

Table of Contents

Part I What's New	1
Part II General Information	42
1 Overview	43
2 Features	47
3 Requirements	55
4 Compatibility	56
5 Using Several DAC Products in One IDE	72
6 Component List	72
7 Hierarchy Chart	75
8 Editions	77
9 Licensing	82
10 Getting Support	86
Part III Getting Started	87
1 Installation	93
2 Migration Wizard	95
3 UniDAC Basics	97
4 Demo Projects	117
5 Deployment	124
Part IV Using UniDAC	125
1 Connecting to Database	126
2 Updating data with UniDAC	130
3 Master/Detail Relationships	133
4 Data Types	135
5 Data Type Mapping	143
6 Data Encryption	151
7 Working in an Unstable Network	153
8 Disconnected Mode	155
9 Batch Operations	156
10 Increasing Performance	162
11 Using Connection Pooling	164
12 Macros	166
13 DataSet Manager	167
14 Executing Stored Procedures	173
15 Transactions	176

16	Unified SQL	178
17	DBMonitor	185
18	Writing GUI Applications with UniDAC	186
19	Syntax Highlighting in Component Editors	186
20	Compatibility with Previous Versions	188
21	64-bit Development with Embarcadero RAD Studio XE2	190
22	C++ Builder Development for Android and iOS	196

Part V Provider-Specific Notes 203

1	Database Providers	204
	UniDAC and Adaptive Server Enterprise	204
	UniDAC and Advantage Database Server	210
	UniDAC and Amazon Redshift	214
	UniDAC and DB2	220
	UniDAC and DBF	224
	UniDAC and InterBase/Firebird	229
	InterBase/Firebird Provider.....	229
	OTW Network Encryption.....	238
	UniDAC and Microsoft Access	240
	UniDAC and Microsoft Excel	245
	UniDAC and MongoDB	248
	UniDAC and MySQL	258
	MySQL Provider.....	258
	Connecting via SSL.....	266
	Connecting via SSH.....	277
	Network Tunneling.....	290
	UniDAC and NexusDB.....	292
	UniDAC and ODBC	297
	UniDAC and Oracle	302
	Oracle Provider.....	302
	Connecting via SSL.....	315
	Connecting via SSH.....	328
	Network Tunneling.....	341
	UniDAC and SQLite	344
	SQLite Provider.....	344
	Database File Encryption.....	352
	UniDAC and PostgreSQL	355
	PostgreSQL Provider.....	355
	Connecting via SSL.....	363
	Connecting via SSH.....	374
	Network Tunneling.....	385
	UniDAC and SQL Server	388
	SQL Server Provider.....	388
	Connecting via SSL.....	403
	Connecting via SSH.....	404
	Network Tunneling.....	417
2	Cloud Providers	420
	UniDAC and BigCommerce	420
	UniDAC and Dynamics 365	424
	UniDAC and FreshBooks	428

UniDAC and Google BigQuery	433
UniDAC and HubSpot	436
UniDAC and Magento	440
UniDAC and Mailchimp	445
UniDAC and NetSuite	449
UniDAC and QuickBooks	454
UniDAC and Salesforce	457
UniDAC and Salesforce MC	462
UniDAC and SugarCRM	467
UniDAC and Zoho CRM	471
3 Database Specific Aspects of 64-bit Development	475
Part VI Reference	478
1 CROAccess	480
Classes	481
TCRCursor Class.....	482
Members	482
Types	483
TBeforeFetchProc Procedure Reference.....	483
Enumerations	484
TCRIsolationLevel Enumeration.....	484
TCRTransactionAction Enumeration.....	485
TCursorState Enumeration.....	486
2 CRBatchMove	486
Classes	487
TCRBatchMove Class.....	488
Members	488
Properties	490
AbortOnKeyViol Property.....	492
AbortOnProblem Property.....	492
ChangedCount Property.....	493
CommitCount Property.....	493
Destination Property.....	494
FieldMappingMode Property.....	494
KeyViolCount Property.....	495
Mappings Property.....	495
Mode Property.....	496
MovedCount Property.....	497
ProblemCount Property.....	497
RecordCount Property.....	498
Source Property.....	498
Methods	499
Execute Method.....	499
Events	500
OnBatchMoveProgress Event.....	500
Types	501
TCRBatchMoveProgressEvent Procedure Reference.....	501
Enumerations	501
TCRBatchMode Enumeration.....	502
TCRFieldMappingMode Enumeration.....	503
3 CREncryption	503
Classes	504

TCREncryptor Class.....	504
Members	505
Properties	505
DataHeader Property.....	506
EncryptionAlgorithm Property.....	507
HashAlgorithm Property.....	507
InvalidHashAction Property.....	508
Passw ord Property.....	508
Methods	509
SetKey Method.....	509
Enumerations	510
TCREncDataHeader Enumeration.....	510
TCREncryptionAlgorithm Enumeration.....	511
TCRHashAlgorithm Enumeration.....	512
TCRInvalidHashAction Enumeration.....	512
4 CRGrid	513
Classes	513
TCRDBGrid Class.....	513
Members	514
Properties	516
Columns Property.....	518
Filtered Property.....	518
LevelDelimiterChar Property.....	519
OnMemoClick Property.....	519
OptionsEx Property.....	519
OptionsMenu Property.....	520
Methods	521
ActivateFilterEdit Method.....	522
ActivateSearchEdit Method.....	523
AdjustColumns Method.....	523
ApplyFilter Method.....	524
CalcTitleLevel Method.....	524
ClearFilters Method.....	525
ClearSorting Method.....	525
DataChanged Method.....	525
GetGridSize Method.....	526
GetTitleLevel Method.....	526
Events	526
OnGetCellParams Event.....	527
5 CRVio	527
Classes	528
THttpOptions Class.....	528
Members	529
Properties	530
Enabled Property.....	530
Passw ord Property.....	531
ProxyOptions Property.....	531
TrustServerCertificate Property.....	532
Url Property	532
Username Property.....	533
TProxyOptions Class.....	533
Members	534
Properties	534

