable Method

Disables the condition

Class

[TDACCondition](#)

Syntax

```
procedure Disable;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.8.3.2 Enable Method

Enables the condition

Class

[TDACondition](#)

Syntax

```
procedure Enable;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9 TDAConditions Class

Holds a collection of [TDACondition](#) objects.

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

Unit

[DBAccess](#)

Syntax

```
TDAConditions = class(TCollection);
```

Remarks

The given example code

```
UniTable1.Conditions.Add('1', 'JOB="MANAGER"');  
UniTable1.Conditions.Add('2', 'SAL>2500');  
UniTable1.Conditions.Enable;  
UniTable1.Open;
```

will return the following SQL:

```
SELECT * FROM EMP  
WHERE (JOB="MANAGER")  
and  
(SAL<2500)
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.1 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

© 1997-2024

Devert. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.11.1.9.2 Properties

Properties of the **TDAConditions** class.

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

Public

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](#)

© 1997-2024

Devert. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.11.1.9.2.1 Condition Property(Indexer)

Used to iterate through all the conditions.

Class

[TDAConditions](#)

Syntax

```
property Condition[Index: Integer]: TDACondition;
```

Parameters

Index

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.2.2 Enabled Property

Indicates whether the condition is enabled

Class

[TDAConditions](#)

Syntax

```
property Enabled: Boolean;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.2.3 Items Property(Indexer)

Used to iterate through all conditions.

Class

[TDAConditions](#)

Syntax

```
property Items[Index: Integer]: TDACondition; default;
```

Parameters

Index

Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all conditions. Index identifies the index in the range

0..Count - 1. Items can reference a particular condition by its index, but the [Condition](#) property is preferred in order to avoid depending on the order of the conditions.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.2.4 Text Property

The property returns condition names and values as `CONDITION_NAME=CONDITION`

Class

[TDAConditions](#)

Syntax

```
property Text: string;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.2.5 WhereSQL Property

Returns the SQL WHERE condition added in the Conditions property.

Class

[TDAConditions](#)

Syntax

```
property whereSQL: string;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.3 Methods

Methods of the **TDAConditions** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.3.1 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: string; Enabled: Boolean)	Adds a condition to the WHERE clause of the query.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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)
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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)
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.3.2 Delete Method

Deletes the condition

Class

[TDAConditions](#)

Syntax

```
procedure Delete(Index: integer);
```

Parameters

Index

Index of the condition

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.3.3 Disable Method

Disables the condition

Class

[TDAConditions](#)

Syntax

```
procedure Disable;
```

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.3.4 Enable Method

Enables the condition

Class

[TDAConditions](#)

Syntax

```
procedure Enable;
```

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.3.5 Find Method

Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.

Class

[TDAConditions](#)

Syntax

```
function Find(const Name: string): TDACondition;
```

Parameters

Name

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.3.6 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

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.3.7 IndexOf Method

Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.

Class

[TDAConditions](#)

Syntax

```
function IndexOf(const Name: string): Integer;
```

Parameters

Name

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.9.3.8 Remove Method

Removes the condition

Class

[TDAConditions](#)

Syntax

```
procedure Remove(const Name: string);
```

Parameters

Name

Specifies the name of the removed condition

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.10 TDACConnectionOptions Class

This class allows setting up the behaviour of the TDACConnection class.

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

Unit

[DBAccess](#)

Syntax

```
TDACConnectionOptions = class(TPersistent);
```

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.10.1 Members

[TDACConnectionOptions](#) 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.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.10.2 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 TCustomDAConnection.OnConnectionLost 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.10.2.1 Allow ImplicitConnect 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.10.2.2 DefaultSortType Property

Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the [TMemDataSet.IndexFieldNames](#) property of a dataset.

Class

[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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.10.2.3 DisconnectedMode Property

Used to open a connection only when needed for performing a server call and closes after performing the operation.

Class

[TDACConnectionOptions](#)

Syntax

```
property DisconnectedMode: boolean default False;
```

Remarks

If True, connection opens only when needed for performing a server call and closes after performing the operation. Datasets remain opened when connection closes. May be useful to save server resources and operate in unstable or expensive network. Drawback of using disconnect mode is that each connection establishing requires some time for authorization. If connection is often closed and opened it can slow down the application work. See the [Disconnected Mode](#) topic for more information.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.10.2.4 KeepDesignConnected Property

Used to prevent an application from establishing a connection at the time of startup.

Class

[TDACConnectionOptions](#)

Syntax

```
property KeepDesignConnected: boolean default True;
```

Remarks

At the time of startup prevents application from establishing a connection even if the Connected property was set to True at design-time. Set KeepDesignConnected to False to initialize the connected property to False, even if it was True at design-time.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.11.1.10.2.5 LocalFailover Property

If True, the [TCustomDACConnection.OnConnectionLost](#) event occurs and a failover operation can be performed after connection breaks.

Class

[TDACconnectionOptions](#)

Syntax

```
property LocalFailover: boolean default False;
```

Remarks

If True, the [TCustomDACConnection.OnConnectionLost](#) event occurs and a failover operation can be performed after connection breaks. Read the [Working in an Unstable Network](#) topic for more information about using failover.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.11 TDACconnectionSSLOptions Class

This class is used to set up the SSL options.

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

Unit

[DBAccess](#)

Syntax

```
TDACconnectionSSLOptions = class(TPersistent);
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.11.1 Members

[TDACconnectionSSLOptions](#) class overview.

Properties

Name	Description
CACert	Holds the path to the certificate authority file.
Cert	Holds the path to the client certificate.
CipherList	Holds the list of allowed SSL ciphers.
Key	Holds the path to the private client key.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.11.2 Properties

Properties of the **TDACConnectionSSLOptions** class.

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

Published

Name	Description
CACert	Holds the path to the certificate authority file.
Cert	Holds the path to the client certificate.
CipherList	Holds the list of allowed SSL ciphers.
Key	Holds the path to the private client key.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.11.2.1 CACert Property

Holds the path to the certificate authority file.

Class

[TDAConnectionSSLOptions](#)

Syntax

```
property CACert: string;
```

Remarks

Use the CACert property to specify the path to the certificate authority file.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.11.2.2 Cert Property

Holds the path to the client certificate.

Class

[TDAConnectionSSLOptions](#)

Syntax

```
property Cert: string;
```

Remarks

Use the Cert property to specify the path to the client certificate.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.11.2.3 CipherList Property

Holds the list of allowed SSL ciphers.

Class

[TDAConnectionSSLOptions](#)

Syntax

```
property CipherList: string;
```

Remarks

Use the CipherList property to specify the list of allowed SSL ciphers.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.11.2.4 Key Property

Holds the path to the private client key.

Class

[TDAConnectionSSLOptions](#)

Syntax

```
property Key: string;
```

Remarks

Use the Key property to specify the path to the private client key.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12 TDADatasetOptions Class

This class allows setting up the behaviour of the TDADataset class.

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

Unit

[DBAccess](#)

Syntax

```
TDADatasetOptions = class(TPersistent);
```

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.11.1.12.1 Members

[TDADatasetOptions](#) class overview.

Properties

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.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2 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.
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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.2 CacheCalcFields Property

Used to enable caching of the TField.Calculated and TField.Lookup fields.

Class

[TDADatasetOptions](#)

Syntax

```
property CacheCalcFields: boolean default false;
```

Remarks

Use the CacheCalcFields property to enable caching of the TField.Calculated and TField.Lookup fields. It can be useful for reducing CPU usage for calculated fields. Using caching of calculated and lookup fields increases memory usage on the client side.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.3 CompressBlobMode Property

Used to store values of the BLOB fields in compressed form.

Class

[TDADatasetOptions](#)

Syntax

```
property CompressBlobMode: TCompressBlobMode default cbNone;
```

Remarks

Use the CompressBlobMode property to store values of the BLOB fields in compressed form. Add the MemData unit to uses list to use this option. Compression rate greatly depends on stored data, for example, usually graphic data compresses badly unlike text.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.4 DefaultValue Property

Used to request default values/expressions from the server and assign them to the DefaultValue property.

Class

[TDADatasetOptions](#)

Syntax

```
property DefaultValue: boolean default False;
```

Remarks

If True, the default values/expressions are requested from the server and assigned to the DefaultValue property of TField objects replacing already existent values.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.5 DetailDelay Property

Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.

Class

[TDADatasetOptions](#)

Syntax

```
property DetailDelay: integer default 0;
```

Remarks

Use the DetailDelay property to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset. If DetailDelay is 0 (the default value) then refreshing of detail dataset occurs immediately. The DetailDelay option should be used for detail dataset.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.6 FieldsOrigin Property

Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.

Class

[TDADatasetOptions](#)

Syntax

```
property Fieldsorigin: boolean;
```

Remarks

If True, TCustomDADataset fills the Origin property of the TField objects by appropriate value when opening a dataset.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.7 FlatBuffers Property

Used to control how a dataset treats data of the ftString and ftVarBytes fields.

Class

[TDADatasetOptions](#)

Syntax

```
property FlatBuffers: boolean default False;
```

Remarks

Use the FlatBuffers property to control how a dataset treats data of the ftString and ftVarBytes fields. When set to True, all data fetched from the server is stored in record pdata without unused tails.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.8 InsertAllSetFields Property

Used to include all set dataset fields in the generated INSERT statement

Class

[TDADatasetOptions](#)

Syntax

```
property InsertAllSetFields: boolean default False;
```

Remarks

If True, all set dataset fields, including those set to NULL explicitly, will be included in the generated INSERT statements. Otherwise, only set fields containing not NULL values will be included to the generated INSERT statement.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.9 LocalMasterDetail Property

Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.

Class

[TDADatasetOptions](#)

Syntax

```
property LocalMasterDetail: boolean default False;
```

Remarks

If True, for detail dataset in master-detail relationship TCustomDADataset uses local filtering for establishing master/detail relationship and does not refer to the server. Otherwise detail dataset performs query each time a record is selected in master dataset. This option is useful for reducing server calls number, server resources economy. It can be useful for slow connection. The [TMemDataSet.CachedUpdates](#) mode can be used for detail dataset only when this option is set to true. Setting the LocalMasterDetail option to True is not recommended when detail table contains too many rows, because when it is set to False,

only records that correspond to the current record in master dataset are fetched.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.10 LongStrings Property

Used to represent string fields with the length that is greater than 255 as TStringField.

Class

[TDADatasetOptions](#)

Syntax

```
property LongStrings: boolean default True;
```

Remarks

Use the LongStrings property to represent string fields with the length that is greater than 255 as TStringField, not as TMemofield.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.11 MasterFieldsNullable Property

Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).

Class

[TDADatasetOptions](#)

Syntax

```
property MasterFieldsNullable: boolean default False;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.12 NumberRange Property

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Class

[TDADatasetOptions](#)

Syntax

```
property NumberRange: boolean default False;
```

Remarks

Use the NumberRange property to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.13 QueryRecCount Property

Used for TCustomDADataset to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.

Class

[TDADatasetOptions](#)

Syntax

```
property QueryRecCount: boolean default False;
```

Remarks

If True, and the FetchAll property is False, TCustomDADataset performs additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records. Does not have any effect if the FetchAll property is True.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.14 QuoteNames Property

Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.

Class

[TDADatasetOptions](#)

Syntax

```
property QuoteNames: boolean default False;
```

Remarks

If True, TCustomDADataset quotes all database object names in autogenerated SQL statements such as update SQL.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.15 RemoveOnRefresh Property

Used for a dataset to locally remove a record that can not be found on the server.

Class

[TDADatasetOptions](#)

Syntax

```
property RemoveOnRefresh: boolean default True;
```

Remarks

When the RefreshRecord procedure can't find necessary record on the server and RemoveOnRefresh is set to True, dataset removes the record locally. Usually RefreshRecord can't find necessary record when someone else dropped the record or changed the key value of it.

This option makes sense only if the StrictUpdate option is set to False. If the StrictUpdate option is True, error will be generated regardless of the RemoveOnRefresh option value.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.11.1.12.2.16 RequiredFields Property

Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.

Class

[TDADatasetOptions](#)

Syntax

```
property RequiredFields: boolean default True;
```

Remarks

If True, TCustomDADataset sets the Required property of the TField objects for the NOT NULL fields. It is useful when table has a trigger which updates the NOT NULL fields.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.17 ReturnParams Property

Used to return the new value of fields to dataset after insert or update.

Class

[TDADatasetOptions](#)

Syntax

```
property ReturnParams: boolean default False;
```

Remarks

Use the ReturnParams property to return the new value of fields to dataset after insert or update. The actual value of field after insert or update may be different from the value stored in the local memory if the table has a trigger. When ReturnParams is True, OUT parameters of the SQLInsert and SQLUpdate statements is assigned to the corresponding fields.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.1.12.2.18 SetFieldsReadOnly Property

Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.

Class

[TDADatasetOptions](#)

Syntax

```
property SetFieldsReadOnly: boolean default True;
```

Remarks

If True, dataset sets the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated. Set this option for datasets that use automatic generation of the update SQL statements only.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.19 StrictUpdate Property

Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.

Class

[TDADatasetOptions](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.20 TrimFixedChar Property

Specifies whether to discard all trailing spaces in the string fields of a dataset.

Class

[TDADatasetOptions](#)

Syntax

```
property TrimFixedChar: boolean default True;
```

Remarks

Specifies whether to discard all trailing spaces in the string fields of a dataset.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.21 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.12.2.22 UpdateBatchSize Property

Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Class

[TDADatasetOptions](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.13 TDAEncryption Class

Used to specify the options of the data encryption in a dataset.

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

Unit

[DBAccess](#)

Syntax

```
TDAEncryption = class(TPersistent);
```

Remarks

Set the properties of Encryption to specify the options of the data encryption in a dataset.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.13.1 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.13.2 Properties

Properties of the **TDAEncryption** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.13.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.13.2.2 Fields Property

Used to set field names for which encryption will be performed.

Class

[TDAEncryption](#)

Syntax

```
property Fields: string;
```

Remarks

Used to set field names for which encryption will be performed. Field names must be

separated by semicolons.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14 TDAMapRule Class

Class that forms rules for Data Type Mapping.

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

Unit

[DBAccess](#)

Syntax

```
TDAMapRule = class(TMapRule);
```

Remarks

Using properties of this class, it is possible to change parameter values of the specified rules from the TDAMapRules set.

Inheritance Hierarchy

TMapRule

TDAMapRule

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2 Properties

Properties of the **TDAMapRule** class.

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2.2 DBLengthMin Property

Minimum DB field length, starting from which the rule is applied.

Class

[TDAMapRule](#)

Syntax

```
property DBLengthMin: Integer default r1Any;
```

Remarks

Setting minimum DB field length, starting from which the rule is applied to the specified DB field.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2.3 DBScaleMax Property

Maximum DB field scale, until which the rule is applied to the specified DB field.

Class

[TDAMapRule](#)

Syntax

```
property DBScaleMax: Integer default r1Any;
```

Remarks

Setting maximum DB field scale, until which the rule is applied to the specified DB field.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.1.14.2.4 DBScaleMin Property

Minimum DB field Scale, starting from which the rule is applied to the specified DB field.

Class

[TDAMapRule](#)

Syntax

```
property DBScaleMin: Integer default r1Any;
```

Remarks

Setting minimum DB field Scale, starting from which the rule is applied to the specified DB field.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2.5 DBType Property

DB field type, that the rule is applied to.

Class

[TDAMapRule](#)

Syntax

```
property DBType: word default dtUnknown;
```

Remarks

Setting DB field type, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields of the specified type in all DataSets related to this Connection.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2.6 FieldLength Property

The resultant field length in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldLength: Integer default r1Any;
```

Remarks

Setting the Delphi field length after conversion.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2.7 FieldName Property

DataSet field name, for which the rule is applied.

Class

[TDAMapRule](#)

Syntax

```
property FieldName: string;
```

Remarks

Specifies the DataSet field name, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields with such name in DataSets related to this Connection.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2.8 FieldScale Property

The resultant field Scale in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldScale: Integer default r1Any;
```

Remarks

Setting the Delphi field Scale after conversion.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2.9 FieldType Property

Delphi field type, that the specified DB type or DataSet field will be mapped to.

Class

[TDAMapRule](#)

Syntax

```
property FieldType: TFieldType stored IsFieldTypeStored default  
ftUnknown;
```

Remarks

Setting Delphi field type, that the specified DB type or DataSet field will be mapped to.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.14.2.10 IgnoreErrors Property

Ignoring errors when converting data from DB to Delphi type.

Class

[TDAMapRule](#)

Syntax

```
property IgnoreErrors: Boolean default False;
```

Remarks

Allows to ignore errors while data conversion in case if data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.15 TDAMapRules Class

Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.

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

Unit

[DBAccess](#)

Syntax

```
TDAMapRules = class(TMapRules);
```

Inheritance Hierarchy

TMapRules

TDAMapRules

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.15.1 Members

[TDAMapRules](#) class overview.

Properties

Name	Description
IgnoreInvalidRules	Used to avoid raising exception on mapping rules that can't be applied.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.15.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.15.2.1 IgnoreInvalidRules Property

Used to avoid raising exception on mapping rules that can't be applied.

Class

[TDAMapRules](#)

Syntax

```
property IgnoreInvalidRules: boolean default False;
```

Remarks

Allows to ignore errors (not to raise exception) during data conversion in case if the data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

Note: In order to ignore errors occurring during data conversion, use the [TDAMapRule.IgnoreErrors](#) property

See Also

- [TDAMapRule.IgnoreErrors](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.16 TDAMetaData Class

A class for retrieving metainformation of the specified database objects in the form of dataset.

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

Unit

[DBAccess](#)

Syntax

```
TDAMetaData = class (TMemDataSet);
```

Remarks

TDAMetaData is a TDataSet descendant standing for retrieving metainformation of the specified database objects in the form of dataset. First of all you need to specify which kind of metainformation you want to see. For this you need to assign the [TDAMetaData.MetaDataKind](#) property. Provide one or more conditions in the [TDAMetaData.Restrictions](#) property to diminish the size of the resultset and get only information you are interested in.

Use the [TDAMetaData.GetMetaDataKinds](#) method to get the full list of supported kinds of meta data. With the [TDAMetaData.GetRestrictions](#) method you can find out what restrictions are applicable to the specified MetaDataKind.

Example

The code below demonstrates how to get information about columns of the 'emp' table:

```

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.16.1 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 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.

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.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.16.2 Properties

Properties of the **TDAMetaData** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.16.2.1 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

[TDAMetaData](#)

Syntax

```
property Connection: TCustomDACConnection;
```

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 TCustomDACConnection or its descendant class objects.

At runtime, set the Connection property to reference an instantiated TCustomDACConnection object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.16.2.2 MetaDataKind Property

Used to specify which kind of metainformation to show.

Class

[TDAMetaData](#)

Syntax

```
property MetaDataKind: string;
```

Remarks

This string property specifies which kind of metainformation to show. The value of this property should be assigned before activating the component. If `MetaDataKind` equals to an empty string (the default value), the full value list that this property accepts will be shown.

They are described in the table below:

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 <code>MetadadataKind</code> 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 <code>MetaDataKind</code>
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.16.2.3 Restrictions Property

Used to provide one or more conditions restricting the list of objects to be described.

Class

[TDAMetaData](#)

Syntax

```
property Restrictions: TStrings;
```

Remarks

Use the Restriction list to provide one or more conditions restricting the list of objects to be described. To see the full list of restrictions and to which metadata kinds they are applicable, you should assign the Restrictions value to the MetaDataKind property and view the result.

See Also

- [MetaDataKind](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.16.3 Methods

Methods of the **TDAMetaData** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.16.3.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.16.3.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17 TDAParam Class

A class that forms objects to represent the values of the [parameters set](#).

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

Unit

[DBAccess](#)

Syntax

```
TDAParam = class(TParam);
```

Remarks

Use the properties of TDAParam to set the value of a parameter. Objects that use parameters create TDAParam objects to represent these parameters. For example, TDAParam objects are used by TCustomDASQL, TCustomDADataset.

TDAParam shares many properties with TField, as both describe the value of a field in a dataset. However, a TField object has several properties to describe the field binding and the way the field is displayed, edited, or calculated, that are not needed in a TDAParam object. Conversely, TDAParam includes properties that indicate how the field value is passed as a parameter.

See Also

- [TCustomDADataset](#)
- [TCustomDASQL](#)
- [TDAParams](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.1 AsBlob Property

Used to set and read the value of the BLOB parameter as string.

Class

[TDAParam](#)

Syntax

```
property AsBlob: TBlobData;
```

Remarks

Use the AsBlob property to set and read the value of the BLOB parameter as string. Setting AsBlob will set the DataType property to ftBlob.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.2 AsBlobRef Property

Used to set and read the value of the BLOB parameter as a TBlob object.

Class

[TDAParam](#)

Syntax

```
property AsBlobRef: TBlob;
```

Remarks

Use the AsBlobRef property to set and read the value of the BLOB parameter as a TBlob object. Setting AsBlobRef will set the DataType property to ftBlob.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.3 AsFloat Property

Used to assign the value for a float field to a parameter.

Class

[TDAParam](#)

Syntax

```
property AsFloat: double;
```

Remarks

Use the AsFloat property to assign the value for a float field to the parameter. Setting AsFloat will set the DataType property to dtFloat.

Read the AsFloat property to determine the value that was assigned to an output parameter, represented as Double. The value of the parameter will be converted to the Double value if possible.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.4 AsInteger Property

Used to assign the value for an integer field to the parameter.

Class

[TDAParam](#)

Syntax