Hostname Property	535
Password Property.....	535
Port Property	536
Username Property.....	536
Enumerations	536
TIPVersion Enumeration.....	537
6 CRXml	537
Structs	537
TAttribute Record.....	538
7 DAAlerter	539
Classes	539
TDAAlerter Class	540
Members	540
Properties	541
Active Property.....	542
AutoRegister Property.....	542
Connection Property.....	543
Methods	543
SendEvent Method.....	544
Start Method	544
Stop Method	545
Events	545
OnError Event.....	546
Types	546
TAlerterErrorEvent Procedure Reference.....	547
TAlerterEventEvent Procedure Reference.....	547
8 DADump	548
Classes	548
TDADump Class	549
Members	549
Properties	551
Connection Property.....	552
Debug Property.....	553
Options Property.....	553
SQL Property	554
TableNames Property.....	555
Methods	555
Backup Method.....	556
BackupQuery Method.....	557
BackupToFile Method.....	557
BackupToStream Method.....	558
Restore Method.....	559
RestoreFromFile Method.....	560
RestoreFromStream Method.....	560
Events	561
OnBackupProgress Event.....	562
OnError Event.....	563
OnRestoreProgress Event.....	563
TDADumpOptions Class.....	564
Members	564
Properties	565
AddDrop Property.....	566
CompleteInsert Property.....	566

GenerateHeader Property.....	567
QuoteNames Property.....	567
Types	568
TDABackupProgressEvent Procedure Reference.....	568
TDARestoreProgressEvent Procedure Reference.....	569
9 DALoader	569
Classes	570
TDAColumn Class.....	570
Members	571
Properties	571
FieldType Property.....	572
Name Property.....	572
TDAColumns Class.....	573
Members	574
Properties	574
Items Property(Indexer).....	574
TDALoader Class.....	575
Members	576
Properties	577
Columns Property.....	577
Connection Property.....	578
TableName Property.....	578
Methods	579
CreateColumns Method.....	580
Load Method	580
LoadFromDataSet Method.....	581
PutColumnData Method	581
PutColumnData Method	582
PutColumnData Method	583
Events	583
OnGetColumnData Event.....	584
OnProgress Event.....	585
OnPutData Event.....	586
TDALoaderOptions Class.....	587
Members	587
Properties	587
UseBlankValues Property.....	588
Types	588
TDAPutDataEvent Procedure Reference.....	589
TGetColumnDataEvent Procedure Reference.....	589
TLoaderProgressEvent Procedure Reference.....	590
10 DAScript	590
Classes	591
TDAScript Class.....	592
Members	592
Properties	594
Connection Property.....	596
DataSet Property.....	596
Debug Property.....	597
Delimiter Property.....	598
EndLine Property.....	598
EndOffset Property.....	599
EndPos Property.....	599

Macros Property	599
SQL Property	600
StartLine Property.....	601
StartOffset Property.....	601
StartPos Property.....	601
Statements Property.....	602
Methods	603
BreakExec Method.....	604
ErrorOffset Method.....	604
Execute Method.....	605
ExecuteFile Method.....	606
ExecuteNext Method.....	606
ExecuteStream Method.....	607
FindMacro Method.....	607
MacroByName Method.....	608
Events	609
AfterExecute Event.....	610
BeforeExecute Event.....	610
OnError Event.....	611
TDAStatement Class	611
Members	612
Properties	613
EndLine Property.....	614
EndOffset Property.....	614
EndPos Property.....	615
Omit Property	615
Params Property.....	616
Script Property.....	616
SQL Property	617
StartLine Property.....	617
StartOffset Property.....	617
StartPos Property.....	618
Methods	618
Execute Method.....	619
TDAStatements Class	619
Members	620
Properties	620
Items Property(Indexer).....	621
Types	621
TAfterStatementExecuteEvent Procedure Reference.....	622
TBeforeStatementExecuteEvent Procedure Reference.....	622
TOnErrorEvent Procedure Reference.....	623
Enumerations	623
TErrorAction Enumeration.....	624
11 DASQLMonitor	624
Classes	625
TCustomDASQLMonitor Class.....	626
Members	626
Properties	627
Active Property.....	628
DBMonitorOptions Property.....	628
Options Property.....	629
TraceFlags Property.....	629
Events	630

OnSQL Event	630
TDBMonitorOptions Class.....	631
Members	631
Properties	632
Host Property	633
Port Property	633
ReconnectTimeout Property.....	634
SendTimeout Property.....	634
Types	635
TDATraceFlags Set.....	635
TMonitorOptions Set.....	635
TOnSQLEvent Procedure Reference.....	636
Enumerations	636
TDATraceFlag Enumeration.....	637
TMonitorOption Enumeration.....	638
12 DBAccess	638
Classes	641
EDAError Class	643
Members	644
Properties	644
Component Property.....	645
ErrorCode Property.....	645
TCRDataSource Class.....	646
Members	646
TCustomConnectDialog Class	646
Members	647
Properties	648
CancelButton Property.....	649
Caption Property.....	650
ConnectButton Property.....	650
DialogClass Property.....	650
LabelSet Property.....	651
PasswordLabel Property.....	652
Retries Property.....	652
SavePassword Property.....	652
ServerLabel Property.....	653
StoreLogInfo Property.....	653
UsernameLabel Property.....	654
Methods	654
Execute Method.....	655
GetServerList Method.....	655
TCustomDAConnection Class	656
Members	657
Properties	659
ConnectDialog Property.....	660
ConnectionString Property.....	661
ConvertEOL Property.....	661
InTransaction Property.....	662
LoginPrompt Property.....	662
Options Property.....	663
Password Property.....	664
Pooling Property.....	665
PoolingOptions Property.....	666
Server Property.....	667