```
property AsInteger: LongInt;
```

Remarks

Use the AsInteger property to assign the value for an integer field to the parameter. Setting AsInteger will set the DataType property to dtInteger.

Read the AsInteger property to determine the value that was assigned to an output parameter, represented as a 32-bit integer. The value of the parameter will be converted to the Integer value if possible.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.5 AsLargeInt Property

Used to assign the value for a LargeInteger field to the parameter.

Class

[TDAParam](#)

Syntax

```
property AsLargeInt: Int64;
```

Remarks

Set the AsLargeInt property to assign the value for an Int64 field to the parameter. Setting AsLargeInt will set the DataType property to dtLargeint.

Read the AsLargeInt property to determine the value that was assigned to an output parameter, represented as a 64-bit integer. The value of the parameter will be converted to the Int64 value if possible.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.6 AsMemo Property

Used to assign the value for a memo field to the parameter.

Class

[TDAParam](#)

Syntax

```
property AsMemo: string;
```

Remarks

Use the AsMemo property to assign the value for a memo field to the parameter. Setting AsMemo will set the DataType property to ftMemo.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.7 AsMemoRef Property

Used to set and read the value of the memo parameter as a TBlob object.

Class

[TDAParam](#)

Syntax

```
property AsMemoRef: TBlob;
```

Remarks

Use the AsMemoRef property to set and read the value of the memo parameter as a TBlob object. Setting AsMemoRef will set the DataType property to ftMemo.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.8 AsSQLTimeStamp Property

Used to specify the value of the parameter when it represents a SQL timestamp field.

Class

[TDAParam](#)

Syntax

```
property AsSQLTimeStamp: TSQLTimeStamp;
```

Remarks

Set the AsSQLTimeStamp property to assign the value for a SQL timestamp field to the parameter. Setting AsSQLTimeStamp sets the DataType property to ftTimeStamp.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.9 AsString Property

Used to assign the string value to the parameter.

Class

[TDAParam](#)

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to assign the string value to the parameter. Setting AsString will set the DataType property to ftString.

Read the AsString property to determine the value that was assigned to an output parameter represented as a string. The value of the parameter will be converted to a string.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.10 AsWideString Property

Used to assign the Unicode string value to the parameter.

Class

[TDAParam](#)

Syntax

```
property AswideString: string;
```

Remarks

Set AsWideString to assign the Unicode string value to the parameter. Setting AsWideString will set the DataType property to ftWideString.

Read the AsWideString property to determine the value that was assigned to an output parameter, represented as a Unicode string. The value of the parameter will be converted to a Unicode string.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.11.1.17.2.11 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.12 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.13 ParamType Property

Used to indicate the type of use for a parameter.

Class

[TDAParam](#)

Syntax

```
property ParamType default DB . ptUnknown;
```

Remarks

Objects that use TDAParam objects to represent field parameters set ParamType to indicate the type of use for a parameter.

To learn the description of TParamType refer to Delphi Help.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.14 Size Property

Specifies the size of a string type parameter.

Class

[TDAParam](#)

Syntax

```
property Size: integer default 0;
```

Remarks

Use the Size property to indicate the maximum number of characters the parameter may contain. Use the Size property only for Output parameters of the **ftString**, **ftFixedChar**, **ftBytes**, **ftVarBytes**, or **ftWideString** type.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.2.15 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.3 Methods

Methods of the **TDAParam** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.3.1 AssignField Method

Assigns field name and field value to a param.

Class

[TDAParam](#)

Syntax

```
procedure AssignField(Field: TField);
```

Parameters

Field

Holds the field which name and value should be assigned to the param.

Remarks

Call the AssignField method to assign field name and field value to a param.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.3.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.3.3 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](#)

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.3.4 LoadFromStream Method

Places the content from a stream into a TDAParam object.

Class

[TDAParam](#)

Syntax

```
procedure LoadFromStream(Stream: TStream; BlobType: TBlobType);  
virtual;
```

Parameters

Stream

Holds the stream to copy content from.

BlobType

Holds a value that modifies the DataType property so that this TDAParam object now holds the BLOB value.

Remarks

Call the LoadFromStream method to place the content from a stream into a TDAParam object. The BlobType value modifies the DataType property so that this TDAParam object now holds the BLOB value.

See Also

- [LoadFromFile](#)

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.17.3.5 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.18 TDAParams Class

This class is used to manage a list of TDAParam objects for an object that uses field parameters.

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

Unit

[DBAccess](#)

Syntax

```
TDAParams = class(TParams);
```

Remarks

Use TDAParams to manage a list of TDAParam objects for an object that uses field parameters. For example, TCustomDADataset objects and TCustomDASQL objects use TDAParams objects to create and access their parameters.

See Also

- [TCustomDADataset.Params](#)
- [TCustomDASQL.Params](#)
- [TDAParam](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.18.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.18.2 Properties

Properties of the **TDAParams** class.

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

Public

Name	Description
Items	Used to iterate through all parameters.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.18.2.1 Items Property(Indexer)

Used to iterate through all parameters.

Class

[TDAParams](#)

Syntax

```
property Items[Index: integer]: TDAParam; default;
```

Parameters

Index

Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all parameters. Index identifies the index in the range 0..Count - 1. Items can reference a particular parameter by its index, but the ParamByName method is preferred in order to avoid depending on the order of the parameters.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.18.3 Methods

Methods of the **TDAParams** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.18.3.1 FindParam Method

Searches for a parameter with the specified name.

Class

[TDAParams](#)

Syntax

```
function FindParam(const value: string): TDAParam;
```

Parameters

Value

Holds the parameter name.

Return Value

a parameter, if a match was found. Nil otherwise.

Remarks

Use the FindParam method to find a parameter with the name passed in Value. If a match is found, FindParam returns the parameter. Otherwise, it returns nil. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate more than one parameter at a time by name, use the GetParamList method instead. To get only the value of a named parameter, use the ParamValues property.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.18.3.2 ParamByName Method

Searches for a parameter with the specified name.

Class

[TDAParams](#)

Syntax

```
function ParamByName(const value: string): TDAParam;
```

Parameters

Value

Holds the parameter name.

Return Value

a parameter, if the match was found. otherwise an exception is raised.

Remarks

Use the ParamByName method to find a parameter with the name passed in Value. If a match was found, ParamByName returns the parameter. Otherwise, an exception is raised. Use this method rather than a direct reference to the [Items](#) property to avoid depending on the order of the entries.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindParam method.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19 TDATransaction Class

A base class that implements functionality for controlling transactions.

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

Unit

[DBAccess](#)

Syntax

```
TDATransaction = class(TComponent);
```

Remarks

TDATransaction is a base class for components implementing functionality for managing transactions.

Do not create instances of TDATransaction. Use descendants of the TDATransaction class instead.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.1 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.
-------------------------------------	---

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.2.2 DefaultCloseAction Property

Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Class

[TDATransaction](#)

Syntax

```
property DefaultCloseAction: TCRTransactionAction default  
taRollback;
```

Remarks

Use DefaultCloseAction to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.3 Methods

Methods of the **TDATransaction** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.3.1 Commit Method

Commits the current transaction.

Class

[TDATransaction](#)

Syntax

```
procedure Commit; virtual;
```

Remarks

Call the Commit method to commit the current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database, and then finishes the transaction.

See Also

- [Rollback](#)
- [StartTransaction](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.3.2 Rollback Method

Discards all modifications of data associated with the current transaction and ends the transaction.

Class

[TDATransaction](#)

Syntax

```
procedure Rollback; virtual;
```

Remarks

Call Rollback to cancel all data modifications made within the current transaction to the database server, and finish the transaction.

See Also

- [Commit](#)
- [StartTransaction](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.3.3 StartTransaction Method

Begins a new transaction.

Class

[TDATransaction](#)

Syntax

```
procedure StartTransaction; virtual;
```

Remarks

Call the StartTransaction method to begin a new transaction against the database server. Before calling StartTransaction, an application should check the [Active](#) property. If TDATransaction.Active is True, indicating that a transaction is already in progress, a

subsequent call to `StartTransaction` will raise `EDatabaseError`. An active transaction must be finished by call to [Commit](#) or [Rollback](#) before call to `StartTransaction`. Call to `StartTransaction` when connection is closed also will raise `EDatabaseError`.

Updates, insertions, and deletions that take place after a call to `StartTransaction` are held by the server until the application calls [Commit](#) to save the changes, or [Rollback](#) to cancel them.

See Also

- [Commit](#)
- [Rollback](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.4 Events

Events of the **TDATransaction** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.4.1 OnCommit Event

Occurs after the transaction has been successfully committed.

Class

[TDATransaction](#)

Syntax

```
property OnCommit: TNotifyEvent;
```

Remarks

The OnCommit event fires when the M:Devart.Dac.TDATransaction.Commit method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.PgDac.TPgTransaction.CommitRetaining() method execution, the [OnCommitRetaining](#) event is used. When an error occurs during commit, the [OnError](#) event fires.

See Also

- [Commit](#)
- [OnError](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.4.2 OnCommitRetaining Event

Occurs after CommitRetaining has been executed.

Class

[TDATransaction](#)

Syntax

```
property OnCommitRetaining: TNotifyEvent;
```

Remarks

The OnCommitRetaining event fires when the CommitRetaining method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.Dac.TDATransaction.Commit method execution, the [OnCommit](#) event is used.

When an error occurs during commit, the [OnError](#) event fired.

See Also

- [Commit](#)
- [OnCommit](#)
- [OnError](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.4.3 OnError Event

Used to process errors that occur during executing a transaction.

Class

[TDATransaction](#)

Syntax

```
property OnError: TDATransactionErrorEvent;
```

Remarks

Add a handler to the OnError event to process errors that occur during executing a transaction control statements such as [Commit](#), [Rollback](#). Check the E parameter to get the error code.

See Also

- [Commit](#)

- [Rollback](#)
- [StartTransaction](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.4.4 OnRollback Event

Occurs after the transaction has been successfully rolled back.

Class

[TDATransaction](#)

Syntax

```
property OnRollback: TNotifyEvent;
```

Remarks

The OnRollback event fires when the M:Devart.Dac.TDATransaction.Rollback method is executed, just after the transaction is successfully rolled back. In order to respond to the M:Devart.PgDac.TPgTransaction.RollbackRetaining() method execution, the [OnRollbackRetaining](#) event is used.

When an error occurs during rollback, the [OnError](#) event fired.

See Also

- [Rollback](#)
- [OnError](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.19.4.5 OnRollbackRetaining Event

Occurs after RollbackRetaining has been executed.

Class

[TDATransaction](#)

Syntax

```
property OnRollbackRetaining: TNotifyEvent;
```

Remarks

The OnRollbackRetaining event fires when the RollbackRetaining method is executed, just after the transaction is successfully rolled back. In order to respond to the [Rollback](#) method execution, the [OnRollback](#) event is used. When an error occurs during rollback, the [OnError](#) event fired.

See Also

- [Rollback](#)
- [OnRollback](#)
- [OnError](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20 TMacro Class

Object that represents the value of a macro.

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

Unit

[DBAccess](#)

Syntax

```
TMacro = class(TCollectionItem);
```

Remarks

TMacro object represents the value of a macro. Macro is a variable that holds string value. You just insert **&** MacroName in a SQL query text and change the value of macro by the Macro property editor at design time or the Value property at run time. At the time of opening query macro is replaced by its value.

If by any reason it is not convenient for you to use the ' **&** ' symbol as a character of macro

replacement, change the value of the MacroChar variable.

See Also

- [TMacros](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20.2.1 Active Property

Used to determine if the macro should be expanded.

Class

[TMacro](#)

Syntax

```
property Active: boolean default True;
```

Remarks

When set to True, the macro will be expanded, otherwise macro definition is replaced by null

string. You can use the Active property to modify the SQL property.

The default value is True.

Example

```
PgQuery.SQL.Text := 'SELECT * FROM Dept WHERE DeptNo > 20 &Cond1';  
PgQuery.Macros[0].Value := 'and DName is NULL';  
PgQuery.Macros[0].Active:= False;
```

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20.2.2 AsDateTime Property

Used to set the TDateTime value to a macro.

Class

[TMacro](#)

Syntax

```
property AsDateTime: TDateTime;
```

Remarks

Use the AsDateTime property to set the TDateTime value to a macro.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20.2.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20.2.5 AsString Property

Used to assign the string value to a macro.

Class

[TMacro](#)

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to assign the string value to a macro. Read the AsString property to determine the value of macro represented as a string.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20.2.6 Name Property

Used to identify a particular macro.

Class

[TMacro](#)

Syntax

```
property Name: string;
```

Remarks

Use the Name property to identify a particular macro.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.20.2.7 Value Property

Used to set the value to a macro.

Class

[TMacro](#)

Syntax

```
property value: string;
```

Remarks

Use the Value property to set the value to a macro.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.21 TMacros Class

Controls a list of TMacro objects for the [TCustomDASQL.Macros](#) or [TCustomDADataset](#) components.

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

Unit

[DBAccess](#)

Syntax

```
TMacros = class(TCollection);
```

Remarks

Use TMacros to manage a list of TMacro objects for the [TCustomDASQL](#) or [TCustomDADataset](#) components.

See Also

- [TMacro](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.21.1 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	Finds a macro with the specified 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.21.2 Properties

Properties of the **TMacros** class.

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

Public

Name	Description
Items	Used to iterate through all the macros parameters.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.21.2.1 Items Property(Indexer)

Used to iterate through all the macros parameters.

Class

[TMacros](#)

Syntax

```
property Items[Index: integer]: TMacro; default;
```

Parameters

Index

Holds the index in the range 0..Count - 1.

Remarks

Use the `Items` property to iterate through all macros parameters. `Index` identifies the index in the range `0..Count - 1`.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.21.3 Methods

Methods of the **TMacros** class.

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

Public

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	Finds a macro with the specified 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.21.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.21.3.2 Expand Method

Changes the macros in the passed SQL statement to their values.

Class

[TMacros](#)

Syntax

```
procedure Expand(var SQL: string);
```

Parameters

SQL

Holds the passed SQL statement.

Remarks

Call the Expand method to change the macros in the passed SQL statement to their values.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.1.21.3.3 FindMacro Method

Finds a macro with the specified name.

Class

[TMacros](#)

Syntax

```
function FindMacro(const value: string): TMacro;
```

Parameters

Value

Holds the value of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the [Items](#) property to avoid depending on the order of the items.

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.1.21.3.4 IsEqual Method

Compares itself with another TMacro object.

Class

[TMacros](#)

Syntax

```
function IsEqual(value: TMacros): boolean;
```

Parameters

Value

Holds the values of TMacro objects.

Return Value

True, if the number of TMacro objects and the values of all TMacro objects are equal.

Remarks

Call the IsEqual method to compare itself with another TMacro object. Returns True if the number of TMacro objects and the values of all TMacro objects are equal.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.21.3.5 MacroByName Method

Used to search for a macro with the specified name.

Class

[TMacros](#)

Syntax

```
function MacroByName(const value: string): TMacro;
```

Parameters*Value*

Holds a name of the macro to search for.

Return Value

TMacro object, if a macro with specified name was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Value. If a match is found, MacroByName returns the Macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the [Items](#) property to avoid depending on the order of the items.

To locate a macro by name without raising an exception if the parameter is not found, use the [FindMacro](#) method.

To set a value to a macro, use the [TMacro.Value](#) property.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.21.3.6 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.22 TPoolingOptions Class

This class allows setting up the behaviour of the connection pool.

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

Unit

[DBAccess](#)

Syntax

```
TPoolingOptions = class(TPersistent);
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.22.1 Members

[TPoolingOptions](#) class overview.

Properties

Name	Description
ConnectionLifetime	Used to specify the maximum time during which an open 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.
PoolId	Used to specify an ID for a connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.22.2 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 open 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.
PoolId	Used to specify an ID for a connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.22.2.1 ConnectionLifetime Property

Used to specify the maximum time during which an open connection can be used by connection pool.

Class

[TPoolingOptions](#)

Syntax

```
property ConnectionLifetime: integer default  
DefaultConnectionLifetime;
```

Remarks

Use the ConnectionLifeTime property to specify the maximum time during which an open connection can be used by connection pool. Measured in milliseconds. Pool deletes connections with exceeded connection lifetime when [TCustomDACConnection](#) is about to close. If ConnectionLifetime is set to 0 (by default), then the lifetime of connection is infinite. ConnectionLifetime concerns only inactive connections in the pool.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.1.22.2.2 MaxPoolSize Property

Used to specify the maximum number of connections that can be opened in connection pool.

Class

[TPoolingOptions](#)

Syntax

```
property MaxPoolSize: integer default DefValMaxPoolSize;
```

Remarks

Specifies the maximum number of connections that can be opened in connection pool. Once this value is reached, no more connections are opened. The valid values are 1 and higher.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.22.2.3 MinPoolSize Property

Used to specify the minimum number of connections that can be opened in the connection pool.

Class

[TPoolingOptions](#)

Syntax

```
property MinPoolSize: integer default DefValMinPoolSize;
```

Remarks

Use the MinPoolSize property to specify the minimum number of connections that can be opened in the connection pool.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.22.2.4 PoolId Property

Used to specify an ID for a connection pool.

Class

[TPoolingOptions](#)

Syntax

```
property PoolId: Integer default DefValPoolId;
```

Remarks

Use the PoolId property to make a group of connections use a specific connection pool.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.22.2.5 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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.23 TSmartFetchOptions Class

Smart fetch options are used to set up the behavior of the SmartFetch mode.

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

Unit

[DBAccess](#)

Syntax

```
TSmartFetchOptions = class(TPersistent);
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.23.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.23.2 Properties

Properties of the **TSmartFetchOptions** class.

For a complete list of the **TSmartFetchOptions** class members, see the

[TSmartFetchOptions Members](#) topic.

Published

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.23.2.1 Enabled Property

Sets SmartFetch mode enabled or not.

Class

[TSmartFetchOptions](#)

Syntax

```
property Enabled: Boolean default False;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.23.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.23.2.3 PrefetchedFields Property

List of fields additional to key fields that will be read from the database on dataset open.

Class

[TSmartFetchOptions](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.1.23.2.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.2 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 TCustomDAConnection.OnConnectionLost 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 <code>TCustomDADataset.AfterUpdateExecute</code> and <code>TCustomDADataset.BeforeUpdateExecute</code> events.
-------------------------------------	--

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.2.1 TAfterExecuteEvent Procedure Reference

This type is used for the [TCustomDADataset.AfterExecute](#) and [TCustomDASQL.AfterExecute](#) events.

Unit

[DBAccess](#)

Syntax

```
TAfterExecuteEvent = procedure (Sender: TObject; Result: boolean)
of object;
```

Parameters

Sender

An object that raised the event.

Result

The result is True if SQL statement is executed successfully. False otherwise.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.2.2 TAfterFetchEvent Procedure Reference

This type is used for the [TCustomDADataset.AfterFetch](#) event.

Unit

[DBAccess](#)

Syntax

```
TAfterFetchEvent = procedure (DataSet: TCustomDADataset) of
```

```
object;
```

Parameters

DataSet

Holds the TCustomDADataset descendant to synchronize the record position with.

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.2.3 TBeforeFetchEvent Procedure Reference

This type is used for the [TCustomDADataset.BeforeFetch](#) event.

Unit

[DBAccess](#)

Syntax

```
TBeforeFetchEvent = procedure (DataSet: TCustomDADataset; var  
Cancel: boolean) of object;
```

Parameters

DataSet

Holds the TCustomDADataset descendant to synchronize the record position with.

Cancel

True, if the current fetch operation should be aborted.

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.2.4 TConnectionLostEvent Procedure Reference

This type is used for the [TCustomDACConnection.OnConnectionLost](#) event.

Unit

[DBAccess](#)

Syntax

```
TConnectionLostEvent = procedure (Sender: TObject; Component:  
TComponent; ConnLostCause: TConnLostCause; var RetryMode:  
TRetryMode) of object;
```

Parameters*Sender*

An object that raised the event.

*Component**ConnLostCause*

The reason of the connection loss.

RetryMode

The application behavior when connection is lost.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.2.5 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 .

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.2.6 TDATransactionErrorEvent Procedure Reference

This type is used for the [TDATransaction.OnError](#) event.

Unit

[DBAccess](#)

Syntax

```
TDATransactionErrorEvent = procedure (Sender: TObject; E: EDAError; var Fail: boolean) of object;
```

Parameters

Sender

An object that raised the event.

E

The error code.

Fail

False, if an error dialog should be prevented from being displayed and EAbort exception to cancel the current operation should be raised.

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.2.7 TRefreshOptions Set

Represents the set of [TRefreshOption](#).

Unit

[DBAccess](#)

Syntax

```
TRefreshOptions = set of TRefreshOption;
```

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.11.2.8 TUpdateExecuteEvent Procedure Reference

This type is used for the TCustomDADataset.AfterUpdateExecute and TCustomDADataset.BeforeUpdateExecute events.

Unit

[DBAccess](#)

Syntax

```
TUpdateExecuteEvent = procedure (Sender: TDataSet; StatementTypes:
TStatementTypes; Params: TDAParams) of object;
```

Parameters

Sender

An object that raised the event.

StatementTypes

Holds the type of the SQL statement being executed.

Params

Holds the parameters with which the SQL statement will be executed.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.3 Enumerations

Enumerations in the **DBAccess** unit.

Enumerations

Name	Description
TLabelSet	Sets the language of labels in the connect dialog.
TLockMode	Specifies the lock mode.
TRefreshOption	Indicates when the editing record will be refreshed.
TRetryMode	Specifies the application behavior when connection is lost.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.3.1 TLabelSet Enumeration

Sets the language of labels in the connect dialog.

Unit

[DBAccess](#)

Syntax

```
TLabelSet = (IsCustom, IsEnglish, IsFrench, IsGerman, IsItalian,
IsPolish, IsPortuguese, IsRussian, IsSpanish);
```

Values

Value	Meaning
IsCustom	Set the language of labels in the connect dialog manually.
IsEnglish	Set English as the language of labels in the connect dialog.
IsFrench	Set French as the language of labels in the connect dialog.
IsGerman	Set German as the language of labels in the connect dialog.
IsItalian	Set Italian as the language of labels in the connect dialog.
IsPolish	Set Polish as the language of labels in the connect dialog.
IsPortuguese	Set Portuguese as the language of labels in the connect dialog.
IsRussian	Set Russian as the language of labels in the connect dialog.
IsSpanish	Set Spanish as the language of labels in the connect dialog.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.3.2 TLockMode Enumeration

Specifies the lock mode.

Unit

[DBAccess](#)

Syntax

```
TLockMode = (ImNone, ImPessimistic, ImOptimistic);
```

Values

Value	Meaning
ImNone	No locking occurs. This mode should only be used in single user applications. The default value.
ImOptimistic	Locking occurs when the user posts an edited record, then the lock is released. Locking is done by the RefreshRecord method.
ImPessimistic	Locking occurs when the user starts editing a record. The lock is released after the user has posted or canceled the changes.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.11.3.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.3.4 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.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.4 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.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.4.1 BaseSQLOldBehavior Variable

After assigning SQL text and modifying it by [AddWhere](#), [DeleteWhere](#), and [SetOrderBy](#), all subsequent changes of the SQL property will not be reflected in the BaseSQL property.

Unit

[DBAccess](#)

Syntax

```
BaseSQLOldBehavior: boolean = False;
```

Remarks

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.4.2 ChangeCursor Variable

When set to True allows data access components to change screen cursor for the execution time.

Unit

[DBAccess](#)

Syntax

```
changeCursor: boolean = True;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.11.4.3 SQLGeneratorCompatibility Variable

The value of the [TCustomDADataset.BaseSQL](#) property is used to complete the refresh SQL statement, if the manually assigned [TCustomDAUpdateSQL.RefreshSQL](#) property contains only WHERE clause.

Unit

[DBAccess](#)

Syntax

```
SQLGeneratorCompatibility: boolean = False;
```

Remarks

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12 MemData

This unit contains classes for storing data in memory.

Classes

Name	Description
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 in-memory database.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1 Classes

Classes in the **MemData** unit.

Classes

Name	Description
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 in-memory database.
TObjectType	This class is not used.
TSharedObject	A base class that allows to simplify memory management for object referenced by several other objects.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.2.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.2.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.2.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3 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](#)

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3.1 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](#)

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3.4 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3.5 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3.6 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3.7 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3.8 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.1.3.9 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](#)

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.12.1.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.2.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.2.2.1 Compressed Property

Used to indicate if the Blob is compressed.

Class

[TCompressedBlob](#)

Syntax

```
property Compressed: boolean;
```

Remarks

Indicates whether the Blob is compressed. Set this property to True or False to compress or decompress the Blob.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.2.2.2 CompressedSize Property

Used to indicate compressed size of the Blob data.

Class

[TCompressedBlob](#)

Syntax

```
property CompressedSize: Cardinal;
```

Remarks

Indicates compressed size of the Blob data.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.3 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.4 TMemData Class

Implements in-memory database.

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

Unit

[MemData](#)

Syntax

```
TMemData = class(TData);
```

Inheritance Hierarchy

TData

TMemData

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.12.1.4.1 Members

[TMemData](#) class overview.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.5 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.5.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.5.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.5.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.5.2.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.5.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.5.2.4 Size Property

Used to learn the size of an object instance.

Class

[TObjectType](#)

Syntax

```
property Size: Integer;
```

Remarks

Use the Size property to find out the size of an object instance. Size is a sum of all attribute sizes.

See Also

- [TAttribute.Size](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.5.3 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](#)

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.12.1.5.3.1 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](#)
- [Attributes](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.6 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.6.1 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.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.12.1.6.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.6.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.6.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.6.3.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.1.6.3.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.2 Types

Types in the **MemData** unit.

Types

Name	Description
TLocateExOptions	Represents the set of TLocateExOption .
TUpdateRecKinds	Represents the set of TUpdateRecKind.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.12.2.1 TLocateExOptions Set

Represents the set of [TLocateExOption](#).

Unit

[MemData](#)

Syntax

```
TLocateExOptions = set of TLocateExOption;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.2.2 TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

[MemData](#)

Syntax

```
TUpdateRecKinds = set of TUpdateRecKind;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.3.1 TCompressBlobMode Enumeration

Specifies when the values should be compressed and the way they should be stored.

Unit

[MemData](#)

Syntax

```
TCompressBlobMode = (cbNone, cbClient, cbServer, cbClientServer);
```

Values

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.3.2 TConnLostCause Enumeration

Specifies the cause of the connection loss.

Unit

[MemData](#)

Syntax

```
TConnLostCause = (clUnknown, clExecute, clOpen, clRefresh, clApply, clServiceQuery, clTransStart, clConnectionApply, clConnect);
```

Values

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.12.3.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.12.3.4 TLocateExOption Enumeration

Allows to set additional search parameters which will be used by the LocateEx method.

Unit

[MemData](#)

Syntax

```
TLocateExOption = (lxCasInsensitive, lXPartialKey, lXNearest,
lXNext, lXUp, lXPartialCompare);
```

Values

Value	Meaning
lxCasInsensitive	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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.3.5 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.12.3.6 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1 Classes

Classes in the **MemDS** unit.

Classes

Name	Description
TMemDataSet	A base class for working with data and manipulating data in memory.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1 TMemDataSet Class

A base class for working with data and manipulating data in memory.

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

Unit

[MemDS](#)

Syntax

```
TMemDataSet = class(TDataSet);
```

Remarks

TMemDataSet derives from the TDataSet database-engine independent set of properties, events, and methods for working with data and introduces additional techniques to store and manipulate data in memory.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.1 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	Cancel 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2 Properties

Properties of the **TMemDataSet** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2.2 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. Column names order is significant. Separate names with semicolons. 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 order for the field. If one of these keywords is not used, the default sort order for the field is ascending.

Use `CIS`, `CS` or `BIN` keywords to specify the 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 the 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.

Sorting is performed locally.

Note:

You cannot sort by `BLOB` fields.

`IndexFieldNames` cannot be set to `True` when [TCustomDADataSet.UniDirectional](#)=`True`.

Example

The following procedure illustrates how to set `IndexFieldNames` in response to a button click:

```
DataSet1.IndexFieldNames := 'LastName ASC CIS; DateDue DESC';
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2.4 LocalConstraints Property

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Class

[TMemDataSet](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2.5 LocalUpdate Property

Used to prevent implicit update of rows on database server.

Class

[TMemDataSet](#)

Syntax

```
property LocalUpdate: boolean default False;
```

Remarks

Set the LocalUpdate property to True to prevent implicit update of rows on database server. Data changes are cached locally in client memory.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2.6 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2.7 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2.8 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](#)

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.2.9 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3 Methods

Methods of the **TMemDataSet** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.4 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.5 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.6 DeferredPost Method

Makes permanent changes to the database server.

Class

[TMemDataSet](#)

Syntax

```
procedure DeferredPost;
```

Remarks

Call DeferredPost to make permanent changes to the database server while retaining dataset in its state whether it is dsEdit or dsInsert.

Explicit call to the Cancel method after DeferredPost has been applied does not abandon modifications to a dataset already fixed in database.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.7 EditRangeEnd Method

Enables changing the ending value for an existing range.

Class

[TMemDataSet](#)

Syntax

```
procedure EditRangeEnd;
```

Remarks

Call `EditRangeEnd` to change the ending value for an existing range.

To specify an end range value, call `FieldByName` after calling `EditRangeEnd`.

After assigning a new ending value, call [ApplyRange](#) to activate the modified range.

See Also

- [ApplyRange](#)
- [CancelRange](#)
- [EditRangeStart](#)
- [IndexFieldNames](#)
- [SetRange](#)
- [SetRangeEnd](#)
- [SetRangeStart](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.8 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.9 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 TBlob object for a field or current record when the field itself is known.
GetBlob(const FieldName: string)	Retrieves TBlob object for a field or current record when its name is known.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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](#)

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.13.1.1.3.10 Locate Method

Searches a dataset for a specific record and positions the cursor on it.

Class

[TMemDataSet](#)

Overload List

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

Class

[TMemDataSet](#)

Syntax

```
function Locate(const KeyFields: string; const KeyValues: variant; Options: TLocateOptions): boolean; overload; override;
```

Parameters*KeyFields*

Holds a semicolon-delimited list of field names in which to search.

KeyValues

Holds the variant that specifies the values to match in the key fields.

Options

Holds additional search latitude when searching in string fields.

Return Value

True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

Remarks

Call the Locate method to search a dataset for a specific record and position cursor on it.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. An example is provided below.

Options is a set that optionally specifies additional search latitude when searching in string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when

matching fields. If Options contains the `loPartialKey` setting, then `Locate` allows partial-string matching on strings in `KeyValues`. If Options is an empty set, or if `KeyFields` does not include any string fields, Options is ignored.

`Locate` returns `True` if it finds a matching record, and makes this record the current one. Otherwise it returns `False`.

The `Locate` function works faster when dataset is locally sorted on the `KeyFields` fields. Local dataset sorting can be set with the [TMemDataSet.IndexFieldNames](#) property.

Example

An example of specifying multiple search values:

```
with CustTable do
    Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P',
        '408-431-1000']), [loPartialKey]);
```

See Also

- [TMemDataSet.IndexFieldNames](#)
- [TMemDataSet.LocateEx](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.11 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 <code>TDataSet</code> 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 <code>TDataSet</code> by the specified fields.

[TLocateExOptions](#))

method of TDataSet by the specified field names.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet by the specified field names.

Class

[TMemDataSet](#)

Syntax

```
function LocateEx(const KeyFields: string; const KeyValues:
```

```
variant; Options: TLocateExOptions): boolean; overload;
```

Parameters

KeyFields

Holds the fields to search in.

KeyValues

Holds the values of the fields to search for.

Options

Holds additional search parameters which will be used by the LocateEx method.

Return Value

True, if a matching record was found. Otherwise returns False.

Remarks

Call the LocateEx method when you need some features not to be included to the [TMemDataSet.Locate](#) method of TDataSet.

LocateEx returns True if it finds a matching record, and makes that record the current one. Otherwise LocateEx returns False.

The LocateEx function works faster when dataset is locally sorted on the KeyFields fields. Local dataset sorting can be set with the [TMemDataSet.IndexFieldNames](#) property.

Note: Please add the MemData unit to the "uses" list to use the TLocalExOption enumeration.

See Also

- [TMemDataSet.IndexFieldNames](#)
- [TMemDataSet.Locate](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.12 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.13 RestoreUpdates Method

Marks all records in the cache of updates as unapplied.

Class

[TMemDataSet](#)

Syntax

```
procedure RestoreUpdates;
```

Remarks

Call the RestoreUpdates method to return the cache of updates to its state before calling ApplyUpdates. RestoreUpdates marks all records in the cache of updates as unapplied. It is useful when ApplyUpdates fails.

See Also

- [CachedUpdates](#)

- [TMemDataSet.ApplyUpdates](#)
- [CancelUpdates](#)
- [UpdateStatus](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.14 RevertRecord Method

Cancels changes made to the current record when cached updates are enabled.

Class

[TMemDataSet](#)

Syntax

```
procedure RevertRecord;
```

Remarks

Call the RevertRecord method to undo changes made to the current record when cached updates are enabled.

See Also

- [CachedUpdates](#)
- [CancelUpdates](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.15 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Saves the current dataset data to a stream in the XML format compatible with ADO format.

Class

[TMemDataSet](#)

Syntax

```
procedure SaveToXML(Destination: TStream); overload;
```

Parameters

Destination

Holds a TStream object.

Remarks

Call the SaveToXML method to save the current dataset data to a file or a stream in the XML format compatible with ADO format.

If the destination file already exists, it is overwritten. It remains open from the first call to SaveToXML until the dataset is closed. This file can be read by other applications while it is opened, but they cannot write to the file.

When saving data to a stream, a TStream object must be created and its position must be set in a preferable value.

See Also

- [TVirtualTable.LoadFromFile](#)
- [TVirtualTable.LoadFromStream](#)

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

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.

See Also

- [TVirtualTable.LoadFromFile](#)
- [TVirtualTable.LoadFromStream](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.16 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 ignored when performing a search.

See Also

- [ApplyRange](#)
- [CancelRange](#)
- [EditRangeEnd](#)
- [EditRangeStart](#)
- [IndexFieldNames](#)
- [KeyExclusive](#)
- [SetRangeEnd](#)
- [SetRangeStart](#)

5.13.1.1.3.17 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.18 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.19 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class

[TMemDataSet](#)

Syntax

```
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.3.20 UpdateResult Method

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Class

[TMemDataSet](#)

Syntax

```
function UpdateResult: TUpdateAction;
```

Return Value

a value of the TUpdateAction enumeration.

Remarks

Call the UpdateResult method to read the status of the latest call to the ApplyUpdates method while cached updates are enabled. UpdateResult reflects updates made on the records that have been edited, inserted, or deleted.

UpdateResult works on the record by record basis and is applicable to the current record only.

See Also

- [CachedUpdates](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.13.1.1.3.21 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.4 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.4.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.1.1.4.2 OnUpdateRecord Event

Occurs when a single update component can not handle the updates.

Class

[TMemDataSet](#)

Syntax

```
property OnUpdateRecord: TUpdateRecordEvent;
```

Remarks

Write the OnUpdateRecord event handler to process updates that cannot be handled by a single update component, such as implementation of cascading updates, insertions, or deletions. This handler is also useful for applications that require additional control over parameter substitution in update components.

UpdateKind describes the type of update to perform.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful, it should set UpdateAction to uaApplied before exiting.

See Also

- [CachedUpdates](#)

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.13.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.13.2.1 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.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.13.2.2 SendDataSetChangeEventAfterOpen Variable

The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

Unit

[MemDS](#)

Syntax

```
SendDataSetChangeEventAfterOpen: boolean = True;
```

Remarks

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

To disable sending this event, change the value of this variable to False.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14 PgAccess

This unit contains main components of PgDAC.

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 is used to set up the behaviour of the TPgConnection class.
TPgCursorField	A class providing access to the PostgreSQL cursor fields.
TPgDataSetOptions	This class allows setting up the behaviour of the TCustomPgDataSet class.
TPgDataSource	TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.
TPgDateField	A class providing access to the PostgreSQL date fields.
TPgEncryptor	The class that performs encrypting and decrypting of data.
TPgGeometricField	A class providing access to the PostgreSQL geometric fields.
TPgIntervalField	A class providing access to the PostgreSQL interval fields.
TPgLargeObject	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.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1 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 is used to set up the behaviour of the TPgConnection class.
TPgCursorField	A class providing access to the PostgreSQL cursor fields.
TPgDataSetOptions	This class allows setting up the behaviour of the TCustomPgDataSet class.
TPgDataSource	TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.
TPgDateField	A class providing access to the PostgreSQL date fields.
TPgEncryptor	The class that performs encrypting and decrypting of data.
TPgGeometricField	A class providing access to the PostgreSQL geometric fields.
TPgIntervalField	A class providing access to the PostgreSQL interval fields.
TPgLargeObject	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.
TPgUpdateSQL	Lets you tune update operations for the DataSet component.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1 TCustomPgDataSet Class

A base class defining functionality for the classes derived from it.

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

Unit

[PgAccess](#)

Syntax

```
TCustomPgDataSet = class(TCustomDADataset);
```

Remarks

TCustomPgDataSet is a base dataset component that defines functionality for classes derived from it. Applications never use TCustomPgDataSet objects directly. Instead they use descendants of TCustomPgDataSet, such as TPgQuery and TPgTable that inherit its dataset-related properties and methods.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

TCustomPgDataSet

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.1 Members

[TCustomPgDataSet](#) 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	Used to fetch data from the REF_CURSOR parameter or REF_CURSOR field.
DataTypeMap (inherited from TCustomDADataset)	Used to set data type mapping rules
Debug (inherited from TCustomDADataset)	Used to display the statement that is being executed and the values and types of its 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	Used to refresh record by RETURNING clause when insert or update is performed.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence	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.
LastInsertOID	Returns OID of the record inserted by the last query for

	table with OIDs.
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 (inherited from TCustomDADDataSet)	Used to get the number of macros associated with the Macros property.
Macros (inherited from TCustomDADDataSet)	Makes it possible to change SQL queries easily.
MasterFields (inherited from TCustomDADDataSet)	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 TCustomDADDataSet)	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 TCustomDADDataSet)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from TCustomDADDataSet)	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 TCustomDADDataSet)	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 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	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	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 find out whether TCustomDADataset has 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)	Finds a macro with the specified name.
FindNearest (inherited from TCustomDADataset)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam	Searches for and returns a parameter with the specified name.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

GetDataType (inherited from TCustomDADataset)	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject (inherited from TCustomDADataset)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from TCustomDADataset)	Retrieves the precision of a number field.
GetFieldScale (inherited from TCustomDADataset)	Retrieves the scale of a number field.
GetKeyFieldNames (inherited from TCustomDADataset)	Provides a list of available key field names.
GetOrderBy (inherited from TCustomDADataset)	Retrieves an ORDER BY clause from a SQL statement.
GetPgCursor	Retrieves a TPgCursor object for a field with known name.
GetPgDate	Retrieves a TPgDate object for a field with known name.
GetPgInterval	Retrieves a TPgInterval object for a field with known name.
GetPgLargeObject	Retrieves a TPgLargeObject object for a field with known name.
GetPgRow	Retrieves a TPgRow object for a field with known name.
GetPgTime	Retrieves a TPgTime object for a field with known name.
GetPgTimeStamp	Retrieves a TPgTimeStamp object for a field with known name.
GotoCurrent (inherited from TCustomDADataset)	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 TCustomDADataset)	Locks the current record.
MacroByName (inherited from TCustomDADataset)	Finds a macro with the specified 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2 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.
Cursor	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 the statement that is being executed and the values and types of its 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	Used to refresh record by RETURNING clause when insert or update is performed.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence	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.

LastInsertOID	Returns OID of the record inserted by the last query for table with OIDs.
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 (inherited from TCustomDADataset)	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.
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 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	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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2.1 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](#)
- [TPgCursorField.AsCursor](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2.2 DMLRefresh Property

Used to refresh record by RETURNING clause when insert or update is performed.

Class

[TCustomPgDataSet](#)

Syntax

```
property DMLRefresh: boolean;
```

Remarks

Use the DMLRefresh property to refresh record by RETURNING clause when insert or update is performed.

The default value is False.

Note: When the DMLRefresh property is set to True, the value of

[TCustomDADataset.RefreshOptions](#) is ignored to avoid refetching field values from the server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2.3 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

[TCustomPgDataSet](#)

Syntax

```
property FetchAll: boolean default True;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

Opening a dataset in FetchAll = False mode requires an active transaction. When the FetchAll property is False, the first call to [TMemDataSet.Locate](#) and [TMemDataSet.LocateEx](#) methods may take a lot of time to retrieve additional records to the client side.

See Also

- [TCustomDADataset.FetchRows](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2.4 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

- [TCustomDADDataSet.KeyFields](#)
- [SequenceMode](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2.5 LastInsertOID Property

Returns OID of the record inserted by the last query for table with OIDs.

Class

[TCustomPgDataSet](#)

Syntax

```
property LastInsertOID: Int64;
```

Remarks

Use the LastInsertOID property to get OID of the record inserted by the last query for table with OIDs.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2.6 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.
DistinctParams	Used for correct TClientDataSet parameters handling.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
ExtendedFieldsInfo	Used to perform an additional query to get information about the returned fields and the tables they belong to.
FullRefresh	Used to specify the fields to include in the automatically generated SQL statement when calling the method.
OIDAsInt	Used to read OID fields as integer values and map these fields to TIntegerField.
PrefetchRows	Used to set the number of rows to be prefetched during the execution of a query.
PrepareUpdateSQL	Used to automatically prepare update queries before execution.
SetEmptyStrToNull	Force replace of empty strings with NULL values in data. The default value is False.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2.7 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2.8 SequenceMode Property

Used to specify the methods used internally to generate a sequenced field.

Class

[TCustomPgDataSet](#)

Syntax

```
property SequenceMode: TSequenceMode default smPost;
```

Remarks

Set the SequenceMode property to specify which method is used internally to generate a sequenced field.

See Also

- [TCustomDADataset.KeyFields](#)
- [KeySequence](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.2.9 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3 Methods

Methods of the **TCustomPgDataSet** class.

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

Public

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.
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 find out whether TCustomDADataset has

	<p>fetches all rows.</p> <p>Used to learn whether TCustomDADataset is still fetching rows.</p>
Fetching (inherited from TCustomDADataset)	
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)	Finds a macro with the specified name.
FindNearest (inherited from TCustomDADataset)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam	Searches for and returns a parameter with the specified name.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from TCustomDADataset)	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject (inherited from TCustomDADataset)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from TCustomDADataset)	Retrieves the precision of a number field.
GetFieldScale (inherited from TCustomDADataset)	Retrieves the scale of a number field.
GetKeyFieldNames (inherited from TCustomDADataset)	Provides a list of available key field names.
GetOrderBy (inherited from TCustomDADataset)	Retrieves an ORDER BY clause from a SQL statement.
GetPgCursor	Retrieves a TPgCursor object for a field with known name.