Username Property.....	667
Methods	668
ApplyUpdates Method.....	669
ApplyUpdates Method.....	670
ApplyUpdates Method.....	670
Commit Method.....	671
Connect Method.....	672
CreateSQL Method.....	672
Disconnect Method.....	673
ExecProc Method.....	674
ExecProcEx Method.....	675
ExecSQL Method.....	677
ExecSQLEx Method.....	678
GetDatabaseNames Method.....	679
GetKeyFieldNames Method.....	680
GetStoredProcNames Method.....	681
GetTableNames Method.....	682
MonitorMessage Method.....	683
PerformConnect Method.....	683
Ping Method	684
RemoveFromPool Method.....	685
Rollback Method.....	685
StartTransaction Method.....	686
Events	687
OnConnectionLost Event.....	687
OnError Event.....	688
TCustomDADataSet Class.....	688
Members	689
Properties	697
BaseSQL Property.....	701
Conditions Property.....	701
Connection Property.....	702
DataTypeMap Property.....	702
Debug Property.....	703
DetailFields Property.....	703
Disconnected Property.....	704
FetchRows Property.....	705
FilterSQL Property.....	705
FinalSQL Property.....	706
IsQuery Property.....	707
KeyFields Property.....	707
MacroCount Property.....	708
Macros Property.....	708
MasterFields Property.....	709
MasterSource Property.....	710
Options Property.....	711
ParamCheck Property.....	713
ParamCount Property.....	714
Params Property.....	714
ReadOnly Property.....	715
RefreshOptions Property.....	716
RowsAffected Property.....	716
SQL Property	717
SQLDelete Property.....	718

SQLInsert Property	718
SQLLock Property.....	719
SQLRecCount Property.....	720
SQLRefresh Property	721
SQLUpdate Property.....	722
UniDirectional Property.....	723
Methods	724
AddWhere Method.....	728
BreakExec Method.....	729
CloneCursor Method.....	729
CreateBlobStream Method.....	730
DeleteWhere Method.....	731
Execute Method.....	732
Execute Method.....	732
Execute Method.....	733
Executing Method.....	734
Fetched Method.....	734
Fetching Method.....	735
FetchingAll Method.....	735
FindKey Method.....	736
FindMacro Method.....	736
FindNearest Method.....	737
FindParam Method.....	738
GetData Type Method.....	739
GetFieldObject Method.....	739
GetFieldPrecision Method.....	740
GetFieldScale Method	741
GetKeyFieldNames Method.....	742
GetOrderBy Method.....	742
GotoCurrent Method.....	743
Lock Method	744
MacroByName Method.....	744
ParamByName Method.....	745
Prepare Method.....	746
RefreshRecord Method.....	747
RestoreSQL Method.....	747
SaveSQL Method.....	748
SetOrderBy Method.....	749
SQLSaved Method.....	749
UnLock Method.....	750
Events	750
AfterExecute Event.....	751
AfterFetch Event.....	752
AfterUpdateExecute Event.....	752
BeforeFetch Event.....	753
BeforeUpdateExecute Event.....	753
TCustomDA SQL Class	754
Members	755
Properties	757
ChangeCursor Property.....	758
Connection Property.....	759
Debug Property.....	759
FinalSQL Property.....	760
MacroCount Property.....	760

Macros Property	761
ParamCheck Property	762
ParamCount Property	762
Params Property	763
ParamValues Property(Indexer)	764
Prepared Property	765
Row sAffected Property	765
SQL Property	766
Methods	766
BreakExec Method	767
Execute Method	768
Execute Method	768
Execute Method	769
Executing Method	770
FindMacro Method	770
FindParam Method	771
MacroByName Method	772
ParamByName Method	773
Prepare Method	773
UnPrepare Method	774
WaitExecuting Method	775
Events	775
AfterExecute Event	776
TCustomDAUpdateSQL Class	776
Members	777
Properties	778
DataSet Property	780
DeleteObject Property	780
DeleteSQL Property	781
InsertObject Property	781
InsertSQL Property	782
LockObject Property	782
LockSQL Property	783
ModifyObject Property	783
ModifySQL Property	784
RefreshObject Property	784
RefreshSQL Property	785
SQL Property(Indexer)	786
Methods	786
Apply Method	787
ExecSQL Method	788
TDACondition Class	788
Members	789
Properties	789
Enabled Property	790
Name Property	790
Value Property	791
Methods	791
Disable Method	792
Enable Method	792
TDAConditions Class	792
Members	793
Properties	794
Condition Property(Indexer)	795

Enabled Property.....	795
Items Property(Indexer).....	796
Text Property	796
WhereSQL Property.....	797
Methods	797
Add Method	798
Add Method	798
Add Method	799
Delete Method.....	800
Disable Method.....	800
Enable Method.....	801
Find Method	801
Get Method	801
IndexOf Method.....	802
Remove Method.....	802
TDAConnectionOptions Class.....	803
Members	803
Properties	804
Allow ImplicitConnect Property.....	805
DefaultSortType Property.....	806
DisconnectedMode Property.....	806
KeepDesignConnected Property.....	807
LocalFailover Property.....	807
TDAConnectionSSLOptions Class.....	808
Members	808
Properties	808
CACert Property.....	809
Cert Property	810
CipherList Property.....	810
Key Property	810
TDADataSetOptions Class.....	811
Members	811
Properties	814
AutoPrepare Property.....	817
CacheCalcFields Property.....	817
CompressBlobMode Property.....	818
DefaultValues Property.....	818
DetailDelay Property.....	819
FieldsOrigin Property.....	819
FlatBuffers Property.....	820
InsertAllSetFields Property.....	820
LocalMasterDetail Property.....	821
LongStrings Property.....	821
MasterFieldsNullable Property.....	822
NumberRange Property.....	822
QueryRecCount Property.....	822
QuoteNames Property.....	823
RemoveOnRefresh Property.....	824
RequiredFields Property.....	824
ReturnParams Property.....	825
SetFieldsReadOnly Property.....	825
StrictUpdate Property.....	826
TrimFixedChar Property.....	826
UpdateAllFields Property.....	827