GetPgDate	Retrieves a TPgDate object for a field with known name.
GetPgInterval	Retrieves a TPgInterval object for a field with known name.
GetPgLargeObject	Retrieves a TPgLargeObject object for a field with known name.
GetPgRow	Retrieves a TPgRow object for a field with known name.
GetPgTime	Retrieves a TPgTime object for a field with known name.
GetPgTimeStamp	Retrieves a TPgTimeStamp object for a field with known name.
GotoCurrent (inherited from TCustomDADataset)	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 TCustomDADataset)	Locks the current record.
MacroByName (inherited from TCustomDADataset)	Finds a macro with the specified 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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.5 GetPgInterval Method

Retrieves a TPgInterval object for a field with known name.

Class

[TCustomPgDataSet](#)

Syntax

```
function GetPgInterval(const FieldName: string): TPgInterval;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TPgInterval object for a field with known name.

Remarks

Call the GetPgInterval method to retrieve a TPgInterval object for a field when only its name is known. FieldName is the name of an existing field.

See Also

- [TPgInterval](#)
- [TPgParam.AsPgInterval](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.6 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.7 `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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.8 `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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.9 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.10 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.1.3.11 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.2 TCustomPgStoredProc Class

A component for accessing and executing stored procedures and functions.

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

Unit

[PgAccess](#)

Syntax

```
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](#)

[TCustomDADDataSet](#)

[TCustomPgDataSet](#)

TCustomPgStoredProc

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.14.1.2.1 Members

[TCustomPgStoredProc](#) 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 TCustomDADataset)	Used to set data type mapping rules
Debug (inherited from TCustomDADataset)	Used to display the statement that is being executed and the values and types of its 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 TCustomPgDataSet)	Used to refresh record by RETURNING clause when insert or update is performed.

FetchAll (inherited from TCustomPgDataSet)	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataSet)	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.
LastInsertOID (inherited from TCustomPgDataSet)	Returns OID of the record inserted by the last query for table with OIDs.
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 (inherited from TCustomDADataset)	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.
Options (inherited from TCustomPgDataSet)	Used to specify the behaviour of TCustomPgDataSet object.
Overload	Used to specify the overloading number.
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 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.

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.
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.
ExecProc	Executes a SQL statement on the server.
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 find out whether TCustomDADataset has 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)	Finds a macro with the specified name.
FindNearest (inherited from TCustomDADataset)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from TCustomPgDataset)	Searches for and returns a parameter with the specified name.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from TCustomDADataset)	Returns internal field types defined in the MemData and

	accompanying modules.
GetFieldObject (inherited from TCustomDADataset)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from TCustomDADataset)	Retrieves the precision of a number field.
GetFieldScale (inherited from TCustomDADataset)	Retrieves the scale of a number field.
GetKeyFieldNames (inherited from TCustomDADataset)	Provides a list of available key field names.
GetOrderBy (inherited from TCustomDADataset)	Retrieves an ORDER BY clause from a SQL statement.
GetPgCursor (inherited from TCustomPgDataset)	Retrieves a TPgCursor object for a field with known name.
GetPgDate (inherited from TCustomPgDataset)	Retrieves a TPgDate object for a field with known name.
GetPgInterval (inherited from TCustomPgDataset)	Retrieves a TPgInterval object for a field with known name.
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 TCustomDADataset)	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 TCustomDADataset)	Locks the current record.
MacroByName (inherited from TCustomDADataset)	Finds a macro with the specified 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 TCustomDADataset)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Describes the parameters of a stored procedure.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.2.2 Properties

Properties of the **TCustomPgStoredProc** class.

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

Public

Name	Description
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 the statement that is being executed and the values and types of its 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 TCustomPgDataset)	Used to refresh record by RETURNING clause when insert or update is performed.
FetchAll (inherited from TCustomPgDataset)	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataset)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataset)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataset)	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.

LastInsertOID (inherited from TCustomPgDataSet)	Returns OID of the record inserted by the last query for table with OIDs.
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 (inherited from TCustomDADataset)	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.
Options (inherited from TCustomPgDataSet)	Used to specify the behaviour of TCustomPgDataSet object.
Overload	Used to specify the overloading number.
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 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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.2.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.2.3 Methods

Methods of the **TCustomPgStoredProc** class.

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

Public

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.
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.
ExecProc	Executes a SQL statement on the server.
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 find out whether TCustomDADataset has 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)	Finds a macro with the specified name.
FindNearest (inherited from TCustomDADataset)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from TCustomPgDataSet)	Searches for and returns a parameter with the specified name.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from TCustomDADataset)	Returns internal field types defined in the MemData and accompanying modules.

GetFieldObject (inherited from TCustomDADataset)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from TCustomDADataset)	Retrieves the precision of a number field.
GetFieldScale (inherited from TCustomDADataset)	Retrieves the scale of a number field.
GetKeyFieldNames (inherited from TCustomDADataset)	Provides a list of available key field names.
GetOrderBy (inherited from TCustomDADataset)	Retrieves an ORDER BY clause from a SQL statement.
GetPgCursor (inherited from TCustomPgDataset)	Retrieves a TPgCursor object for a field with known name.
GetPgDate (inherited from TCustomPgDataset)	Retrieves a TPgDate object for a field with known name.
GetPgInterval (inherited from TCustomPgDataset)	Retrieves a TPgInterval object for a field with known name.
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 TCustomDADataset)	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 TCustomDADataset)	Locks the current record.
MacroByName (inherited from TCustomDADataset)	Finds a macro with the specified 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 TCustomDADataset)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Describes the parameters of a stored procedure.
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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.2.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.2.3.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.3 TCustomPgTable Class

A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.

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

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.3.1 Members

[TCustomPgTable](#) 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 TCustomDADataset)	Used to set data type mapping rules
Debug (inherited from TCustomDADataset)	Used to display the statement that is being executed and the values and types of its 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 TCustomPgDataset)	Used to refresh record by RETURNING clause when insert or update is performed.
FetchAll (inherited from TCustomPgDataset)	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataset)	Used to get or set the list of fields on which the recordset is sorted.

IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataSet)	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.
LastInsertOID (inherited from TCustomPgDataSet)	Returns OID of the record inserted by the last query for table with OIDs.
Limit	Used to set the number of rows retrieved from the query.
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 (inherited from TCustomDADataset)	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 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.

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.
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 find out whether TCustomDADataset has 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)	Finds a macro with the specified name.
FindNearest (inherited from TCustomDADataset)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from TCustomPgDataset)	Searches for and returns a parameter with the specified name.
GetBlob (inherited from TMemDataset)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from TCustomDADataset)	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject (inherited from TCustomDADataset)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from TCustomDADataset)	Retrieves the precision of a number field.
GetFieldScale (inherited from TCustomDADataset)	Retrieves the scale of a number field.
GetKeyFieldNames (inherited from TCustomDADataset)	Provides a list of available key field names.
GetOrderBy (inherited from TCustomDADataset)	Retrieves an ORDER BY clause from a SQL statement.
GetPgCursor (inherited from TCustomPgDataset)	Retrieves a TPgCursor object for a field with known name.
GetPgDate (inherited from TCustomPgDataset)	Retrieves a TPgDate object for a field with known name.

GetPgInterval (inherited from TCustomPgDataSet)	Retrieves a TPgInterval object for a field with known name.
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 TCustomDADataset)	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 TCustomDADataset)	Locks the current record.
MacroByName (inherited from TCustomDADataset)	Finds a macro with the specified 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 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.3.2 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 the statement that is being executed and the values and types of its 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 TCustomPgDataSet)	Used to refresh record by RETURNING clause when insert or update is performed.
FetchAll (inherited from TCustomPgDataSet)	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a

	query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataSet)	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.
LastInsertOID (inherited from TCustomPgDataSet)	Returns OID of the record inserted by the last query for table with OIDs.
Limit	Used to set the number of rows retrieved from the query.
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 (inherited from TCustomDADataset)	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 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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.3.2.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.3.2.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.4 TCustomPgTimeStampField Class

A base class defining functionality for the classes derived from it.

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

Unit

[PgAccess](#)

Syntax

```
TCustomPgTimeStampField = class(TField);
```

Remarks

TCustomPgTimeStampField is a base dataset component that defines functionality for classes derived from it. Applications never use TCustomPgTimeStampField objects directly. Instead they use descendants of TCustomPgTimeStampField

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.4.1 Members

[TCustomPgTimeStampField](#) class overview.

Properties

Name	Description
AsPgTimeStamp	Used to provide access to a

	TPgTimeStamp object.
--	----------------------

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.14.1.4.2 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](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.14.1.4.2.1 AsPgTimeStamp Property

Used to provide access to a TPgTimeStamp object.

Class

[TCustomPgTimeStampField](#)

Syntax

```
property ASPgTimeStamp: TCustomPgTimeStamp;
```

Remarks

Use the AsTimeStamp property to get access to a TPgTimeStamp object which you can use for manipulations with timestamp value.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5 TPgConnection Class

Represents an open connection to a PostgreSQL database.

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

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 interoperability with PostgreSQL database.

TPgConnection object represents a unique connection to PostgreSQL database.

Note: If the port differs from the default one (5432) please use the Port property to set it.

Inheritance Hierarchy

[TCustomDAConnection](#)

TPgConnection

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.1 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 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 TCustomDACConnection)	Performs disconnect.
ExecProc (inherited from TCustomDACConnection)	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx (inherited from TCustomDACConnection)	Allows to execute a stored procedure or function.
ExecSQL (inherited from TCustomDACConnection)	Executes a SQL statement with parameters.
ExecSQLEx (inherited from TCustomDACConnection)	Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames (inherited from TCustomDACConnection)	Returns a database list from the server.
GetKeyFieldNames (inherited from TCustomDACConnection)	Provides a list of available key field names.
GetRowType	Overloaded. Used to get a TPgRowType object for composite type.
GetStoredProcNames (inherited from TCustomDACConnection)	Returns a list of stored procedures from the server.
GetTableNames (inherited from TCustomDACConnection)	Provides a list of available tables names.
MonitorMessage (inherited from TCustomDACConnection)	Sends a specified message through the TCustomDASQLMonitor component.
Ping (inherited from TCustomDACConnection)	Used to check state of connection to the server.
ReleaseSavepoint	Releases the specified savepoint without affecting any work that has been performed after its creation.
RemoveFromPool (inherited from	Marks the connection that should not be returned to the pool after disconnect.

TCustomDACConnection)	
Rollback (inherited from TCustomDACConnection)	Discards all current data changes and ends transaction.
RollbackToSavepoint	Cancels all updates for the current transaction.
Savepoint	Defines a point in the transaction to which you can roll back later.
StartTransaction	Overloaded. Starts a new user transaction against the database server.

Events

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.
OnNotice	Occurs when a message or notice is received from PostgreSQL server.
OnNotification	Occurs when an asynchronous notification is received from PostgreSQL server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2 Properties

Properties of the **TPgConnection** class.

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

Public

Name	Description
ConnectDialog (inherited from TCustomDAConnection)	Allows to link a TCustomConnectDialog component.
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.
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.
Pooling (inherited from TCustomDAConnection)	Enables or disables using connection pool.
PoolingOptions (inherited from TCustomDAConnection)	Specifies the behaviour of connection pool.
ProcessID	Returns the process ID (PID) of the backend server process handling this connection.
ServerVersion	Holds the PostgreSQL server version number.
ServerVersionFull	Holds full PostgreSQL server version including version number and some additional information.

Published

Name	Description
ConnectionTimeout	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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.1 ConnectionTimeout Property

Used to set the time to wait while trying to establish a connection before terminating the attempt and generating an error.

Class

[TPgConnection](#)

Syntax

```
property ConnectionTimeout: integer default
```

```
DefaultConnectionTimeout;
```

Remarks

Use the ConnectionTimeout property to set the time to wait while trying to establish a connection before terminating the attempt and generating an error. The 0 value indicates no limit. The default value of ConnectionTimeout is 15 seconds.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.2 Database Property

Used to specify the name of the database to be used once a connection is open.

Class

[TPgConnection](#)

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

- [TCustomDACConnection.Server](#)
- [Port](#)
- [TCustomDACConnection.Username](#)
- [TCustomDACConnection.Password](#)
- [TCustomDACConnection.GetDatabaseNames](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.3 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.
EnableBCD	Used to enable currency type. Default value of this option is False.
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.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
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.
ImmediateNotices	Sets the behavior of receiving notices from

	PostgreSQL.
IPVersion	Used to specify Internet Protocol version.
MessagesCharset	Used to set a character set for PostgreSQL error messages.
MultipleConnections	Used to enable or disable the creation of an additional internal connection for TPgQuery, when necessary.
UseUnicode	Used to specify a value indicating whether the UTF8 charset will be used.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.4 Password Property

Used to specify a password for a connection.

Class

[TPgConnection](#)

Syntax

```
property Password: string;
```

Remarks

Use the Password property to specify a password for a connection. When property is being changed TPgConnection calls Disconnect method.

See Also

- [Username](#)
- [Server](#)
- [TCustomDACConnection.Connect](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.5 Port Property

Used to specify the port for connection.

Class

[TPgConnection](#)

Syntax

```
property Port: integer default PgDefValPort;
```

Remarks

Use the Port property to specify the port number of PostgreSQL database for connection.

See Also

- [TCustomDACConnection.Server](#)
- [Database](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.6 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).

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.14.1.5.2.7 ProtocolVersion Property

Used to set the version of protocol for communication with PostgreSQL server.

Class

[TPgConnection](#)

Syntax

```
property ProtocolVersion: TProtocolVersion default  
DefaultProtocol;
```

Remarks

Set ProtocolVersion to pv20 to work with PostgreSQL server version 7.3 or older that don't support protocol version 3.0. Set ProtocolVersion to pv30 to enforce the protocol version 3.0. Set ProtocolVersion to pvAuto to automatically select between protocol versions depending on the specific query. The default value is pvAuto.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.8 Schema Property

Used to change the search path of the connection to the specified schema, or get the first value from the search path.

Class

[TPgConnection](#)

Syntax

```
property Schema: string stored IsSchemaStored;
```

Remarks

Use the Schema property to change the search path of the connection to the specified schema, or get the first value from the search path.

Set the Schema property to change the search path of the connection to the specified

schema. When connection is open, read the value of the property to get the name of the current schema (the first value from the current search path).

The TPgQuery, TPgStoredProc and other PgDAC components can implicitly use this property. Therefore you may have problems in case your search path contains several schemas and you use objects not from the current one.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.9 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](#)
- [TCustomDACConnection.Connect](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.10 ServerVersion Property

Holds the PostgreSQL server version number.

Class

[TPgConnection](#)

Syntax

```
property ServerVersion: string;
```

Remarks

Read the ServerVersion property to get the PostgreSQL server version number in a string, for a example '8.2.5'

Works only when a TPgConnection instance is connected.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.11 ServerVersionFull Property

Holds full PostgreSQL server version including version number and some additional information.

Class

[TPgConnection](#)

Syntax

```
property ServerVersionFull: string;
```

Remarks

Read the ServerVersionFull property to get full PostgreSQL server version including version number and some additional information.

Works only when a TPgConnection instance is connected.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.12 SSLOptions Property

Used to set the properties required for protected SSL connection with the server.

Class

[TPgConnection](#)

Syntax

```
property SSLOptions: TPgConnectionSSLOptions;
```

Remarks

Use the SSLOptions property to set the properties required for protected SSL connection with the server.

Descriptions of all options are in the table below.

Option Name	Description
Mode	Used to determine whether or with what priority an SSL connection will be negotiated with the server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.2.13 Username Property

Contains username.

Class

[TPgConnection](#)

Syntax

```
property Username: string;
```

Remarks

Use the Username property to supply a user name to handle server's request for a login. When the property is being changed TPgConnection calls Disconnect method.

Note: The UserName property will be used as a default value for the Database and Schema parameters if they are empty.

See Also

- [Password](#)

- [Server](#)
- [TCustomDAConnection.Connect](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3 Methods

Methods of the **TPgConnection** class.

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

Public

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 TCustomDACConnection)	Allows to execute a stored procedure or function.
ExecSQL (inherited from TCustomDACConnection)	Executes a SQL statement with parameters.
ExecSQLEx (inherited from TCustomDACConnection)	Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames (inherited from TCustomDACConnection)	Returns a database list from the server.
GetKeyFieldNames (inherited from TCustomDACConnection)	Provides a list of available key field names.
GetRowType	Overloaded. Used to get a TPgRowType object for composite type.
GetStoredProcNames (inherited from TCustomDACConnection)	Returns a list of stored procedures from the server.
GetTableNames (inherited from TCustomDACConnection)	Provides a list of available tables names.
MonitorMessage (inherited from TCustomDACConnection)	Sends a specified message through the TCustomDASQLMonitor component.
Ping (inherited from TCustomDACConnection)	Used to check state of connection to the server.
ReleaseSavepoint	Releases the specified savepoint without affecting any work that has been performed after its creation.
RemoveFromPool (inherited from TCustomDACConnection)	Marks the connection that should not be returned to the pool after disconnect.
Rollback (inherited from TCustomDACConnection)	Discards all current data changes and ends transaction.
RollbackToSavepoint	Cancel 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.2 CreateDataSet Method

Returns a new instance of TPgQuery class and associates it with the connection object.

Class

[TPgConnection](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.3 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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.4 CreateSQL Method

Returns a new instance of TPgSQL class and associates it with the connection object.

Class

[TPgConnection](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.5 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.6 GetRow Type Method

Used to get a TPgRowType object for composite type.

Class

[TPgConnection](#)

Overload List

Name	Description
GetRowType	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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Used to get a TPgRowType object for composite type by its OID.

Unit

Syntax

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.7 ReleaseSavepoint Method

Releases the specified savepoint without affecting any work that has been performed after its creation.

Class

[TPgConnection](#)

Syntax

```
procedure ReleaseSavepoint(const Name: string);
```

Parameters

Name

Holds the savepoint name.

Remarks

Call the ReleaseSavepoint method to release the specified savepoint without affecting any work that has been performed after its creation.

See Also

- [RollbackToSavepoint](#)
- [Savepoint](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.8 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.9 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.3.10 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Starts a new user transaction against the database server.

Class

[TPgConnection](#)

Syntax

```
procedure StartTransaction; overload; override;
```

Remarks

StartTransaction is an overload method for [TCustomDACConnection.StartTransaction](#). Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction property. If InTransaction is True, it indicates that a transaction is already in progress, a subsequent call to StartTransaction without first calling [TCustomDACConnection.Commit](#) or [TCustomDACConnection.Rollback](#) to end the current transaction raises EDatabaseError. Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes or Rollback to cancel them.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.4 Events

Events of the **TPgConnection** class.

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

Public

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](#)

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.4.1 OnNotice Event

Occurs when a message or notice is received from PostgreSQL server.

Class

[TPgConnection](#)

Syntax

```
property OnNotice: TPgNoticeEvent;
```

Remarks

The OnNotice event occurs when PostgreSQL server sends a message or multiple messages with Severity lower than ERROR.

To set the message levels to be sent to the client, use the PostgreSQL client_min_messages configuration parameter. For more information refer to the PostgreSQL documentation.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.5.4.2 OnNotification Event

Occurs when an asynchronous notification is received from PostgreSQL server.

Class

[TPgConnection](#)

Syntax

```
property OnNotification: TPgNotificationEvent;
```

Remarks

The OnNotification event occurs when an asynchronous notification is received from PostgreSQL server. Notification can be sent using NOTIFY command. To receive notifications use LISTEN command. To stop receiving notifications use UNLISTEN command.

You can read more information about these commands in the PostgreSQL documentation.

Note: Notifications can be received only when a call to the server is performed (for example, on executing SQL statement). While the connection is idle, notifications are not received. To receive notifications while the connection is idle, use the TPgAlerter component.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6 TPgConnectionOptions Class

This class allows setting up the behaviour of the TPgConnection class.

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

Unit

[PgAccess](#)

Syntax

```
TPgConnectionOptions = class(TDACConnectionOptions);
```

Inheritance Hierarchy

[TDACConnectionOptions](#)

TPgConnectionOptions

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.1 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.
EnableBCD	Used to enable currency type. Default value of this option is False.
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.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
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.
ImmediateNotices	Sets the behavior of receiving notices from PostgreSQL.
IPVersion	Used to specify Internet Protocol version.
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 TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
MessagesCharset	Used to set a character set for PostgreSQL error messages.
MultipleConnections	Used to enable or disable the creation of an additional internal connection for TPgQuery, when necessary.
UseUnicode	Used to specify a value indicating whether the UTF8 charset will be used.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2 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 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.
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 TCustomDAConnection.OnConnectionLost 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 <code>pg_stat_activity</code> view and included in CSV log entries. Only printable ASCII characters may be used in the <code>ApplicationName</code> 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.
EnableBCD	Used to enable currency type. Default value of this option is False.
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.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
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.
ImmediateNotices	Sets the behavior of receiving notices from PostgreSQL.
IPVersion	Used to specify Internet Protocol version.
MessagesCharset	Used to set a character set for PostgreSQL error messages.
MultipleConnections	Used to enable or disable the creation of an additional internal connection for TPgQuery, when necessary.
UseUnicode	Used to specify a value indicating whether the UTF8 charset will be used.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.2 Charset Property

Used to set the character set that PgDAC uses to read and write character data.

Class

[TPgConnectionOptions](#)

Syntax

```
property Charset: string;
```

Remarks

Use the Charset property to set the character set that PgDAC uses to read and write character data.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.3 EnableBCD Property

Used to enable currency type. Default value of this option is False.

Class

[TPgConnectionOptions](#)

Syntax

```
property EnableBCD: boolean;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.4 EnableComposites Property

Used to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.

Class

[TPgConnectionOptions](#)

Syntax

```
property EnableComposites: boolean default True;
```

Remarks

Use the EnableComposites property to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.

When EnableComposites is set to True, PgDAC detects fields of composite type and creates separate fields for each attribute of composite type. If the ObjectView property is set to True, PgDAC also creates an instance of TADTField that is parent for all attributes fields.

When EnableComposites is False, fields of composite types are not detected and mapped to TStringField, TMemofield, TWideStringField, or TWideMemofield depending on the UnknownAsString and UseUnicode options.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.5 EnableDomains Property

Used to map fields of domain types to TField with the base domain type.

Class

[TPgConnectionOptions](#)

Syntax

```
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.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.6 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class

[TPgConnectionOptions](#)

Syntax

```
property EnableFMTBCD: boolean;
```

Remarks

Use the EnableFMTBCD property to enable using FMTBCD instead of float for large integer numbers to keep precision.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.7 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.8 EnablePgTimeStamps Property

Used to map DATE, TIME, and TIMESTAMP fields to TPgDateField, TPgTimeField, and TPgTimeStampField accordingly.

Class

[TPgConnectionOptions](#)

Syntax

```
property EnablePgTimeStamps: boolean default False;
```

Remarks

Use the `EnablePgTimeStamp` property to map `DATE`, `TIME`, and `TIMESTAMP` fields to `TPgDateField`, `TPgTimeField`, and `TPgTimeStampField` accordingly.

By default this option is set to `False`, and `DATE`, `TIME`, and `TIMESTAMP` fields are mapped to standard `TDateField`, `TTimeField`, and `TDateTimeField`. Set this option to `True` to work with values than can be represented by `TDateTime` variable. These values include dates before 01-01-0100 or after 12-31-9999 and special values `INFINITY` and `-INFINITY` that can hold `TIMESTAMP` field. You can work with such values using special PgDAC field types: `TPgDateField`, `TPgTimeField`, and `TPgTimeStampField`. `TPgTimeField` and `TPgTimeStampField` allows also getting timezone value for fields `WITH TIMEZONE`.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.9 ImmediateNotices Property

Sets the behavior of receiving notices from PostgreSQL.

Class

[TPgConnectionOptions](#)

Syntax

```
property ImmediateNotices: boolean default False;
```

Remarks

If `True`, the [TPgConnection.OnNotice](#) event occurs immediately without waiting for the SQL query to be completed. The default value is `False`.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.10 IPVersion Property

Used to specify Internet Protocol version.

Class

[TPgConnectionOptions](#)

Syntax

```
property IPVersion: TIPVersion default DefValIPVersion;
```

Remarks

Use the IPVersion property to specify Internet Protocol Version.

- **ivIPBoth** - either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) is used.
- **ivIPv4** - The default value. Internet Protocol Version 4 (IPv4) is used.
- **ivIPv6** - Internet Protocol Version 6 (IPv6) is used.

Note: When the TIPVersion property is set to **ivIPBoth** , a connection attempt is made via IPv6 if it is enabled in the operating system settings. If the connection attempt fails, a new connection attempt is made via IPv4.

See Also

- [TIPVersion](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.11 MessagesCharset Property

Used to set a character set for PostgreSQL error messages.

Class

[TPgConnectionOptions](#)

Syntax

```
property MessagesCharset: string;
```

Remarks

If the property value is empty, the character set from PostgreSQL settings will be used.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.12 MultipleConnections Property

Used to enable or disable the creation of an additional internal connection for TPgQuery, when necessary.

Class

[TPgConnectionOptions](#)

Syntax

```
property MultipleConnections: boolean default  
DefaultMultipleConnections;
```

Remarks

Use the MultipleConnections property to enable or disable the creation of an additional internal connection for TPgQuery, when necessary. The default value is True.

See Also

- [TPgQuery](#)

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.6.2.13 UseUnicode Property

Used to specify a value indicating whether the UTF8 charset will be used.

Class

[TPgConnectionOptions](#)

Syntax

```
property UseUnicode: boolean default DefaultUseUnicode;
```

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.7 TPgConnectionSSLOptions Class

This class is used to set up the behaviour of the TPgConnection class.

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

Unit

[PgAccess](#)

Syntax

```
TPgConnectionSSLOptions = class(TDACConnectionSSLOptions);
```

Inheritance Hierarchy

[TDACConnectionSSLOptions](#)

TPgConnectionSSLOptions

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.7.1 Members

[TPgConnectionSSLOptions](#) class overview.

Properties

Name	Description
CACert (inherited from TDACConnectionSSLOptions)	Holds the path to the certificate authority file.
Cert (inherited from TDACConnectionSSLOptions)	Holds the path to the client certificate.
CipherList (inherited from TDACConnectionSSLOptions)	Holds the list of allowed SSL ciphers.
Key (inherited from TDACConnectionSSLOptions)	Holds the path to the private client key.
Mode	Used to determine whether or with what priority an SSL connection will be negotiated with the server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.7.2 Properties

Properties of the **TPgConnectionSSLOptions** class.

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

Published

Name	Description
CACert (inherited from TDACConnectionSSLOptions)	Holds the path to the certificate authority file.
Cert (inherited from TDACConnectionSSLOptions)	Holds the path to the client certificate.
CipherList (inherited from TDACConnectionSSLOptions)	Holds the list of allowed SSL ciphers.
Key (inherited from TDACConnectionSSLOptions)	Holds the path to the private client key.
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.7.2.1 Mode Property

Used to determine whether or with what priority an SSL connection will be negotiated with the server.

Class

[TPgConnectionSSLOptions](#)

Syntax

```
property Mode: TSSLMode default DefValTSSLMode;
```

Remarks

Use the Mode property to determine whether or with what priority an SSL connection will be negotiated with the server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.8 TPgCursorField Class

A class providing access to the PostgreSQL cursor fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgCursorField = class(TDACursorField);
```

Remarks

The TPgCursorField class provides access to the PostgreSQL cursor fields.

Inheritance Hierarchy

TDACursorField

TPgCursorField

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.8.1 Members

[TPgCursorField](#) class overview.

Properties

Name	Description
AsCursor	Used to provide access to a TPgCursor object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.8.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.8.2.1 AsCursor Property

Used to provide access to a TPgCursor object.

Class

[TPgCursorField](#)

Syntax

```
property AsCursor: TPgRefCursor;
```

Remarks

Use the AsCursor property to provide access to a TPgCursor object you can use for

manipulations with the cursor value.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9 TPgDataSetOptions Class

This class allows setting up the behaviour of the TCustomPgDataSet class.

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

Unit

[PgAccess](#)

Syntax

```
TPgDataSetOptions = class(TDADatasetOptions);
```

Inheritance Hierarchy

[TDADatasetOptions](#)

TPgDataSetOptions

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.1 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 TDADatasetOptions)	Used to execute automatic TCustomDADataset.Prepare on the query execution.
CacheBlobs	Used to allocate local memory buffer to hold a copy of the large object

	content.
CacheCalcFields (inherited from TDADatasetOptions)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode (inherited from TDADatasetOptions)	Used to store values of the BLOB fields in compressed form.
CursorWithHold	Used to open query in the FetchAll=False mode without transaction.
DefaultValues (inherited from TDADatasetOptions)	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DeferredBlobRead	Used to fetch values of large objects when they are explicitly requested.
DetailDelay (inherited from TDADatasetOptions)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
DistinctParams	Used for correct TClientDataSet parameters handling.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
ExtendedFieldsInfo	Used to perform an additional query to get information about the returned fields and the tables they belong to.
FieldsOrigin (inherited from TDADatasetOptions)	Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.

FlatBuffers (inherited from TDADatasetOptions)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
FullRefresh	Used to specify the fields to include in the automatically generated SQL statement when calling the method.
InsertAllSetFields (inherited from TDADatasetOptions)	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail (inherited from TDADatasetOptions)	Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from TDADatasetOptions)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from TDADatasetOptions)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange (inherited from TDADatasetOptions)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
OIDAsInt	Used to read OID fields as integer values and map these fields to TIntegerField.
PrefetchRows	Used to set the number of rows to be prefetched during the execution of a query.
PrepareUpdateSQL	Used to automatically prepare update queries before execution.
QueryRecCount (inherited from TDADatasetOptions)	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 (inherited from TDADatasetOptions)	Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh (inherited from TDADatasetOptions)	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields (inherited from TDADatasetOptions)	Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams (inherited from TDADatasetOptions)	Used to return the new value of fields to dataset after insert or update.
SetEmptyStrToNull	Force replace of empty strings with NULL values in data. The default value is False.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.14.1.9.2 Properties

Properties of the **TPgDataSetOptions** class.

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

Public

Name	Description
AutoPrepare (inherited from TDADatasetOptions)	Used to execute automatic TCustomDADataset.Prepare on the query execution.
CacheCalcFields (inherited from TDADatasetOptions)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode (inherited from TDADatasetOptions)	Used to store values of the BLOB fields in compressed form.
DefaultValues (inherited from TDADatasetOptions)	Used to request default values/expressions from the server and assign them to the DefaultExpression

	property.
DetailDelay (inherited from TDADatasetOptions)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin (inherited from TDADatasetOptions)	Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers (inherited from TDADatasetOptions)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields (inherited from TDADatasetOptions)	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail (inherited from TDADatasetOptions)	Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from TDADatasetOptions)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from TDADatasetOptions)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange (inherited from TDADatasetOptions)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount (inherited from TDADatasetOptions)	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 (inherited from TDADatasetOptions)	Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh (inherited from TDADatasetOptions)	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields (inherited from TDADatasetOptions)	Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams (inherited from TDADatasetOptions)	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.
UpdateAllFields (inherited from TDADatasetOptions)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize (inherited from TDADatasetOptions)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be

	executed in a batch.
--	----------------------

Published

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.
DistinctParams	Used for correct TClientDataSet parameters handling.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
ExtendedFieldsInfo	Used to perform an additional query to get information about the returned fields and the tables they belong to.
FullRefresh	Used to specify the fields to include in the automatically generated SQL statement when calling the method.
OIDAsInt	Used to read OID fields as integer values and map these fields to TIntegerField.
PrefetchRows	Used to set the number of rows to be prefetched during the execution of a query.

PrepareUpdateSQL	Used to automatically prepare update queries before execution.
SetEmptyStrToNull	Force replace of empty strings with NULL values in data. The default value is False.
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.1 AutoDeleteBlob Property

Used to delete large objects from the database automatically when a record is deleted from the dataset.

Class

[TPgDataSetOptions](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.2 CacheBlobs Property

Used to allocate local memory buffer to hold a copy of the large object content.

Class

[TPgDataSetOptions](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.3 CursorWithHold Property

Used to open query in the FetchAll=False mode without transaction.

Class

[TPgDataSetOptions](#)

Syntax

```
property cursorWithHold: boolean default False;
```

Remarks

Use the CursorWithHold option to open query in the FetchAll=False mode without transaction.

When this option is False (default), an active transaction is required to open a query in the FetchAll=False mode. If there is no active transaction, PgDAC opens additional internal connection and starts transaction on this connection.

When this option is True, PgDAC uses DECLARE CURSOR ... WITH HOLD statement to open the query. In this case no active transaction is required but this may take additional server resources.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.4 DeferredBlobRead Property

Used to fetch values of large objects when they are explicitly requested.

Class

[TPgDataSetOptions](#)

Syntax

```
property DeferredBlobRead: boolean default True;
```

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.5 DistinctParams Property

Used for correct TClientDataSet parameters handling.

Class

[TPgDataSetOptions](#)

Syntax

```
property DistinctParams: boolean default True;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.6 EnableBCD Property

Used to enable currency type. Default value of this option is False.

Class

[TPgDataSetOptions](#)

Syntax

```
property EnableBCD: boolean;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.7 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class

[TPgDataSetOptions](#)

Syntax

```
property EnableFMTBCD: boolean;
```

Remarks

Use the EnableFMTBCD property to enable using FMTBCD instead of float for large integer numbers to keep precision.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.8 ExtendedFieldsInfo Property

Used to perform an additional query to get information about the 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 the returned fields and the tables they belong to.

If True, an additional query is performed to get information about the returned fields and the tables they belong to. This information includes the NOT NULL attribute of the field, the SEQUENCE linked to the field, and the table name corresponding to the field. The table name is needed to detect fields that belong to the updated table and set the read-only attribute for all other fields returned by the query.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.9 FullRefresh Property

Used to specify the fields to include in the automatically generated SQL statement when calling the method.

Class

[TPgDataSetOptions](#)

Syntax

```
property FullRefresh: boolean;
```

Remarks

Use the FullRefresh property to specify what fields to include in the automatically generated

SQL statement when calling the [TCustomDADataset.RefreshRecord](#) method. If the FullRefresh property is True, all fields from a query are included into SQL statement to refresh a single record. If FullRefresh is False, only fields from [TPgQuery.UpdatingTable](#) are included.

Note: If FullRefresh is True, the refresh of SQL statement for complex queries and views may be generated with errors. The default value is False.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.10 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 True to read OIDs as integer values and map OID fields to TIntegerField. Set this option to False to read OIDs as large objects descriptors, and then read the values of the corresponding large objects. When OIDAsInt is False, OID fields are mapped on TBlobField.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.11 PrefetchRows Property

Used to set the number of rows to be prefetched during the execution of a query.

Class

[TPgDataSetOptions](#)

Syntax

```
property PrefetchRows: integer default 0;
```

Remarks

Use the PrefetchRows property to set the number of rows to be prefetched during the execution of a query. Setting the property to a value greater than 0 reduces the server round-trip count, which increases the performance of the application. The default value is 0 - the number of prefetched rows is determined automatically. To disable row prefetching, set the property to -1.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.12 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.13 SetEmptyStrToNull Property

Force replace of empty strings with NULL values in data. The default value is False.

Class

[TPgDataSetOptions](#)

Syntax

```
property SetEmptyStrToNull: boolean;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.14 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.

Memor is used because maximum length of values from such fields is unknown.

If True, fields of unknown data types are mapped to TStringField or TWideStringField depending on the value of the UseUnicode option. Size of fields is set to 8192. Values larger than this size are truncated.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.14.1.9.2.15 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.9.2.16 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.10 TPgDataSource Class

TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.

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

Unit

[PgAccess](#)

Syntax

```
TPgDataSource = class(TCRDataSource);
```

Remarks

TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.

TPgDataSource inherits its functionality directly from the TDataSource component.

At design-time assign individual data-aware components' DataSource properties from their drop-down listboxes.

Inheritance Hierarchy

[TCRDataSource](#)

TPgDataSource

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.10.1 Members

[TPgDataSource](#) class overview.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.11 TPgDateField Class

A class providing access to the PostgreSQL date fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgDateField = class(TCustomPgTimeStampField);
```

Remarks

The TPgDateField class provides access to the PostgreSQL date fields.

Inheritance Hierarchy

[TCustomPgTimeStampField](#)

TPgDateField

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.11.1 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.11.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.11.2.1 AsPgDate Property

Used to provide access to a TPgDate object.

Class

[TPgDateField](#)

Syntax

```
property ASPgDate: TPgDate;
```

Remarks

Use the ASPgDate property to provide access to a TPgDate object you can use for manipulations with the date value.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.12 TPgEncryptor Class

The class that performs encrypting and decrypting of data.

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

Unit

[PgAccess](#)

Syntax

```
TPgEncryptor = class(TCREncryptor);
```

Inheritance Hierarchy

[TCREncryptor](#)

TPgEncryptor

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.12.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.13 TPgGeometricField Class

A class providing access to the PostgreSQL geometric fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgGeometricField = class(TField);
```

Remarks

The TPgGeometricField class provides access to the PostgreSQL geometric fields.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.13.1 Members

[TPgGeometricField](#) class overview.

Properties

Name	Description
AsPgGeometric	Used to provide access to a TPgGeometric object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.13.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.13.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.14 TPgIntervalField Class

A class providing access to the PostgreSQL interval fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgIntervalField = class(TField);
```

Remarks

The TPgIntervalFields class provides access to the PostgreSQL interval fields.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.14.1 Members

[TPgIntervalField](#) class overview.

Properties

Name	Description
AsPgInterval	Used to provide access to a TPgInterval object.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.14.2 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 TPgInterval object.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.14.1.14.2.1 AsPgInterval Property

Used to provide access to a TPgInterval object.

Class

[TPgIntervalField](#)

Syntax

```
property ASPgInterval: TPgInterval;
```

Remarks

Use the ASPgInterval property to provide access to a TPgInterval object you can use for manipulations with the interval value.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.14.1.15 TPgLargeObject Class

A class providing support of PostgreSQL large objects.

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

Unit

[PgAccess](#)

Syntax

```
TPgLargeObject = class(TPgSQLLargeObject);
```

Remarks

The TPgLargeObject class provides support of PostgreSQL large objects.

Inheritance Hierarchy

[TSharedObject](#)

[TBlob](#)

[TCompressedBlob](#)

[TPgSQLLargeObject](#)

TPgLargeObject

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.15.1 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

Truncate (inherited from TBlob)	TBlob object to a stream. 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.15.2 Properties

Properties of the **TPgLargeObject** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.15.2.1 Connection Property

Connection used to read and write value of large object.

Class

[TPgLargeObject](#)

Syntax

```
property Connection: TPgConnection;
```

Remarks

If you call methods that read or write large object value from/to database, set the Connection property to an appropriate TPgConnection objects. Otherwise these methods will fail.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.16 TPgLargeObjectField Class

A class providing access to the PostgreSQL large object fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgLargeObjectField = class(TBlobField);
```

Remarks

The TPgLargeObjectFields class provides access to the PostgreSQL large object fields.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.16.1 Members

[TPgLargeObjectField](#) class overview.

Properties

Name	Description
AsLargeObject	Used to provide access to a TPgLargeObject object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.16.2 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](#)

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.16.2.1 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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.17 TPgMetaData Class

A component for obtaining metainformation about database objects from the server.

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

Unit

[PgAccess](#)

Syntax

```
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.17.1 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 NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from TMemDataSet)	Used to prevent implicit update of rows on database server.

MetaDataKind (inherited from TDAMetaData)	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 (inherited from TDAMetaData)	Used to provide one or more conditions restricting the list of objects to be described.
UpdateRecordTypes (inherited from TMemDataSet)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from TMemDataSet)	Used to check the status of the cached updates buffer.

Methods

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 (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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.18 TPgParam Class

A class that is used to set the values of individual parameters passed with queries or stored procedures.

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

Unit

[PgAccess](#)

Syntax

```
TPgParam = class (TDAParam);
```

Remarks

Use the properties of TPgParam to set the value of a parameter. Objects that use parameters create TPgParam objects to represent these parameters. For example, TPgParam objects are used by TPgSQL, TCustomPgDataSet.

TPgParam shares many properties with TField, as both describe the value of a field in a dataset. However, a TField object has several properties to describe the field binding, and how the field is displayed, edited, or calculated that are not needed in a TPgParam object. Conversely, TPgParam includes properties that indicate how the field value is passed as a parameter.

Inheritance Hierarchy

[TDAParam](#)

TPgParam

See Also

- [TCustomPgDataSet](#)
- [TPgSQL](#)
- [TPgParams](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.18.1 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 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.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.18.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.18.2.1 AsPgDate Property

Used to specify the parameter value when it represents the date type.

Class

[TPgParam](#)

Syntax

```
property ASPgDate: TPgDate;
```

Remarks

Use the ASPgDate property to specify the parameter value when it represents the date type.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.18.2.2 AsPgInterval Property

Used to specify the parameter value when it represents the INTERVAL type.

Class

[TPgParam](#)

Syntax

```
property ASPgInterval: TPgInterval;
```

Remarks

Use the ASPgInterval property to specify the parameter value when it represents the INTERVAL type.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.18.2.3 ASPgTime Property

Used to specify the parameter value when it represents the time type.

Class

[TPgParam](#)

Syntax

```
property ASPgTime: TPgTime;
```

Remarks

Use the ASPgTime property to specify the parameter value when it represents the time type.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.18.2.4 ASPgTimeStamp Property

Used to specify the parameter value when it represents the TimeStamp type.

Class

[TPgParam](#)

Syntax

```
property ASPgTimeStamp: TPgTimeStamp;
```

Remarks

Use the `AsPgTimeStamp` property to specify the parameter value when it represents the `TimeStamp` type.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.19 TPgParams Class

Used to control `TPgParam` objects.

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

Unit

[PgAccess](#)

Syntax

```
TPgParams = class(TDAParams);
```

Remarks

Use `TPgParams` to manage a list of `TPgParam` objects for an object that uses field parameters. For example, `TPgStoredProc` objects and `TPgQuery` objects use `TPgParams` objects to create and access their parameters.

Inheritance Hierarchy

[TDAParams](#)

TPgParams

See Also

- [TPgParam](#)
- [TCustomDASQL.Params](#)
- [TCustomDADataset.Params](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.19.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.19.2 Properties

Properties of the **TPgParams** class.

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

Public

Name	Description
Items	Used to iterate through all field parameters.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.19.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.20 TPgQuery Class

A component for executing queries and operating record sets. It also provides flexible way to update data.

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

Unit

[PgAccess](#)

Syntax