UpdateBatchSize Property.....	827
TDAEncryption Class.....	828
Members.....	828
Properties.....	829
Encryptor Property.....	829
Fields Property.....	830
TDAMapRule Class.....	830
Members.....	831
Properties.....	832
DBLengthMax Property.....	833
DBLengthMin Property.....	833
DBScaleMax Property.....	834
DBScaleMin Property.....	834
DBType Property.....	835
FieldLength Property.....	835
FieldName Property.....	835
FieldScale Property.....	836
FieldType Property.....	836
IgnoreErrors Property.....	837
TDAMapRules Class.....	837
Members.....	838
Properties.....	838
IgnoreInvalidRules Property.....	839
TDAMetaData Class.....	839
Members.....	840
Properties.....	844
Connection Property.....	845
MetaDataKind Property.....	846
Restrictions Property.....	847
Methods.....	847
GetMetaDataKinds Method.....	850
GetRestrictions Method.....	850
TDAParam Class.....	851
Members.....	852
Properties.....	854
AsBlob Property.....	855
AsBlobRef Property.....	856
AsFloat Property.....	856
AsInteger Property.....	857
AsLargeInt Property.....	857
AsMemo Property.....	858
AsMemoRef Property.....	858
AsSQLTimeStamp Property.....	859
AsString Property.....	859
AsWideString Property.....	860
DataType Property.....	860
IsNull Property.....	861
ParamType Property.....	861
Size Property.....	862
Value Property.....	862
Methods.....	863
AssignField Method.....	863
AssignFieldValue Method.....	864
LoadFromFile Method.....	864

LoadFromStream Method.....	865
SetBlobData Method.....	866
SetBlobData Method.....	866
SetBlobData Method.....	867
TDAParams Class.....	867
Members.....	868
Properties.....	868
Items Property(Indexer).....	869
Methods.....	870
FindParam Method.....	870
ParamByName Method.....	871
TDATransaction Class.....	872
Members.....	872
Properties.....	873
Active Property.....	874
DefaultCloseAction Property.....	874
Methods.....	875
Commit Method.....	876
Rollback Method.....	876
StartTransaction Method.....	877
Events.....	878
OnCommit Event.....	878
OnCommitRetaining Event.....	879
OnError Event.....	880
OnRollback Event.....	880
OnRollbackRetaining Event.....	881
TMacro Class.....	882
Members.....	883
Properties.....	883
Active Property.....	884
AsDateTime Property.....	885
AsFloat Property.....	885
AsInteger Property.....	886
AsString Property.....	886
Name Property.....	886
Value Property.....	887
TMacros Class.....	887
Members.....	888
Properties.....	889
Items Property(Indexer).....	889
Methods.....	890
AssignValues Method.....	890
Expand Method.....	891
FindMacro Method.....	891
IsEqual Method.....	892
MacroByName Method.....	893
Scan Method.....	893
TPoolingOptions Class.....	894
Members.....	894
Properties.....	895
ConnectionLifetime Property.....	896
MaxPoolSize Property.....	896
MinPoolSize Property.....	897
PoolId Property.....	897

Validate Property.....	898
TSmartFetchOptions Class.....	898
Members	899
Properties	899
Enabled Property.....	900
LiveBlock Property.....	900
PrefetchedFields Property.....	901
SQLGetKeyValues Property.....	901
Types	902
TAfterExecuteEvent Procedure Reference.....	903
TAfterFetchEvent Procedure Reference.....	903
TBeforeFetchEvent Procedure Reference.....	904
TConnectionLostEvent Procedure Reference.....	904
TDAConnectionErrorEvent Procedure Reference.....	905
TDATransactionErrorEvent Procedure Reference.....	905
TRefreshOptions Set.....	906
TUpdateExecuteEvent Procedure Reference.....	906
Enumerations	907
TLabelSet Enumeration.....	907
TLockMode Enumeration.....	908
TRefreshOption Enumeration.....	908
TRetryMode Enumeration.....	909
Variables	909
ChangeCursor Variable.....	910
13 LiteCollation	910
Types	911
TLiteAnsiCollation Function Reference.....	911
TLiteCollation Function Reference.....	911
TLiteWideCollation Function Reference.....	912
14 LiteFunction	912
Types	913
TLiteFunction Function Reference.....	913
15 MemData	914
Classes	915
TBlob Class.....	916
Members	916
Properties	918
AsString Property.....	918
AsWideString Property.....	919
IsUnicode Property.....	919
Size Property	920
Methods	920
Assign Method.....	921
Clear Method	922
LoadFromFile Method.....	922
LoadFromStream Method.....	923
Read Method	924
SaveToFile Method.....	925
SaveToStream Method.....	925
Truncate Method.....	926
Write Method	926
TCompressedBlob Class.....	927
Members	928