```
TPgQuery = class (TCustomPgDataSet);
```

Remarks

TPgQuery is a direct descendant of the [TCustomPgDataSet](#) component. It publishes most of

its inherited properties and events so that they can be manipulated at design-time.

Use TPgQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. TPgQuery provides automatic blocking of records, their checking before edit and refreshing after post. Set SQL, SQLInsert, SQLDelete, SQLRefresh, and SQLUpdate properties to define SQL statements for subsequent accesses to the database server. There is no restriction to their syntax, so any SQL statement is allowed. Usually you need to use INSERT, DELETE, and UPDATE statements but you also may use stored procedures in more diverse cases.

To modify records, you can specify KeyFields. If they are not specified, TPgQuery will retrieve primary keys for UpdatingTable from metadata. TPgQuery can automatically update only one table. Updating table is defined by the UpdatingTable property if this property is set. Otherwise, the table a field of which is the first field in the field list in the SELECT clause is used as an updating table.

The SQLInsert, SQLDelete, SQLUpdate, SQLRefresh properties support automatic binding of parameters which have identical names to fields captions. To retrieve the value of a field as it was before the operation use the field name with the 'OLD_' prefix. This is especially useful when doing field comparisons in the WHERE clause of the statement. Use the [TCustomDADataset.BeforeUpdateExecute](#) event to assign the value to additional parameters and the [TCustomDADataset.AfterUpdateExecute](#) event to read them.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

[TCustomPgDataSet](#)

TPgQuery

See Also

- [Updating Data with PgDAC Dataset Components](#)
- [Master/Detail Relationships](#)
- [TPgStoredProc](#)
- [TPgTable](#)

Reserved.

5.14.1.20.1 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 TCustomDADataset)	Used to set data type mapping rules
Debug (inherited from TCustomDADataset)	Used to display the statement that is being executed and the values and types of its 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 TCustomPgDataSet)	Used to refresh record by RETURNING clause when insert or update is performed.
FetchAll	Defines whether to request all records of the query from database server when the

	dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataSet)	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.
LastInsertOID (inherited from TCustomPgDataSet)	Returns OID of the record inserted by the last query for table with OIDs.
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.

LockMode	Used to specify what kind of lock will be performed when editing a record.
MacroCount (inherited from TCustomDADataset)	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.
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 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.
UpdatingTable	Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Methods

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.

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 find out whether TCustomDADataset has 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)	Finds a macro with the specified name.
FindNearest (inherited from TCustomDADataset)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from TCustomPgDataSet)	Searches for and returns a parameter with the specified name.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

GetDataType (inherited from TCustomDADataset)	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject (inherited from TCustomDADataset)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from TCustomDADataset)	Retrieves the precision of a number field.
GetFieldScale (inherited from TCustomDADataset)	Retrieves the scale of a number field.
GetKeyFieldNames (inherited from TCustomDADataset)	Provides a list of available key field names.
GetOrderBy (inherited from TCustomDADataset)	Retrieves an ORDER BY clause from a SQL statement.
GetPgCursor (inherited from TCustomPgDataset)	Retrieves a TPgCursor object for a field with known name.
GetPgDate (inherited from TCustomPgDataset)	Retrieves a TPgDate object for a field with known name.
GetPgInterval (inherited from TCustomPgDataset)	Retrieves a TPgInterval object for a field with known name.
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 TCustomDADataset)	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 TCustomDADataset)	Locks the current record.
MacroByName (inherited from TCustomDADataset)	Finds a macro with the specified 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 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.20.2 Properties

Properties of the **TPgQuery** class.

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

Public

Name	Description
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 the statement that is being executed and the values and types of its 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 TCustomPgDataset)	Used to refresh record by RETURNING clause when insert or update is performed.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataset)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataset)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataset)	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.
LastInsertOID (inherited from TCustomPgDataset)	Returns OID of the record inserted by the last query for table with OIDs.
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 (inherited from TCustomDADataset)	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.
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 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
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
LockMode	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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.20.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

[TPgQuery](#)

Syntax

```
property FetchAll: boolean;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

Opening a dataset in FetchAll = False mode requires an active transaction. When the FetchAll property is False, the first call to [TMemDataSet.Locate](#) and [TMemDataSet.LocateEx](#) methods may take a lot of time to retrieve additional records to the client side.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.20.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

[TPgQuery](#)

Syntax

```
property LockMode: TLockMode;
```

Remarks

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is ImNone.

See Also

- [TPgStoredProc.LockMode](#)
- [TPgTable.LockMode](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.20.2.3 UpdatingTable Property

Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Class

[TPgQuery](#)

Syntax

```
property UpdatingTable: string;
```

Remarks

Use the UpdatingTable property to specify which table in a query is assumed to be the target for the subsequent data-modification queries as a result of user incentive to insert, update or delete records.

This property is used on Insert, Update, Delete or RefreshRecord (see also [TCustomPgDataSet.Options](#)) if appropriate SQL (SQLInsert, SQLUpdate or SQLDelete) is not provided.

If UpdatingTable is not set then the first table used in a query is assumed to be the target.

Example

Below are two examples for the query, where:

1. the only allowed value for UpdatingTable property is 'Dept';
2. allowed values for UpdatingTable are 'Dept' and 'Emp'.

In the first case (or by default) editable field is ShipName, in the second - all fields from Emp.

```
1) Example 1.  
   SELECT * FROM Dept  
2) Example 2.  
   SELECT * FROM Dept, Emp  
   WHERE Dept.DeptNo = Emp.DeptNo
```

© 1997-2024

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.14.1.21 TPgSQL Class

A component for executing SQL statements and calling stored procedures on the database server.

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

Unit

[PgAccess](#)

Syntax

```
TPgSQL = class(TCustomDASQL);
```

Remarks

The TPgSQL component is a direct descendant of the [TCustomDASQL](#) class.

Use The TPgSQL component when a client application must execute SQL statement or the PL/SQL block, and call stored procedure on the database server. The SQL statement should not retrieve rows from the database.

Inheritance Hierarchy

[TCustomDASQL](#)

TPgSQL

See Also

- [TPgQuery](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.1 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.
Debug (inherited from TCustomDASQL)	Used to display the statement that is being executed and the values and types of its parameters.
FinalSQL (inherited from TCustomDASQL)	Used to return a SQL statement with expanded macros.
LastInsertOID	Returns OID of the record inserted by the last query for table with OIDs.
MacroCount (inherited from TCustomDASQL)	Used to get the number of macros associated with the Macros property.
Macros (inherited from TCustomDASQL)	Makes it possible to change SQL queries easily.
ParamCheck (inherited from TCustomDASQL)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
ParamCount (inherited from TCustomDASQL)	Indicates the number of parameters in the Params property.
Params	Contains parameters for a query's SQL statement.
ParamValues (inherited from TCustomDASQL)	Used to get or set the values of individual field parameters that are identified by name.
Prepared (inherited from TCustomDASQL)	Used to indicate whether a query is prepared for execution.
RowsAffected (inherited from TCustomDASQL)	Used to indicate the number of rows which were inserted,

	updated, or deleted during the last query operation.
SQL (inherited from TCustomDASQL)	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.
UnpreparedExecute	Used to apply simple executing to a SQL statement.
UseParamTypes	Used to disable automatic detection of the parameters data types.

Methods

Name	Description
BreakExec (inherited from TCustomDASQL)	Breaks execution of an SQL statement on the server.
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)	Finds 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 specified 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.2 Properties

Properties of the **TPgSQL** class.

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

Public

Name	Description
ChangeCursor (inherited from TCustomDASQL)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
Debug (inherited from TCustomDASQL)	Used to display the statement that is being executed and the values and types of its parameters.
FinalSQL (inherited from TCustomDASQL)	Used to return a SQL statement with expanded macros.
LastInsertOID	Returns OID of the record inserted by the last query for table with OIDs.
MacroCount (inherited from TCustomDASQL)	Used to get the number of macros associated with the Macros property.
Macros (inherited from TCustomDASQL)	Makes it possible to change SQL queries easily.
ParamCheck (inherited from TCustomDASQL)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

ParamCount (inherited from TCustomDASQL)	Indicates the number of parameters in the Params property.
ParamValues (inherited from TCustomDASQL)	Used to get or set the values of individual field parameters that are identified by name.
Prepared (inherited from TCustomDASQL)	Used to indicate whether a query is prepared for execution.
RowsAffected (inherited from TCustomDASQL)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SQL (inherited from TCustomDASQL)	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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.2.1 CommandTimeout Property

The time to wait for a statement to be executed.

Class

[TPgSQL](#)

Syntax

```
property CommandTimeout: integer default 0;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.2.2 Connection Property

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

Class

[TPgSQL](#)

Syntax

```
property Connection: TPgConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TPgConnection objects.

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

See Also

- [TPgConnection](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.14.1.21.2.3 LastInsertOID Property

Returns OID of the record inserted by the last query for table with OIDs.

Class

[TPgSQL](#)

Syntax

```
property LastInsertOID: Int64;
```

Remarks

Use the LastInsertOID property to get OID of the record inserted by the last query for table with OIDs.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.2.4 Params Property

Contains parameters for a query's SQL statement.

Class

[TPgSQL](#)

Syntax

```
property Params: TPgParams stored False;
```

Remarks

Contains parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information).

Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is

known is to call ParamByName.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.2.5 UnpreparedExecute Property

Used to apply simple executing to a SQL statement.

Class

[TPgSQL](#)

Syntax

```
property UnpreparedExecute: boolean default False;
```

Remarks

If the UnpreparedExecute property is set to True, the simple execute is used for SQL statement. Statement is not prepared before execute. It allows to add multiple statements separated by semicolon to the SQL property.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.2.6 UseParamTypes Property

Used to disable automatic detection of the parameters data types.

Class

[TPgSQL](#)

Syntax

```
property UseParamTypes: boolean default False;
```

Remarks

Set the UseParamTypes option to True to disable automatic detection of parameter types. When this option is True, data types of parameters are set basing on the DataType property. When this option is False, data types of the parameters are detected by server automatically.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.3 Methods

Methods of the **TPgSQL** class.

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

Public

Name	Description
BreakExec (inherited from TCustomDASQL)	Breaks execution of an SQL statement on the server.
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)	Finds 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 specified 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.3.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.21.3.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.22 TPgStoredProc Class

A component for accessing and executing stored procedures and functions.

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

Unit

[PgAccess](#)

Syntax

```
TPgStoredProc = class(TCustomPgStoredProc);
```

Remarks

Use TPgStoredProc to access stored procedures on the database server.

You need only to define the StoredProcName property, and the SQL statement to call the stored procedure will be generated automatically.

Use the Execute method at runtime to generate request that instructs server to execute procedure and PrepareSQL to describe parameters at run time

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

[TCustomPgDataSet](#)

[TCustomPgStoredProc](#)

TPgStoredProc

See Also

- [TPgQuery](#)
- [TPgSQL](#)
- [Updating Data with PgDAC Dataset Components](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.22.1 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 TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
CommandTimeout	The time to wait for a statement to be executed.
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 the statement that is being executed and the values and types of its 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 TCustomPgDataSet)	Used to refresh record by RETURNING clause when insert or update is performed.
FetchAll (inherited from TCustomPgDataSet)	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and

	lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataset)	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.
LastInsertOID (inherited from TCustomPgDataset)	Returns OID of the record inserted by the last query for table with OIDs.
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.
LockMode	Used to specify what kind of lock will be performed when editing a record.
MacroCount (inherited from TCustomDADataset)	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.
Options (inherited from TCustomPgDataset)	Used to specify the behaviour of

	TCustomPgDataSet object.
Overload (inherited from TCustomPgStoredProc)	Used to specify the overloading number.
ParamCheck (inherited from TCustomDADDataSet)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from TCustomDADDataSet)	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 TCustomDADDataSet)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions (inherited from TCustomDADDataSet)	Used to indicate when the editing record is refreshed.
RowsAffected (inherited from TCustomDADDataSet)	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 TCustomDADDataSet)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from TCustomDADDataSet)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from TCustomDADDataSet)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from TCustomDADDataSet)	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.
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.
ExecProc (inherited from TCustomPgStoredProc)	Executes a SQL statement on the server.
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 find out whether TCustomDADataset has 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)	Finds a macro with the specified name.
FindNearest (inherited from TCustomDADataset)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from TCustomPgDataset)	Searches for and returns a parameter with the specified name.
GetBlob (inherited from TMemDataset)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from TCustomDADataset)	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject (inherited from TCustomDADataset)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from TCustomDADataset)	Retrieves the precision of a number field.
GetFieldScale (inherited from TCustomDADataset)	Retrieves the scale of a number field.
GetKeyFieldNames (inherited from TCustomDADataset)	Provides a list of available key field names.
GetOrderBy (inherited from TCustomDADataset)	Retrieves an ORDER BY clause from a SQL statement.
GetPgCursor (inherited from TCustomPgDataset)	Retrieves a TPgCursor object for a field with known name.

GetPgDate (inherited from TCustomPgDataSet)	Retrieves a TPgDate object for a field with known name.
GetPgInterval (inherited from TCustomPgDataSet)	Retrieves a TPgInterval object for a field with known name.
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 TCustomDADataset)	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 TCustomDADataset)	Locks the current record.
MacroByName (inherited from TCustomDADataset)	Finds a macro with the specified 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 TCustomDADataset)	Allocates, opens, and parses cursor for a query.
PrepareSQL (inherited from TCustomPgStoredProc)	Describes the parameters of a stored procedure.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.22.2 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 the statement that is being executed and the values and types of its 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 TCustomPgDataset)	Used to refresh record by RETURNING clause when insert or update is performed.
FetchAll (inherited from TCustomPgDataset)	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.

FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataSet)	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.
LastInsertOID (inherited from TCustomPgDataSet)	Returns OID of the record inserted by the last query for table with OIDs.
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 (inherited from TCustomDADataset)	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.
Options (inherited from TCustomPgDataset)	Used to specify the behaviour of TCustomPgDataset object.
Overload (inherited from TCustomPgStoredProc)	Used to specify the overloading number.
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 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.22.2.1 CommandTimeout Property

The time to wait for a statement to be executed.

Class

[TPgStoredProc](#)

Syntax

```
property CommandTimeout: integer;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.22.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.22.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.23 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.23.1 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.
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 the statement that is being executed and the values and types of its 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 TCustomPgDataset)	Used to refresh record by RETURNING clause when insert or update is performed.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.

FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataSet)	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.
LastInsertOID (inherited from TCustomPgDataSet)	Returns OID of the record inserted by the last query for table with OIDs.
Limit (inherited from TCustomPgTable)	Used to set the number of rows retrieved from the query.
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.
LockMode	Used to specify what kind of lock will be performed when editing a record.

MacroCount (inherited from TCustomDADataset)	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 (inherited from TCustomPgTable)	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.
OrderFields	Used to build ORDER BY clause of SQL statements.
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 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.
TableName	Used to specify the name of the database table this component encapsulates.
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.
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 find out whether TCustomDADataset has 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)	Finds a macro with the specified name.
FindNearest (inherited from TCustomDADataset)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from TCustomPgDataset)	Searches for and returns a parameter with the specified name.
GetBlob (inherited from TMemDataSet)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from TCustomDADataset)	Returns internal field types defined in the MemData and

	accompanying modules.
GetFieldObject (inherited from TCustomDADataset)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from TCustomDADataset)	Retrieves the precision of a number field.
GetFieldScale (inherited from TCustomDADataset)	Retrieves the scale of a number field.
GetKeyFieldNames (inherited from TCustomDADataset)	Provides a list of available key field names.
GetOrderBy (inherited from TCustomDADataset)	Retrieves an ORDER BY clause from a SQL statement.
GetPgCursor (inherited from TCustomPgDataset)	Retrieves a TPgCursor object for a field with known name.
GetPgDate (inherited from TCustomPgDataset)	Retrieves a TPgDate object for a field with known name.
GetPgInterval (inherited from TCustomPgDataset)	Retrieves a TPgInterval object for a field with known name.
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 TCustomDADataset)	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 TCustomDADataset)	Locks the current record.
MacroByName (inherited from TCustomDADataset)	Finds a macro with the specified 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 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.14.1.23.2 Properties

Properties of the **TPgTable** class.

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

Public

Name	Description
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 the statement that is being executed and the values and types of its 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 TCustomPgDataSet)	Used to refresh record by RETURNING clause when insert or update is performed.
FetchRows (inherited from TCustomDADataset)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from TCustomDADataset)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from TCustomDADataset)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from TMemDataSet)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery (inherited from TCustomDADataset)	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from TMemDataSet)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from TCustomDADataset)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
KeySequence (inherited from TCustomPgDataSet)	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.
LastInsertOID (inherited from TCustomPgDataSet)	Returns OID of the record inserted by the last query for table with OIDs.
Limit (inherited from TCustomPgTable)	Used to set the number of rows retrieved from the query.
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 (inherited from TCustomDADataset)	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 (inherited from TCustomPgTable)	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 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
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
LockMode	Used to specify what kind of lock will be performed when editing a record.
OrderFields	Used to build ORDER BY clause of SQL statements.
TableName	Used to specify the name of the database table this component encapsulates.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.23.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

[TPgTable](#)

Syntax

```
property FetchAll: boolean;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

Opening a dataset in FetchAll = False mode requires an active transaction. When the FetchAll property is False, the first call to [TMemDataSet.Locate](#) and [TMemDataSet.LocateEx](#) methods may take a lot of time to retrieve additional records to the client side.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.23.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.23.2.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.23.2.4 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 [TCustomDADataset.Connection](#) is assigned at design time, select a valid table name from the `TableName` drop-down list in Object Inspector.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.24 TPgTimeField Class

A class providing access to the PostgreSQL time fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgTimeField = class(TCustomPgTimeStampField);
```

Remarks

The `TPgTimeFields` class provides access to the PostgreSQL time fields.

Inheritance Hierarchy

[TCustomPgTimeStampField](#)

TPgTimeField

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.24.1 Members

[TPgTimeField](#) class overview.

Properties

Name	Description
AsPgTime	Used to provide access to a <code>TPgTime</code> object.
AsPgTimeStamp (inherited from	Used to provide access to a

[TCustomPgTimeStampField](#))

TPgTimeStamp object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.14.1.24.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.14.1.24.2.1 AsPgTime Property

Used to provide access to a TPgTime object.

Class

[TPgTimeField](#)

Syntax

```
property ASPgTime: TPgTime;
```

Remarks

Use the `AsPgTime` property to provide access to a `TPgTime` object you can use for manipulations with the time value.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.25 TPgTimeStampField Class

A class providing access to the PostgreSQL timestamp fields.

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

Unit

[PgAccess](#)

Syntax

```
TPgTimeStampField = class(TCustomPgTimeStampField);
```

Remarks

`TPgTimeStampField` provides access to PostgreSQL timestamp fields.

Inheritance Hierarchy

[TCustomPgTimeStampField](#)

TPgTimeStampField

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.25.1 Members

[TPgTimeStampField](#) class overview.

Properties

Name	Description
AsPgTimeStamp	Used to provide access to a <code>TPgTimeStamp</code> object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.14.1.25.2 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](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.25.2.1 AsPgTimeStamp Property

Used to provide access to a TPgTimeStamp object.

Class

[TPgTimeStampField](#)

Syntax

```
property ASPgTimeStamp: TPgTimeStamp;
```

Remarks

Use the AsTimeStamp property to get access to a TPgTimeStamp object which you can use for manipulations with the timestamp value.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.26 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.1.26.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.3 Enumerations

Enumerations in the **PgAccess** unit.

Enumerations

Name	Description
TPgIsolationLevel	Specifies the way the transactions containing database modifications are handled.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.4 Constants

Constants in the **PgAccess** unit.

Constants

Name	Description
PgDACVersion	The version of PostgreSQL Data Access Components.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.14.4.1 PgDACVersion Constant

The version of PostgreSQL Data Access Components.

Unit

[PgAccess](#)

Syntax

```
PgDACVersion = '8.1.0';
```

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.15 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.15.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.15.1.1 TPgAlerter Class

A component that allows to register interest in and asynchronously handle event notifications posted by PostgreSQL server.

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

Unit

[PgAlerter](#)

Syntax

```
TPgAlerter = class(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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.15.1.1.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.15.1.1.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.15.1.1.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.15.1.1.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.15.1.1.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.15.1.1.4 Events

Events of the **TPgAlerter** class.

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

Public

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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.15.1.1.4.1 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.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.16 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 protocol version 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.16.1 Classes

Classes in the **PgClasses** unit.

Classes

Name	Description
TPgCursor	A class for working with PostgreSQL cursors.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.16.1.1 TPgCursor Class

A class for working with PostgreSQL cursors.

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

Unit

[PgClasses](#)

Syntax

```
TPgCursor = class(TCRCursor);
```

Remarks

The TPgCursor class is used to work with PostgreSQL cursors. This is a PgDAC internal class, and application should not use it directly.

Inheritance Hierarchy

[TSharedObject](#)

[TCRCursor](#)

TPgCursor

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.16.1.1.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.16.1.1.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.16.1.1.2.1 State Property

Used to set the cursor state.

Class

[TPgCursor](#)

Syntax

```
property State: TCursorState;
```

See Also

- [TCursorState](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.16.2 Enumerations

Enumerations in the **PgClasses** unit.

Enumerations

Name	Description
TProtocolVersion	Specifies protocol version 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.16.2.1 TProtocolVersion Enumeration

Specifies protocol version to be used when several versions are available.

Unit

[PgClasses](#)

Syntax

```
TProtocolVersion = (pv20, pv30, pvAuto);
```

Values

Value	Meaning
pv20	Set ProtocolVersion to pv20 to work with PostgreSQL server

	version 7.3 or older that don't support protocol version 3.0.
pv30	Set ProtocolVersion to pv30 to enforce protocol version 3.0.
pvAuto	Set ProtocolVersion to pvAuto to automatically select between protocol versions depending on the specific query for the best possible performance.

Remarks

The default value is pvAuto.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.16.2.2 TSSLMode Enumeration

This option determines whether or with what priority an SSL connection will be negotiated with the server.

Unit

[PgClasses](#)

Syntax

```
TSSLMode = (smDisable, smRequire, smPrefer, smAllow, smVerifyCA, smVerifyFull);
```

Values

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.
smVerifyCA	Verifies server identity by validating the server certificate chain up to the root certificate installed on the client machine.
smVerifyFull	Verifies server identity by validating the server certificate chain up to the root certificate installed on the client machine and validates that the server hostname matches the server certificate.

Remarks

The default value is smDisable.

If PostgreSQL is compiled without SSL support, using option smRequire will cause an error, while options smAllow and smPrefer will be accepted, but PgDAC will not in fact attempt an SSL connection.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.17 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.17.1 Classes

Classes in the **PgConnectionPool** unit.

Classes

Name	Description
TPgConnectionPoolManager	A class of methods that are used for managing PgDAC connection pool.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.17.1.1 TPgConnectionPoolManager Class

A class of methods that are used for managing PgDAC connection pool.

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

Unit

[PgConnectionPool](#)

Syntax

```
TPgConnectionPoolManager = class(TCRConnectionPoolManager);
```

Remarks

Use the TPgConnectionPoolManager methods to manage PgDAC connection pool.

Inheritance Hierarchy

TCRConnectionPoolManager

TPgConnectionPoolManager

See Also

- [Connection Pooling](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.17.1.1.1 Members

[TPgConnectionPoolManager](#) class overview.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18 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.
----------------------------------	--

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18.1 Classes

Classes in the **PgDacVcl** unit.

Classes

Name	Description
TPgConnectDialog	A component providing a dialog box for user to supply his login information.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18.1.1 TPgConnectDialog Class

A component providing a dialog box for user to supply his login information.

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

Unit

[PgDacVcl](#)

Syntax

```
TPgConnectDialog = class(TCustomConnectDialog);
```

Remarks

The TPgConnectDialog class provides a dialog box for user to supply his login information.

The TPgConnectDialog component is a direct descendant of the TCustomConnectDialog class. Use TPgConnectDialog to provide dialog box for a user to supply user name, password, server, port, and database name. You may want to customize appearance of the dialog box using properties of this class.

Inheritance Hierarchy

[TCustomConnectDialog](#)

TPgConnectDialog

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18.1.1.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18.1.1.2 Properties

Properties of the **TPgConnectDialog** class.

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

Public

Name	Description
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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18.1.1.2.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18.1.1.2.2 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](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18.1.1.2.3 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](#)

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18.1.1.2.4 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.18.1.1.2.5 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19 PgDataTypeMap

This unit contains the implementation of mapping between PostgreSQL and Delphi data types.

Constants

Name	Description
pgBigInt	Used to map bigint to Delphi data types.
pgBigSerial	Used to map bigserial to Delphi data types.
pgBit	Used to map bit to Delphi data types.
pgBitVarying	Used to map bit varying to Delphi data types.
pgBoolean	Used to map boolean to Delphi data types.
pgBytea	Used to map bytea to Delphi data types.
pgCharacter	Used to map character to Delphi data types.
pgCharacterVarying	Used to map character varying to Delphi data types.
pgDate	Used to map date to Delphi data types.
pgDoublePrecision	Used to map double precision to Delphi data types.
pgInteger	Used to map integer to Delphi data types.
pgMoney	Used to map money to Delphi data types.
pgNumeric	Used to map numeric to Delphi data types.
pgReal	Used to map real to Delphi data types.
pgSerial	Used to map serial to Delphi data types.
pgSmallint	Used to map smallint to Delphi data types.

pgText	Used to map text to Delphi data types.
pgTime	Used to map time to Delphi data types.
pgTimeStamp	Used to map timestamp to Delphi data types.
pgTimeStampWithTimeZone	Used to map timestamp with time zone to Delphi data types.
pgTimeWithTimeZone	Used to map time with time zone to Delphi data types.
pgUUID	Used to map uuid to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1 Constants

Constants in the **PgDataTypeMap** unit.

Constants

Name	Description
pgBigInt	Used to map bigint to Delphi data types.
pgBigSerial	Used to map bigserial to Delphi data types.
pgBit	Used to map bit to Delphi data types.
pgBitVarying	Used to map bit varying to Delphi data types.
pgBoolean	Used to map boolean to Delphi data types.
pgBytea	Used to map bytea to Delphi data types.
pgCharacter	Used to map character to Delphi data types.
pgCharacterVarying	Used to map character varying to Delphi data types.
pgDate	Used to map date to Delphi

	data types.
pgDoublePrecision	Used to map double precision to Delphi data types.
pgInteger	Used to map integer to Delphi data types.
pgMoney	Used to map money to Delphi data types.
pgNumeric	Used to map numeric to Delphi data types.
pgReal	Used to map real to Delphi data types.
pgSerial	Used to map serial to Delphi data types.
pgSmallint	Used to map smallint to Delphi data types.
pgText	Used to map text to Delphi data types.
pgTime	Used to map time to Delphi data types.
pgTimeStamp	Used to map timestamp to Delphi data types.
pgTimeStampWithTimeZone	Used to map timestamp with time zone to Delphi data types.
pgTimeWithTimeZone	Used to map time with time zone to Delphi data types.
pgUUID	Used to map uuid to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.1 pgBigInt Constant

Used to map **bigint** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgBigInt = pgBase + 1;
```

Remarks

Use the **pgBigInt** constant to map the PostgreSQL **bigint** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.2 pgBigSerial Constant

Used to map **bigserial** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgBigSerial = pgBase + 2;
```

Remarks

Use the **pgBigSerial** constant to map the PostgreSQL **bigserial** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.3 pgBit Constant

Used to map **bit** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgBit = pgBase + 51;
```

Remarks

Use the **pgBit** constant to map the PostgreSQL **bit** data type to Delphi data types.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.4 pgBitVarying Constant

Used to map **bit varying** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgBitVarying = pgBase + 52;
```

Remarks

Use the **pgBitVarying** constant to map the PostgreSQL **bit varying** data type to Delphi data types.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.5 pgBoolean Constant

Used to map **boolean** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgBoolean = pgBase + 5;
```

Remarks

Use the **pgBoolean** constant to map the PostgreSQL **boolean** data type to Delphi data types.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.6 pgBytea Constant

Used to map **bytea** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgBytea = pgBase + 6;
```

Remarks

Use the **pgBytea** constant to map the PostgreSQL **bytea** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.7 pgCharacter Constant

Used to map **character** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgCharacter = pgBase + 7;
```

Remarks

Use the **pgCharacter** constant to map the PostgreSQL **character** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.8 pgCharacterVarying Constant

Used to map **character varying** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgCharacterVarying = pgBase + 8;
```

Remarks

Use the **pgCharacterVarying** constant to map the PostgreSQL **character varying** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.9 pgDate Constant

Used to map **date** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgDate = pgBase + 9;
```

Remarks

Use the **pgDate** constant to map the PostgreSQL **date** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.10 pgDoublePrecision Constant

Used to map **double precision** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgDoublePrecision = pgBase + 10;
```

Remarks

Use the **pgDoublePrecision** constant to map the PostgreSQL **double precision** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.11 pgInteger Constant

Used to map **integer** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgInteger = pgBase + 11;
```

Remarks

Use the **pgInteger** constant to map the PostgreSQL **integer** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.12 pgMoney Constant

Used to map **money** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgMoney = pgBase + 12;
```

Remarks

Use the **pgMoney** constant to map the PostgreSQL **money** data type to Delphi data types.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.19.1.13 pgNumeric Constant

Used to map **numeric** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgNumeric = pgBase + 13;
```

Remarks

Use the **pgNumeric** constant to map the PostgreSQL **numeric** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.14 pgReal Constant

Used to map **real** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgReal = pgBase + 14;
```

Remarks

Use the **pgReal** constant to map the PostgreSQL **real** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.15 pgSerial Constant

Used to map **serial** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgSerial = pgBase + 16;
```

Remarks

Use the **pgSerial** constant to map the PostgreSQL **serial** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.16 pgSmallint Constant

Used to map **smallint** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgSmallint = pgBase + 15;
```

Remarks

Use the **pgSmallint** constant to map the PostgreSQL **smallint** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.17 pgText Constant

Used to map **text** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgText = pgBase + 17;
```

Remarks

Use the **pgText** constant to map the PostgreSQL **text** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.18 pgTime Constant

Used to map **time** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgTime = pgBase + 18;
```

Remarks

Use the **pgTime** constant to map the PostgreSQL **time without time zone** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.19 pgTimeStamp Constant

Used to map **timestamp** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgTimeStamp = pgBase + 20;
```

Remarks

Use the **pgTimeStamp** constant to map the PostgreSQL **timestamp without time zone** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.20 pgTimeStampWithTimeZone Constant

Used to map **timestamp with time zone** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgTimeStampWithTimeZone = pgBase + 21;
```

Remarks

Use the **pgTimeStampWithTimeZone** constant to map the PostgreSQL **timestamp with time zone** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.21 pgTimeWithTimeZone Constant

Used to map **time with time zone** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgTimeWithTimeZone = pgBase + 19;
```

Remarks

Use the **pgTimeWithTimeZone** constant to map the PostgreSQL **time with time zone** data

type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.19.1.22 pgUUID Constant

Used to map **uuid** to Delphi data types.

Unit

[PgDataTypeMap](#)

Syntax

```
pgUUID = pgBase + 23;
```

Remarks

Use the **pgUUID** constant to map the PostgreSQL **uuid** data type to Delphi data types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.1 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.1.1 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 the statement that is being executed and the values and types of its parameters.

Mode	Used to specify the mode of backup performed by TPgDump.
ObjectTypes	Used to specify the types of objects that are 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.

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.Ba

	ckupToStream(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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.1.2 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 the statement that is being executed and the values and types of its parameters.
Mode	Used to specify the mode of backup performed by TPgDump.
ObjectTypes	Used to specify the types of objects that are 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.1.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.1.2.2 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.1.2.3 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 constrains to the dump.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.1.2.4 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 the property is empty, the Backup method will dump all available schemas.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.2 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.2.1 Members

[TPgDumpOptions](#) class overview.

Properties

Name	Description
AddDrop (inherited from TDADumpOptions)	Used to add drop statements to a script before creating statements.
CompleteInsert (inherited from TDADumpOptions)	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
CreateConstraints	Used to add statements to create table constrains 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.2.2 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.
CompleteInsert (inherited from TDADumpOptions)	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is

	False.
CreateConstraints	Used to add statements to create table constrains 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.1.2.2.1 CreateConstraints Property

Used to add statements to create table constrains to the dump.

Class

[TPgDumpOptions](#)

Syntax

```
property CreateConstraints: boolean default True;
```

Remarks

If True (the default value), statements to create table constrains are added to the dump. If False, tables are created without constraints.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.2 Types

Types in the **PgDump** unit.

Types

Name	Description
TPgDumpObjects	Represents the set of TPgDumpObject .

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.2.1 TPgDumpObjects Set

Represents the set of [TPgDumpObject](#).

Unit

[PgDump](#)

Syntax

```
TPgDumpObjects = set of TPgDumpObject;
```

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.3 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 .

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.20.3.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1 Classes

Classes in the **PgError** unit.

Classes

Name	Description
EPgError	Represents a PostgreSQL notice, warning, and error

messages.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.21.1.1 EPgError Class

Represents a PostgreSQL notice, warning, and error messages.

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

Unit

[PgError](#)

Syntax

```
EPgError = class(EDAError);
```

Remarks

The EPgError object contains information that the server returns in the form of error, warning, log, notice etc. messages.

Inheritance Hierarchy

[EDAError](#)

EPgError

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.21.1.1.1 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 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2.2 DetailMsg Property

Holds a secondary error message.

Class

[EPgError](#)

Syntax

```
property DetailMsg: string;
```

Remarks

The DetailMsg property holds a secondary error message.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2.3 ErrorCode Property

Holds the SQLSTATE code.

Class

[EPgError](#)

Syntax

```
property ErrorCode: string;
```

Remarks

The ErrorCode property holds the SQLSTATE code.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2.4 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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2.5 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.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2.6 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2.7 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2.8 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.1.1.2.9 Severity Property

Holds the [TPgSeverity](#) value.

Class

[EPgError](#)

Syntax

```
property Severity: TPgSeverity;
```

Remarks

The Severity property holds the message severity value.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.2 Enumerations

Enumerations in the **PgError** unit.

Enumerations

Name	Description
TPgSeverity	Specifies the severity of the event occurred.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.21.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22 PgLoader

This unit contains the implementation of the TPgLoader component.

Classes

Name	Description
TPgLoader	This component serves for loading external data into the database table.
TPgLoaderColumn	A base class holding a collection of TPgLoaderColumn objects.

TPgLoaderOptions	This class allows setting up the behaviour of the TPgLoader component.
----------------------------------	--

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1 Classes

Classes in the **PgLoader** unit.

Classes

Name	Description
TPgLoader	This component serves for loading external data into the database table.
TPgLoaderColumn	A base class holding a collection of TPgLoaderColumn objects.
TPgLoaderOptions	This class allows setting up the behaviour of the TPgLoader component.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1 TPgLoader Class

This component serves for loading external data into the database table.

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

Unit

[PgLoader](#)

Syntax

```
TPgLoader = class(TDALoader);
```

Remarks

TPgLoader allows to load external data into the PostgreSQL database. It uses the COPY

command to load data. To specify the name of the loading table set the `TableName` property. Use the `Columns` property to access individual columns. Write the `OnGetColumnData` or `OnPutData` event handler to read external data and pass it to the database. Call the `Load` method to start loading data.

Inheritance Hierarchy

[TDALoader](#)

TPgLoader

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.1 Members

[TPgLoader](#) class overview.

Properties

Name	Description
BufferSize	Holds the size of the memory buffer used by <code>TPgLoader</code> .
Columns	Used to access individual columns.
Connection	Used to specify a connection object and populate the columns connection.
Options	This class allows setting up the behaviour of the <code>TPgLoader</code> class.
TableName	Used to specify the name of the loading table set.
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	Used to read external data.
OnProgress (inherited from TDALoader)	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.
OnPutData	Used to pass external data to the database.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.2 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.
TextMode	Used to load data in the text mode.

Published

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

Columns	Used to access individual columns.
Connection	Used to specify a connection object and populate the columns connection.
Options	This class allows setting up the behaviour of the TPgLoader class.
TableName	Used to specify the name of the loading table set.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.2.1 BufferSize Property

Holds the size of the memory buffer used by TPgLoader.

Class

[TPgLoader](#)

Syntax

```
property bufferSize: integer;
```

Remarks

The BufferSize property contains the size of the memory buffer used by TPgLoader. When the buffer is filled, the loader sends the block of data to the server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.2.2 Columns Property

Used to access individual columns.

Class

[TPgLoader](#)

Syntax

```
property columns: TDAColumns;
```

Remarks

Use the Columns property to access individual columns.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.2.3 Connection Property

Used to specify a connection object and populate the columns connection.

Class

[TPgLoader](#)

Syntax

```
property connection: TPgConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store and to populate the columns collection.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.2.4 Options Property

This class allows setting up the behaviour of the TPgLoader class.

Class

[TPgLoader](#)

Syntax

```
property options: TPgLoaderOptions;
```

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.2.5 TableName Property

Used to specify the name of the loading table set.

Class

[TPgLoader](#)

Syntax

```
property TableName: string;
```

Remarks

Use the TableName property to specify the name of the loading table set.

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.2.6 TextMode Property

Used to load data in the text mode.

Class

[TPgLoader](#)

Syntax

```
property TextMode: boolean;
```

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.3 Events

Events of the **TPgLoader** class.

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

Public

Name	Description
OnProgress (inherited from TDALoader)	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.

Published

Name	Description
OnGetColumnData	Used to read external data.
OnPutData	Used to pass external data to the database.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.3.1 OnGetColumnData Event

Used to read external data.

Class

[TPgLoader](#)

Syntax

```
property OnGetColumnData: TGetColumnDataEvent;
```

Remarks

Write the OnGetColumnData event handler to read external data.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.1.3.2 OnPutData Event

Used to pass external data to the database.

Class

[TPgLoader](#)

Syntax

```
property OnPutData: TDAPutDataEvent;
```

Remarks

Write the OnPutData event handler to pass external data to the database.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.2.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.2.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.3 TPgLoaderOptions Class

This class allows setting up the behaviour of the TPgLoader component.

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

Unit

[PgLoader](#)

Syntax

```
TPgLoaderOptions = class(TDALoaderOptions);
```

Inheritance Hierarchy

[TDALoaderOptions](#)**TPgLoaderOptions**

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.22.1.3.1 Members

[TPgLoaderOptions](#) class overview.

Properties

Name	Description
BufferSize	Holds the size of the memory buffer used by TPgLoader.
QuoteNames	Used to quote columns names in the INSERT statement used for loading data.
TextMode	Used to load data in the text mode.
UseBlankValues	Forces PgDAC to fill the buffer with null values after loading a row to the database.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.22.1.3.2 Properties

Properties of the **TPgLoaderOptions** class.

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

Published

Name	Description
BufferSize	Holds the size of the memory buffer used by TPgLoader.
QuoteNames	Used to quote columns names in the INSERT statement used for loading data.
TextMode	Used to load data in the text mode.
UseBlankValues	Forces PgDAC to fill the buffer with null values after loading a row to the database.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.3.2.1 BufferSize Property

Holds the size of the memory buffer used by TPgLoader.

Class

[TPgLoaderOptions](#)

Syntax

```
property BufferSize: integer default defaultBufferSize;
```

Remarks

The BufferSize property contains the size of the memory buffer used by TPgLoader. When buffer is filled, the loader sends block of data to the server.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.3.2.2 QuoteNames Property

Used to quote columns names in the INSERT statement used for loading data.

Class

[TPgLoaderOptions](#)

Syntax

```
property QuoteNames: boolean;
```

Remarks

Use the QuoteNames property to specify whether the column names in the INSERT statement will be quoted.

When the property value is *True* , then the column names will be quoted. When the property value is *False* then the column names will not be quoted.

The default value is *False* .

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.3.2.3 TextMode Property

Used to load data in the text mode.

Class

[TPgLoaderOptions](#)

Syntax

```
property TextMode: boolean default False;
```

Remarks

Use the TextMode property to load data in the text mode.

TPgLoader supports two load modes: text and binary. By default the binary mode is used for a connection with 3.0 protocol. Set TextMode property to True to force text mode. In binary mode TPgLoader may work slightly faster but some data type are not supported in this mode. In text mode you can load data to columns with any PostgreSQL data type.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.22.1.3.2.4 UseBlankValues Property

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

Class

[TPgLoaderOptions](#)

Syntax

```
property UseBlankValues: boolean;
```

Remarks

Used to force PgDAC to fill the buffer with null values after loading a row to the database.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23 PgObjects

This unit contains classes for PostgreSQL specific data types.

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.
TPgType	A class holding the description of PostgreSQL composite types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1 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.
TPgType	A class holding the description of PostgreSQL composite types.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1 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(TPgSharedObject);
```

Remarks

The TCustomPgTmeStamp class is a base class for the TPgDate, TPgTime, and TPgTimeStamp classes.

Inheritance Hierarchy

[TSharedObject](#)

TPgSharedObject

TCustomPgTimeStamp

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.1 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.
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.
Ticks	Holds the time part of timestamp in microseconds.

TimeZoneOffset	Specifies the time zone offset of a timestamp object (in seconds).
--------------------------------	--

Methods

Name	Description
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2 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.
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.
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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2.1 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2.2 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2.3 Days Property

Holds the date part of timestamp.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property Days: integer;
```

Remarks

The Days property holds the date part of timestamp as a number of days since 01-01-2000.

See Also

- [DecodeDate](#)
- [EncodeDate](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2.5 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2.6 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2.7 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2.8 Ticks Property

Holds the time part of timestamp in microseconds.

Class

[TCustomPgTimeStamp](#)

Syntax

```
property Ticks: int64;
```

Remarks

The Ticks property holds the time part of timestamp in microseconds.

See Also

- [DecodeTime](#)
- [EncodeTime](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.2.9 TimeZoneOffset Property

Specifies the time zone offset of a timestamp object (in seconds).

Class

[TCustomPgTimeStamp](#)

Syntax

```
property TimeZoneOffset: integer;
```

Remarks

Use the TimeZoneOffset property to specify the time zone offset of a timestamp object (in seconds).

For values of `TIMESTAMP WITH TIMEZONE` read this property to get the offset of the timestamp value from Universal Time (UTC).

The value of `TimeZoneOffset` is specified in seconds.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.3 Methods

Methods of the **TCustomPgTimeStamp** class.

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

Public

Name	Description
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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.3.1 Assign Method

Assigns a value from another TCustomPgTimeStamp object to this object.