Properties	930
Compressed Property	931
CompressedSize Property	931
TDBObject Class	932
Members	932
TMemData Class	933
Members	933
TObjectType Class	933
Members	934
Properties	935
AttributeCount Property	936
Attributes Property (Indexer)	936
DataType Property	937
Size Property	937
Methods	938
FindAttribute Method	938
TSharedObject Class	939
Members	940
Properties	940
RefCount Property	941
Methods	941
AddRef Method	942
Release Method	942
Types	943
TLocateExOptions Set	943
TUpdateRecKinds Set	944
Enumerations	944
TCompressBlobMode Enumeration	945
TConnLostCause Enumeration	946
TDANumericType Enumeration	946
TLocateExOption Enumeration	947
TSortType Enumeration	948
TUpdateRecKind Enumeration	948
16 MemDS	949
Classes	950
TMemDataSet Class	950
Members	951
Properties	954
CachedUpdates Property	955
IndexFieldNames Property	956
KeyExclusive Property	957
LocalConstraints Property	958
LocalUpdate Property	959
Prepared Property	959
Ranged Property	960
UpdateRecordTypes Property	960
UpdatesPending Property	961
Methods	961
ApplyRange Method	964
ApplyUpdates Method	965
ApplyUpdates Method	965
ApplyUpdates Method	967
CancelRange Method	968
CancelUpdates Method	969

CommitUpdates Method.....	970
DeferredPost Method.....	971
EditRangeEnd Method.....	971
EditRangeStart Method.....	972
GetBlob Method.....	973
GetBlob Method.....	973
GetBlob Method.....	974
Locate Method.....	974
Locate Method.....	975
Locate Method.....	975
LocateEx Method.....	977
LocateEx Method.....	977
LocateEx Method.....	978
Prepare Method.....	979
RestoreUpdates Method.....	980
RevertRecord Method.....	980
SaveToJSON Method.....	981
SaveToJSON Method.....	981
SaveToJSON Method.....	982
SaveToXML Method.....	983
SaveToXML Method.....	984
SaveToXML Method.....	985
SetRange Method.....	985
SetRangeEnd Method.....	987
SetRangeStart Method.....	987
UnPrepare Method.....	988
UpdateResult Method.....	989
UpdateStatus Method.....	990
Events	990
OnUpdateError Event.....	991
OnUpdateRecord Event.....	992
Types	993
TJsonOptions Set.....	993
Enumerations	994
TJsonOption Enumeration.....	994
TXmlIndentFormat Enumeration.....	995
17 OracleUniProvider	995
Classes	996
TOraUtils Class.....	996
Members	996
Methods	997
ChangePassw ord Method.....	997
18 SQLiteUniProvider	998
Classes	999
TLiteUtils Class.....	999
Members	999
Methods	1000
EncryptDatabase Method.....	1001
RegisterAnsiCollation Method.....	1002
RegisterCollation Method.....	1003
RegisterFunction Method.....	1003
RegisterWideCollation Method.....	1004
UnRegisterAnsiCollation Method.....	1005

UnRegisterCollation Method.....	1005
UnRegisterFunction Method.....	1006
UnRegisterWideCollation Method.....	1006
19 SQLServerUniProvider	1007
Classes	1007
TMSSqlUtils Class.....	1007
Members	1008
Methods	1008
ChangePassw ord Method.....	1009
20 Uni	1009
Classes	1011
TCustomUniDataSet Class.....	1013
Members	1014
Properties	1023
DMLRefresh Property.....	1027
LastInsertId Property.....	1027
Options Property.....	1029
Params Property.....	1029
SpecificOptions Property.....	1030
Transaction Property.....	1032
UpdateObject Property.....	1032
UpdateTransaction Property.....	1033
Methods	1034
CreateProcCall Method.....	1038
FindParam Method.....	1039
OpenNext Method.....	1040
ParamByName Method.....	1040
TCustomUniTable Class.....	1041
Members	1042
Properties	1051
OrderFields Property.....	1055
TableName Property.....	1056
Methods	1057
PrepareSQL Method.....	1061
TUniBlob Class.....	1061
Members	1062
TUniConnection Class.....	1064
Members	1065
Properties	1069
AutoCommit Property.....	1071
Database Property.....	1072
DefaultTransaction Property.....	1073
Macros Property.....	1073
Port Property.....	1074
ProviderName Property.....	1075
SpecificOptions Property.....	1076
Methods	1077
ActiveMacroValueByName Method.....	1080
AssignConnect Method.....	1080
CommitRetaining Method.....	1081
CreateDataSet Method.....	1082
CreateSQL Method.....	1082
CreateTransaction Method.....	1083

ParamByName Method.....	1083
ReleaseSavepoint Method.....	1084
RollbackRetaining Method.....	1085
RollbackToSavepoint Method.....	1085
Savepoint Method.....	1086
StartTransaction Method.....	1087
StartTransaction Method.....	1087
StartTransaction Method.....	1089
TUniDataSetOptions Class.....	1089
Members.....	1090
Properties.....	1093
EnableBCD Property.....	1096
EnableFMTBCD Property.....	1096
FullRefresh Property.....	1097
SetEmptyStrToNull Property.....	1097
TrimVarChar Property.....	1098
TUniDataSource Class.....	1098
Members.....	1099
TUniEncryptor Class.....	1099
Members.....	1100
TUniMacro Class.....	1100
Members.....	1101
Properties.....	1101
Condition Property.....	1102
Name Property.....	1103
Value Property.....	1103
TUniMacros Class.....	1104
Members.....	1104
Properties.....	1105
Items Property(Indexer).....	1105
Methods.....	1106
Add Method.....	1106
FindMacro Method.....	1107
MacroByName Method.....	1108
TUniMetaData Class.....	1108
Members.....	1109
Properties.....	1113
Connection Property.....	1114
Transaction Property.....	1115
TUniParam Class.....	1115
Members.....	1116
TUniParams Class.....	1118
Members.....	1118
TUniQuery Class.....	1119
Members.....	1120
Properties.....	1129
LockMode Property.....	1134
UpdatingTable Property.....	1135
TUniSQL Class.....	1136
Members.....	1136
Properties.....	1139
Connection Property.....	1141
LastInsertId Property.....	1141
SpecificOptions Property.....	1142