Class

[TCustomPgTimeStamp](#)

Syntax

```
procedure Assign(Source: TCustomPgTimeStamp); reintroduce;  
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.3.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.3.3 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.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.23.1.1.3.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.3.5 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.3.6 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.3.7 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.1.3.8 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.2 TPgAttribute Class

A class holding the description of an attribute of PostgreSQL composite type.

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

Unit

[PgObjects](#)

Syntax

```
TPgAttribute = class(TAttribute);
```

Remarks

The TPgAttribute class holds the description of an attribute of PostgreSQL composite type. You can use TPgRowType.Attributes to access individual attributes.

Inheritance Hierarchy

[TAttribute](#)

TPgAttribute

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.2.1 Members

[TPgAttribute](#) class overview.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.3 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](#)

TPgSharedObject

[TPgGeometric](#)

[TPgPointsArray](#)

TPgBox

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.3.1 Members

[TPgBox](#) class overview.

Properties

Name	Description
LowerLeft	Holds the lower-left corner of the box.
UpperRight	Holds the upper-right corner of the box.

Methods

Name	Description
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.3.2 Properties

Properties of the **TPgBox** class.

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

Public

Name	Description
LowerLeft	Holds the lower-left corner of the box.
UpperRight	Holds the upper-right corner of the box.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.3.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.3.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.4 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](#)

TPgSharedObject

[TPgGeometric](#)

TPgCircle

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.23.1.4.1 Members

[TPgCircle](#) class overview.

Properties

Name	Description
Center	Holds the center point of the circle.
Radius	Holds the radius of the circle.

Methods

Name	Description
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.4.2 Properties

Properties of the **TPgCircle** class.

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

Public

Name	Description
Center	Holds the center point of the circle.
Radius	Holds the radius of the circle.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.4.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.4.2.2 Radius Property

Holds the radius of the circle.

Class

[TPgCircle](#)

Syntax

```
property Radius: Double;
```

Remarks

The Radius property holds the radius of the circle.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.5 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](#)

TPgSharedObject

[TCustomPgTimeStamp](#)

TPgDate

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.5.1 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.
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.
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
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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.23.1.6 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(TPgSharedObject);
```

Remarks

The TPgGeometric class is a class for classes that work with geometric data types.

Inheritance Hierarchy

[TSharedObject](#)

TPgSharedObject

TPgGeometric

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.23.1.6.1 Members

[TPgGeometric](#) class overview.

Methods

Name	Description
Assign	Assigns a value from another TPgGeometric object to this object.

© 1997-2024

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

Devart. All Rights Reserved.

5.23.1.6.2 Methods

Methods of the **TPgGeometric** class.

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

Public

Name	Description
Assign	Assigns a value from another TPgGeometric object to this object.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.6.2.1 Assign Method

Assigns a value from another TPgGeometric object to this object.

Class

[TPgGeometric](#)

Syntax

```
procedure Assign(Source: TPgGeometric); reintroduce; 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.

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7 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(TPgSharedObject);
```

Remarks

The TPgInterval class is used to work with PostgreSQL INTERVAL data type.

Inheritance Hierarchy

[TSharedObject](#)

TPgSharedObject

TPgInterval

© 1997-2024
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7.1 Members

[TPgInterval](#) class overview.

Properties

Name	Description
Days	Used to get the days part of an interval.
MonthsFull	Used to get the months part of an interval.
SecondsFull	Used to get the time part of an interval represented in

	seconds.
--	----------

Methods

Name	Description
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.23.1.7.2 Properties

Properties of the **TPgInterval** class.

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

Public

Name	Description
Days	Used to get the days part of an interval.
MonthsFull	Used to get the months part of an interval.
SecondsFull	Used to get the time part of an interval represented in seconds.

See Also

- [TPgInterval Class](#)

- [TPgInterval Class Members](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7.3 Methods

Methods of the **TPgInterval** class.

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

Public

Name	Description
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.

See Also

- [TPgInterval Class](#)

- [TPgInterval Class Members](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7.3.1 Assign Method

Assigns a value from another TPgInterval object to this object.

Class

[TPgInterval](#)

Syntax

```
procedure Assign(Source: TPgInterval); reintroduce;
```

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7.3.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7.3.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.7.3.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.8 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](#)

TPgSharedObject

[TPgGeometric](#)

[TPgPointsArray](#)

TPgLSeg

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.8.1 Members

[TPgLSeg](#) class overview.

Properties

Name	Description
EndPoint	Holds the end point of the line segment.

StartPoint	Holds the start point of the line segment.
----------------------------	--

Methods

Name	Description
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.8.2 Properties

Properties of the **TPgLSeg** class.

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

Public

Name	Description
EndPoint	Holds the end point of the line segment.
StartPoint	Holds the start point of the line segment.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.8.2.1 EndPoint Property

Holds the end point of the line segment.

Class

[TPgLSeg](#)

Syntax

```
property EndPoint: TPgPoint;
```

Remarks

The EndPoint property holds the end point of the line segment.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.8.2.2 StartPoint Property

Holds the start point of the line segment.

Class

[TPgLSeg](#)

Syntax

```
property StartPoint: TPgPoint;
```

Remarks

The StartPoint property holds the start point of the line segment.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.9 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](#)

TPgSharedObject

[TPgGeometric](#)

[TPgPointsArray](#)

TPgPath

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.9.1 Members

[TPgPath](#) class overview.

Properties

Name	Description
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.

Methods

Name	Description
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.9.2 Properties

Properties of the **TPgPath** class.

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

Public

Name	Description
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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.9.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.9.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.9.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.10 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](#)

TPgSharedObject

[TPgGeometric](#)

TPgPoint

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.10.1 Members

[TPgPoint](#) class overview.

Properties

Name	Description
X	Holds the X coordinate of the point.
Y	Holds the Y coordinate of the point.

Methods

Name	Description
Assign	Assigns a value from another TPgPoint object to this object.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.10.2 Properties

Properties of the **TPgPoint** class.

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

Public

Name	Description
X	Holds the X coordinate of the point.
Y	Holds the Y coordinate of the point.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.10.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.10.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.10.3 Methods

Methods of the **TPgPoint** class.

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

Public

Name	Description
Assign	Assigns a value from another TPgPoint object to this object.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.10.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.11 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](#)

TPgSharedObject

[TPgGeometric](#)**TPgPointsArray**

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.23.1.11.1 Members

[TPgPointsArray](#) class overview.

Methods

Name	Description
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)**5.23.1.12 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](#)

TPgSharedObject

[TPgGeometric](#)

[TPgPointsArray](#)**TPgPolygon**

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.23.1.12.1 Members

[TPgPolygon](#) class overview.

Properties

Name	Description
Count	Holds the count of points in the polygon.
Points	Used to access a point in the polygon by its index.

Methods

Name	Description
Assign (inherited from TPgGeometric)	Assigns a value from another TPgGeometric object to this object.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.23.1.12.2 Properties

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

Public

Name	Description
Count	Holds the count of points in the polygon.
Points	Used to access a point in the polygon by its index.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.12.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.12.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

5.23.1.13 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.13.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.13.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.13.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14 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**

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.23.1.14.1 Members

[TPgRow](#) class overview.

Properties

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 TPgLargeObjecte 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.

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.4 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.5 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.6 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.7 AttrAsPgLargeObject Property(Indexer)

Used to get a reference to a TPgLargeObjecte 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.8 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.9 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.10 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.11 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.12 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.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.23.1.14.2.13 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.14 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.15 AttrIsNull Property(Indexer)

Used to find out if an attribute is NULL, or to set an attribute value to NULL.

Class

[TPgRow](#)

Syntax

```
property AttrIsNull[const Name: string]: boolean;
```

Parameters

Name

Holds the name of an attribute.

Remarks

Use the `AttrIsNull` property to find out if an attribute is NULL, or to set an attribute value to NULL.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.16 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.2.17 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.14.3.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.15 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 TPgRowType.

Inheritance Hierarchy

[TSharedObject](#)

[TObjectType](#)

[TPgType](#)

TPgRowType

See Also

- [TPgConnection.GetRowType](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.15.1 Members

[TPgRowType](#) class overview.

Properties

Name	Description
AttributeCount (inherited from TObjectType)	Used to indicate the number of attributes of type.
Attributes (inherited from TObjectType)	Used to access separate attributes.
BaseTypeOID (inherited from TPgType)	Holds the OID of the base type.
BufferSize	Holds the memory buffer size used by TPgRowType.
DataType (inherited from TObjectType)	Used to indicate the type of object dtObject, dtArray or dtTable.

IsDescribed (inherited from TPgType)	Specifies whether the composite type has been described.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size (inherited from TObjectType)	Used to learn the size of an object instance.
TableOID (inherited from TPgType)	Holds the OID of the table.
TypeOID (inherited from TPgType)	Holds the OID of the composite type.

Methods

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Describe (inherited from TPgType)	Overloaded. Provides information about a composite type from the database.
FindAttribute (inherited from TObjectType)	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from TSharedObject)	Decrements the reference count.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.15.2 Properties

Properties of the **TPgRowType** class.

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

Public

Name	Description
AttributeCount (inherited from TObjectType)	Used to indicate the number of attributes of type.
Attributes (inherited from TObjectType)	Used to access separate attributes.
BaseTypeOID (inherited from TPgType)	Holds the OID of the base type.
BufferSize	Holds the memory buffer size used by TPgRowType.
DataType (inherited from TObjectType)	Used to indicate the type of object dtObject, dtArray or dtTable.
IsDescribed (inherited from TPgType)	Specifies whether the composite type has been described.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size (inherited from TObjectType)	Used to learn the size of an object instance.
TableOID (inherited from TPgType)	Holds the OID of the table.
TypeOID (inherited from TPgType)	Holds the OID of the composite type.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.15.2.1 BufferSize Property

Holds the memory buffer size used by TPgRowType.

Class

[TPgRowType](#)

Syntax

```
property BufferSize: Integer;
```

Remarks

The BufferSize property holds the memory buffer size used by TPgRowType.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16 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

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.2 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 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.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.23.1.16.2.1 Cached Property

Used to specify whether the content of large object should be stored in memory buffer on the client.

Class

[TPgSQLLargeObject](#)

Syntax

```
property cached: boolean;
```

Remarks

Use the Cached property to specify whether the content of large object should be stored in memory buffer on the client.

By default the whole content of large object is stored in the memory buffer on the client. Set Cached to False if you want to reduce memory usage in your application when working with very large object. When Cached is False, the Read and Write methods read/write data directly from/to the database.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.2.2 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.

© 1997-2024

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

5.23.1.16.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.3 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.3.1 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.3.2 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.3.3 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.3.4 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.3.5 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.16.3.6 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.17 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](#)

TPgSharedObject

[TCustomPgTimeStamp](#)

TPgTime

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.17.1 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.
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.
Ticks (inherited from TCustomPgTimeStamp)	Holds the time part of

	timestamp in microseconds.
TimeZoneOffset	Used to get or set the time zone offset.

Methods

Name	Description
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.17.2 Properties

Properties of the **TPgTime** class.

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

Public

Name	Description
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.
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.
Ticks (inherited from TCustomPgTimeStamp)	Holds the time part of timestamp in microseconds.
TimeZoneOffset	Used to get or set the time zone offset.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.23.1.17.2.1 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.18 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](#)

TPgSharedObject

[TCustomPgTimeStamp](#)

TPgTimeStamp

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.18.1 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.
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.
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
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.19 TPgType Class

A class holding the description of PostgreSQL composite types.

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

Unit

[PgObjects](#)

Syntax

```
TPgType = class(TObjectType);
```

Remarks

The TPgType class holds the description of PostgreSQL composite types.

Inheritance Hierarchy

[TSharedObject](#)

[TObjectType](#)

TPgType

See Also

- [TPgRowType](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.19.1 Members

[TPgType](#) class overview.

Properties

Name	Description
AttributeCount (inherited from TObjectType)	Used to indicate the number of attributes of type.
Attributes (inherited from TObjectType)	Used to access separate attributes.
BaseTypeOID	Holds the OID of the base type.
DataType (inherited from TObjectType)	Used to indicate the type of object dtObject, dtArray or dtTable.
IsDescribed	Specifies whether the composite type has been described.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size (inherited from TObjectType)	Used to learn the size of an object instance.
TableOID	Holds the OID of the table.
TypeOID	Holds the OID of the composite type.

Methods

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

Describe	Overloaded. Provides information about a composite type from the database.
FindAttribute (inherited from TObjectType)	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from TSharedObject)	Decrements the reference count.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.19.2 Properties

Properties of the **TPgType** class.

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

Public

Name	Description
AttributeCount (inherited from TObjectType)	Used to indicate the number of attributes of type.
Attributes (inherited from TObjectType)	Used to access separate attributes.
BaseTypeOID	Holds the OID of the base type.
DataType (inherited from TObjectType)	Used to indicate the type of object dtObject, dtArray or dtTable.
IsDescribed	Specifies whether the composite type has been described.
RefCount (inherited from TSharedObject)	Used to return the count of reference to a TSharedObject object.
Size (inherited from TObjectType)	Used to learn the size of an object instance.
TableOID	Holds the OID of the table.
TypeOID	Holds the OID of the composite type.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.19.2.1 BaseTypeOID Property

Holds the OID of the base type.

Class

[TPgType](#)

Syntax

```
property BaseTypeOID: Cardinal;
```

Remarks

The BaseTypeOID property holds the OID of the base type.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.19.2.2 IsDescribed Property

Specifies whether the composite type has been described.

Class

[TPgType](#)

Syntax

```
property IsDescribed: boolean;
```

Remarks

Use the the IsDescribed property to find out whether the composite type has been described.

See Also

- [Describe](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.19.2.3 TableOID Property

Holds the OID of the table.

Class

[TPgType](#)

Syntax

```
property TableOID: Cardinal;
```

Remarks

The TableOID property holds the OID of the table.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.19.2.4 TypeOID Property

Holds the OID of the composite type.

Class

[TPgType](#)

Syntax

```
property TypeOID: Cardinal;
```

Remarks

The TypeOID property holds the OID of the composite type.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.19.3 Methods

Methods of the **TPgType** class.

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

Public

Name	Description
AddRef (inherited from TSharedObject)	Increments the reference count for the number of references dependent on the TSharedObject object.
Describe	Overloaded. Provides information about a composite type from the database.
FindAttribute (inherited from TObjectType)	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from TSharedObject)	Decrements the reference count.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.23.1.19.3.1 Describe Method

Provides information about a composite type from the database.

Class

[TPgType](#)

Overload List

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

Describe	Provides information about a composite type from the database by the OID.
Describe(Connection: TPgSQLConnection; const TypeName: string)	Provides information about a composite type from the database by the name.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Provides information about a composite type from the database by the name.

Class

[TPgType](#)

Syntax

```
function Describe(Connection: TPgSQLConnection; const TypeName: string): boolean; overload; virtual;
```

Parameters

Connection

Holds the connection used to get information about a type.

TypeName

Holds the name of a composite type.

Remarks

Call the Describe method to get information about a composite type from the database by the name.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.24 PgScript

This unit contains the implementation of the TPgScript component.

Classes

Name	Description
TPgScript	Executes sequences of SQL statements.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.24.1 Classes

Classes in the **PgScript** unit.

Classes

Name	Description
TPgScript	Executes sequences of SQL statements.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.24.1.1 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.24.1.1.1 Members

[TPgScript](#) class overview.

Properties

Name	Description
Connection	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.
FindMacro (inherited from TDAScript)	Finds a macro with the specified name.
MacroByName (inherited from TDAScript)	Finds a macro with the specified 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.24.1.1.2 Properties

Properties of the **TPgScript** class.

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

Public

Name	Description
DataSet (inherited from TDAScript)	Refers to a dataset that holds the result set of query execution.
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.
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.

Published

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

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

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.24.1.1.2.1 Connection Property

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

Class

[TPgScript](#)

Syntax

```
property Connection: TPgConnection;
```

Remarks

Use the Connection property to specify the connection in which the script will be executed. If a connection has not been opened before running the Execute method, the TPgConnection.Connect method is called.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.25 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.25.1 Classes

Classes in the **PgSQLMonitor** unit.

Classes

Name	Description
TPgSQLMonitor	This component serves for monitoring dynamic SQL execution in PgDAC-based applications.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.25.1.1 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](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.25.1.1.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.26 PgTransaction

5.26.1 Classes

Classes in the **PgTransaction** unit.

Classes

Name	Description
TPgTransaction	A component for managing transactions in an application.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.26.1.1 TPgTransaction Class

A component for managing transactions in an application.

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

Unit

PgTransaction

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

- [TCustomDACConnection.StartTransaction](#)
- [TCustomDACConnection.Commit](#)
- [TCustomDACConnection.Rollback](#)

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.26.1.1.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.26.1.1.2 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.

Published

Name	Description
IsolationLevel	Used to specify how the transactions containing database modifications are handled.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.26.1.1.2.1 IsolationLevel Property

Used to specify how the transactions containing database modifications are handled.

Class

[TPgTransaction](#)

Syntax

```
property IsolationLevel: TPgIsolationLevel default
    pilReadCommitted;
```

Remarks

Use the IsolationLevel property to specify how the transactions containing database modifications are handled.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.27 VirtualDataSet

This unit contains implementation of the TVirtualDataSet component.

Classes

Name	Description
TCustomVirtualDataSet	A base class for representation of arbitrary data in tabular form.
TVirtualDataSet	Dataset that processes arbitrary non-tabular data.

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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)

5.27.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.27.1.1 TCustomVirtualDataSet Class

A base class for representation of arbitrary data in tabular form.

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

Unit

[VirtualDataSet](#)

Syntax

```
TCustomVirtualDataSet = class(TMemDataSet);
```

Inheritance Hierarchy

[TMemDataSet](#)

TCustomVirtualDataSet

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.27.1.1.1 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.27.1.2 TVirtualDataSet Class

Dataset that processes arbitrary non-tabular data.

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

Unit

[VirtualDataSet](#)

Syntax

```
TVirtualDataSet = class(TCustomVirtualDataSet);
```

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomVirtualDataSet](#)

TVirtualDataSet

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.27.1.2.1 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.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.27.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.27.2.1 TOnDeleteRecordEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.

Unit

[VirtualDataSet](#)

Syntax

```
TOnDeleteRecordEvent = procedure (Sender: TObject; RecNo: Integer) of object;
```

Parameters

Sender

An object that raised the event.

RecNo

Number of the record being deleted.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.27.2.2 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.27.2.3 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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.27.2.4 TOnModifyRecordEvent Procedure Reference

This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.

Unit

[VirtualDataSet](#)

Syntax

```
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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28 VirtualTable

This unit contains implementation of the TVirtualTable component.

Classes

Name	Description
TVirtualTable	Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28.1 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.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28.1.1 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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28.1.1.1 Members

[TVirtualTable](#) class overview.

Properties

Name	Description
CachedUpdates (inherited from TMemDataSet)	Used to enable or disable the use of cached updates for a dataset.
DefaultSortType	Used to determine the default type of local sorting for string fields.
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.
Assign	Copies fields and data from another TDataSet component.
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.
LoadFromFile	Loads data from a file into a TVirtualTable component.
LoadFromStream	Copies data from a stream into a TVirtualTable component.
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.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28.1.1.2 Properties

Properties of the **TVirtualTable** class.

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

Public

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.

Published

Name	Description
DefaultSortType	Used to determine the default type of local sorting for string fields.

See Also

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28.1.1.2.1 DefaultSortType Property

Used to determine the default type of local sorting for string fields.

Class

[TVirtualTable](#)

Syntax

```
property DefaultSortType: TSortType default stCaseSensitive;
```

Remarks

The DefaultSortType property is used when a sort type is not specified explicitly after the field name in the [TMemDataSet.IndexFieldNames](#) property of a dataset.

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28.1.1.3 Methods

Methods of the **TVirtualTable** class.

For a complete list of the **TVirtualTable** class members, see the [TVirtualTable 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.
Assign	Copies fields and data from another TDataSet component.
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.
LoadFromFile	Loads data from a file into a TVirtualTable component.
LoadFromStream	Copies data from a stream into a TVirtualTable component.
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

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

© 1997-2024

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28.1.1.3.1 Assign Method

Copies fields and data from another TDataSet component.

Class

[TVirtualTable](#)

Syntax

```
procedure Assign(Source: TPersistent); override;
```

Parameters

Source

Holds the TDataSet component to copy fields and data from.

Remarks

Call the Assign method to copy fields and data from another TDataSet component.

Note: Unsupported field types are skipped (i.e. destination dataset will contain less fields than the source one). This may happen when Source is not a TVirtualTable component but some server-oriented dataset.

Example

```
Query1.SQL.Text := 'SELECT * FROM DEPT';  
Query1.Active := True;  
VirtualTable1.Assign(Query1);  
VirtualTable1.Active := True;
```

See Also

- [TVirtualTable](#)

© 1997-2024

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28.1.1.3.2 LoadFromFile Method

Loads data from a file into a TVirtualTable component.

Class

[TVirtualTable](#)

Syntax

```
procedure LoadFromFile(const FileName: string; LoadFields:  
boolean = True; DecodeHTMLEntities: boolean = True);
```

Parameters

FileName

Holds the name of the file to load data from.

LoadFields

Indicates whether to load fields from the file.

DecodeHTMLEntities

Indicates whether to decode HTML entities from the file.

Remarks

Call the LoadFromFile method to load data from a file into a TVirtualTable component. Specify the name of the file to load into the field as the value of the FileName parameter. This file may be an XML document in ADO-compatible format or in virtual table data format. The file format is detected automatically.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.28.1.1.3.3 LoadFromStream Method

Copies data from a stream into a TVirtualTable component.

Class

[TVirtualTable](#)

Syntax

```
procedure LoadFromStream(Stream: TStream; LoadFields: boolean = True; DecodeHTMLEntities: boolean = True);
```

Parameters

Stream

Holds the stream from which the field's value is copied.

LoadFields

Indicates whether to load fields from the stream.

DecodeHTMLEntities

Indicates whether to decode HTML entities from the stream.

Remarks

Call the LoadFromStream method to copy data from a stream into a TVirtualTable component. Specify the stream from which the field's value is copied as the value of the Stream parameter. Data in the stream may be in ADO-compatible format or in virtual table data format. The data format is detected automatically.

© 1997-2024

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)