Transaction Property	1144
Methods	1144
CreateProcCall Method.....	1145
FindParam Method.....	1146
ParamByName Method.....	1147
TUniStoredProc Class.....	1148
Members	1149
Properties	1158
LockMode Property.....	1162
StoredProcName Property.....	1163
Methods	1163
ExecProc Method.....	1168
PrepareSQL Method.....	1168
TUniTable Class.....	1169
Members	1170
Properties	1179
LockMode Property.....	1183
OrderFields Property.....	1184
TableName Property.....	1185
TUniTransaction Class.....	1185
Members	1186
Properties	1188
Connections Property(Indexer).....	1189
ConnectionsCount Property.....	1189
IsolationLevel Property.....	1190
Methods	1190
AddConnection Method.....	1191
CommitRetaining Method.....	1192
RemoveConnection Method.....	1192
RollbackRetaining Method.....	1193
TUniUpdateSQL Class.....	1194
Members	1194
Constants	1196
UniDACVersion Constant.....	1196
21 UniAlerter	1196
Classes	1197
TUniAlerter Class.....	1197
Members	1198
Properties	1199
Connection Property.....	1199
22 UniDacVcl	1200
Classes	1200
TUniConnectDialog Class.....	1201
Members	1201
Properties	1203
Connection Property.....	1204
DatabaseLabel Property.....	1205
PortLabel Property.....	1205
ProviderLabel Property.....	1206
23 UniDump	1206
Classes	1206
TUniDump Class.....	1207
Members	1208

24 UniLoader	1209
Classes	1210
TUniLoader Class.....	1210
Members	1211
25 UniProvider	1212
Classes	1212
TUniProvider Class.....	1213
Members	1213
26 UniScript	1213
Classes	1214
TUniScript Class.....	1214
Members	1215
Properties	1217
Connection Property.....	1219
DataSet Property.....	1219
SpecificOptions Property.....	1220
Transaction Property.....	1221
27 UniSQLMonitor	1222
Classes	1222
TUniSQLMonitor Class.....	1222
Members	1223
28 VirtualDataSet	1224
Classes	1224
TCustomVirtualDataSet Class.....	1224
Members	1225
TVirtualDataSet Class.....	1228
Members	1229
Events	1233
OnDeleteRecord Event.....	1234
OnGetFieldValue Event.....	1234
OnGetRecordCount Event.....	1235
OnInsertRecord Event.....	1235
OnModifyRecord Event.....	1236
Types	1237
TOnDeleteRecordEvent Procedure Reference.....	1237
TOnGetFieldValueEvent Procedure Reference.....	1238
TOnGetRecordCountEvent Procedure Reference.....	1238
TOnModifyRecordEvent Procedure Reference.....	1239
29 VirtualQuery	1239
Classes	1239
TCustomVirtualQuery Class.....	1240
Members	1240
Properties	1249
Options Property.....	1253
SourceDataSets Property.....	1253
Events	1254
OnRegisterCollations Event.....	1255
OnRegisterFunctions Event.....	1255
TDataSetLink Class.....	1256
Members	1257
Properties	1257
DataSet Property.....	1258

SchemaName Property.....	1259
TableName Property.....	1259
TDataSetLinks Class.....	1260
Members	1260
Methods	1261
Add Method	1261
Add Method	1262
Add Method	1263
TVirtualCollationManager Class.....	1264
Members	1264
Methods	1265
RegisterAnsiCollation Method.....	1266
RegisterAnsiCollation Method.....	1266
RegisterAnsiCollation Method.....	1267
RegisterCollation Method.....	1267
RegisterCollation Method.....	1268
RegisterCollation Method.....	1268
RegisterDefaultCollations Method.....	1269
RegisterWideCollation Method.....	1269
RegisterWideCollation Method.....	1269
RegisterWideCollation Method.....	1270
UnRegisterAnsiCollation Method.....	1270
UnRegisterCollation Method.....	1271
UnRegisterDefaultCollations Method.....	1271
UnRegisterWideCollation Method.....	1272
TVirtualFunctionManager Class.....	1272
Members	1272
Methods	1273
RegisterFunction Method.....	1273
RegisterFunction Method.....	1274
RegisterFunction Method.....	1274
TVirtualQuery Class.....	1275
Members	1276
Properties	1284
FetchAll Property.....	1289
UpdatingTable Property.....	1289
TVirtualQueryOptions Class.....	1290
Members	1291
Properties	1294
AutoOpenSources Property.....	1297
FullRefresh Property.....	1298
SetEmptyStrToNull Property.....	1298
TrimVarChar Property.....	1299
UseUnicode Property.....	1299
Types	1300
TRegisterFunctionsEvent Procedure Reference.....	1300
30 VirtualTable	1300
Classes	1300
TVirtualAutoIncField Class.....	1301
Members	1302
Properties	1302
AutoGenerateValue Property.....	1303
Increment Property.....	1303
InitialValue Property.....	1304

TVirtualTable Class	1305
Members	1306
Properties	1309
DefaultSortType Property	1311
Options Property	1311
Methods	1312
AddField Method	1314
Assign Method	1315
Clear Method	1316
DeleteField Method	1317
DeleteFields Method	1317
LoadFromFile Method	1318
LoadFromStream Method	1318
SaveToFile Method	1319
SaveToStream Method	1320
Types	1320
TVirtualTableOptions Set	1321
Enumerations	1321
TVirtualTableOption Enumeration	1321

Index

0

1 What's New

New Features in UniDAC 11.0

- Added support for RAD Studio 13 Florence
- Added support for Lazarus 4.2
- Added support for system proxy settings to the TProxyOptions class for HTTP tunnelling
- Added the Pooled function to the TCustomDACConnection component
- Added support for ReportBuilder 23 in the demo project

Oracle data provider

- Added support for the BOOLEAN data type in Direct mode for Oracle 23 and later
- Added support for passwords longer than 30 characters in Direct mode for Oracle 23 and later
- Added support for Bearer Token authentication when connecting through an HTTP tunnel
- Added the SelfEvents specific option to TUniAlerter for enabling or disabling receiving your own events

SQL Server data provider

- Added support for Bearer Token authentication when connecting through an HTTP tunnel
- Added the MSOLEDBEncryptType connection option for the MSOLEDB provider
- Added the UseNoBrowseTable connection option to prevent calling "SET NO_BROWSETABLE ON"
- Added support for the MERGE batch operation (MSSQL Server provider)

MySQL data provider

- Added support for MariaDB 12
- Added support for the UUID data type in MariaDB 10.7 and later
- Added support for the INET4 and INET6 data types in MariaDB 10.10 and later
- Added support for Bearer Token authentication when connecting through an HTTP tunnel
- Added the InsertBatchSize option to the TUniDump component to set the size of the

SQLInsert query

- Added the AnsiQuotesMode option to set the ANSI_QUOTES value for the sql_mode variable
- Improved trigger backup with TUniDump

PostgreSQL data provider

- Added support for Bearer Token authentication when connecting through an HTTP tunnel
- Improved support for array data types
- Improved connection opening performance

Redshift data provider

- Added support for Bearer Token authentication when connecting through an HTTP tunnel

InterBase data provider

- Added the SQLState property to EUniError.InnerException to store the SQLSTATE completion code
- Improved the format of the DEFAULT_VALUE column in TIBCMetaData (when MetaKind = Column) to return values in clear text representation

SQLite data provider

- Added support for the ISO8601-like time format for the TimeFormat property of the LiteConnection component

DBF data provider

- Improved support of the HiPerSix database format

Access data provider

- Significantly improved the performance of DML operations executed within a transaction
- Added the CacheSize and CacheLifetime options to improve performance in ExclusiveLock mode
- Added the PartialIndex option to improve SELECT performance

NexusDB data provider

- Added support for NexusDB 4.75.17
- Significantly improved support for filtering when ServerCursor = True

New Features in UniDAC 10.4

- Added support for RAD Studio 12 Athens Release 3
- Added support for the RAD Studio 64-bit IDE
- Added support for Lazarus 3.8
- Added support for the JSON file format in the LoadFromFile method of the VirtualTable component
- Added support for the TurboPack SynEdit package in component editors
- Added support for syntax highlighting in component editors in Lazarus on the Windows platform
- Added support for parameters in the Script component
- Added the IndentFormat argument to the SaveToXML method of the dataset to decrease file size
- Added the SaveToJSON method for serializing a dataset to the JSON format
- Added the ability to use readonly fields in the SQL Generator at design-time
- Improved work with design-time editors in the High DPI mode

Oracle data provider

- Added support for auto-login wallets
- Added support for the BOOLEAN data type in Oracle 23 and higher
- Added support for parameters in the Script component
- Improved query performance for obtaining a primary key
- Improved detection of the MERGE statement type in the Direct mode

SQL Server data provider

- Added the Charset-specific connection option for the Direct mode

MySQL data provider

- Added support for MySQL 9
- Added support for password-protected private key for SSL protocol
- Added the EMySQLNetException class for internal errors in the MySQL protocol

PostgreSQL data provider

- Added support for PostgreSQL 17
- Added support for the COPY ... TO STDOUT statement
- Added support for password-protected private key for SSL protocol

ASE data provider

- Added support for connection via the SSL protocol
- Added support for password-protected private key for SSL protocol

New Features in UniDAC 10.3

- Added support for RAD Studio 12 Athens Release 2
- Added support for C++Builder 64-bit Windows (Modern) Application Development
- Added support for Lazarus 3.6
- Added provider for Microsoft Excel
- Added C++Builder demo components for working with SecureBridge

SQLServer data provider

- Added support for Variant OUT parameters in Direct mode
- Added datetime format when using timestamp macros

MySQL data provider

- Added the WriteBOM specific option for TUniDump

PostgreSQL data provider

- Improved performance
- Added support for the pgvector extension
- Added the UseGeneratedColumns specific option for TUniTable, TUniQuery,

TUniStoredProc

- Added the WriteBOM specific option for TUniDump

SQLite data provider

- Added the EnableAutoInc specific option for UniTable and UniQuery components
- Added the WriteBOM specific option for TUniDump

NexusDB data provider

- Added support of NexusDB 4.75.10
- Improved performance of working with stored procedures

Access data provider

- Added support for accessing Select and Union queries as views in the Direct mode

New Features in UniDAC 10.2

- Now Release 1 for RAD Studio 12 Athens, Delphi 12 Athens, and C++Builder 12 Athens is required
- Added support for Lazarus 3.2
- Added voResetAutoInc in the VirtualTable component that allows to reset AutoInc values on Clear

SQLServer data provider

- Microsoft Entra Service Principal authentication (auADServicePrincipal) in the prMSOLEDB provider is supported

PostgreSQL data provider

- Improved work with generated fields

NexusDB data provider

- Improved the Locate and FindKey methods' execution when using ServerCursor

SQLite data provider

- Now the Direct mode is based on version 3.45.2 of the SQLite engine

- Added support for automatic detection of transaction state when the transaction is controlled by an SQL statement

BigCommerce, Dynamics, HubSpot, Magento, NetSuite, Salesforce, Salesforce Marketing Cloud, SugarCRM, and ZohoCRM data providers

- Added metadata cache

New Features in UniDAC 10.1

- Lazarus 3.0 is supported

Oracle data provider

- Added support for connect to servers with enabled Security Banners in the Direct mode
- Improved error message about a non-existent database object in another schema

PostgreSQL data provider

- Improved work with column default values
- Improved work with RefreshRecord queries
- Improved work with arrays in queries

InterBase data provider

- Added support for Firebird 5
- Added support for multiple rows being returned by DML with the RETURNING clause

SQLite data provider

- The RealAsDouble option for the Connection component is added

DBF data provider

- Added support for the UTF8 encoding
- Added support for tables with duplicate column names
- Improved compatibility with Quantum GIS
- Improved error handling when using the IgnoreDataErrors option

NexusDB data provider

- Added support of NexusDB 4.75.01

MS Access data provider

- Added support for Direct mode

DB2 data provider

- Added support SSL connection

New Features in UniDAC 10.0

- Added support for RAD Studio 12
- Added support for macOS Sonoma
- Added support for iOS 17
- Added support for Android 13
- 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
- Added support for DBMonitor in the VirtualQuery component
- Added support for storing AutoInc fields when saving TVirtualTable to XML
- Added support of NexusDB 4.75.01

Oracle data provider

- Added support for Oracle 23c
- Improved UnicodeEnvironment support for non-Unicode Delphi versions

SQLServer data provider

- Added support for SQL Server 2022
- Added support for LastInsertID
- Added DisableConstraints specific option for the Dump component
- Added AddDateTimeFormat specific option for the Dump component

MySQL data provider

- Added support for MariaDB 11
- Added support for mysql_clear_password authentication mechanism

PostgreSQL data provider

- Added support for PostgreSQL 16
- Added support for PREPARE/EXECUTE commands
- Added several specific options for the Dump component

InterBase data provider

- Added support for iOS Simulator ARM 64-bit target platform

SQLite data provider

- Now the Direct mode is based on version 3.42.0 of the SQLite engine

DBF data provider

- Performance of DML operations is significantly improved
- Added support for CREATE TABLE IF NOT EXISTS and DROP TABLE IF EXISTS statements
- Added support for the CREATE TABLE AS SELECT statement

NexusDB data provider

- Added support of NexusDB 4.75.01

MongoDB data provider

- Added support for MongoDB 7
- Added support for MongoDB 6
- Added support for empty database name when establishing connection
- Improved compatibility with Linked Server in MSSMS

ODBC data provider

- Information from SQLGetInfo is added to ODBCMetaData

New Features in UniDAC 9.4

- Added support for RAD Studio 11 Alexandria Release 3
- Added support for iOS Simulator ARM 64-bit target platform
- Added support for Lazarus 2.2.6
- Added support for the YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, GETDATE, DATE, TIME, TRIM, TRIMLEFT, TRIMRIGHT statements in TDADataset.Filter
- Added support for the mathematical operations in TDADataset.Filter
- Added support for Aggregate Fields and InternalCalc Fields
- Added ability to restore from file with TEncoding via the Dump component
- Improved the execution of statement that have either IS NULL or IS NOT NULL in the WHERE clause with TVirtualQuery
- Now the SetRange will function according to the case sensitivity of keywords in IndexFieldNames
- Fixed bug with AV on using master/detail relationship with VirtualTable
- Fixed bug with loading data that already has persistent fields into VirtualTable

Oracle data provider

- Improved detection of home directories in recent versions of Oracle
- Now valid exception will be raised instead of AV when memory can't be allocated for the large row count
- Fixed bug with LOBs reading in the Direct mode
- Fixed bug with reading BFLOAT and BDOUBLE values from ANYDATA
- Fixed bug with querying record count if SQL statement contains PIVOT

SQLServer data provider

- Improved compatibility with Linked Server in MSSMS
- Improved the GetOrderBy method behavior
- Fixed bug with Unicode (UTF-8) in the Direct mode for Windows
- Fixed bug with an assertion failure in the CachedUpdate mode
- Fixed bug with connecting to default instance by prSQL(SQLOLEDB.1) provider

- Fixed bug with fetching datetimeoffset fields in the Direct mode
- Fixed bug with fetching sql_variant fields in the Direct mode
- Fixed bug with preparing a stored procedure in the Direct mode

MySQL data provider

- Fixed bug with BLOBs for MySQL version below 4.1
- Fixed bug with BIGINT UNSIGNED key fields in version 4.1 or higher
- Fixed bug with BIT field default values

PostgreSQL data provider

- Added support for PostgreSQL 15
- Improved process of getting extended fields info
- Fixed bug with extra connections in GetType methods
- Fixed bug with TIMESTAMPTZ fields
- Fixed bug with reading fields of type REAL using the Auto protocol
- Fixed bug with writing Bytea type in Batch operations
- Fixed bug with using "ON CONFLICT RETURNING" in batch operations
- Fixed problem with handling fields of type REAL
- Fixed bug with the "Range Check Error" exception when reading BLOB data

InterBase data provider

- Improved work with alias
- Fixed bug with "The SQL statement is not allowable for a bulk operation" when using SP in batch operations
- Fixed bug with "Validation error" when using the Loader component

SQLite data provider

- Added support for GUID fields stored in binary format
- Improved reading fields of the BLOB family

DBF data provider

- Improved ROUND() function
- Fixed bug with a field name case when creating a table

NexusDB data provider

- Added support for NexusDB 4.70.01
- Added support for working with tables protected by a password

ODBC data provider

- Added the UuidWithBraces specific option for the Connection component

MS Access data provider

- Fixed bug with "Cannot modify a read-only dataset"

MongoDB data provider

- Added support for MongoDB Atlas
- Added support for DNS Seed List connection format
- Fixed bug with reading international characters when UseUnicode option is set

New Features in UniDAC 9.3

- 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 the CloneCursor method for Query and Table components that allows sharing data between datasets
- Added support of standard collations in a SQL statement in the VirtualQuery component
- Improved the performance of exporting to XML
- Fixed bug when a connection string parameter value contains a single quote

SQLServer data provider

- Added support for Microsoft OLE DB Driver 19 for SQL Server

- Added support for Azure AD authentication support for the Microsoft OLE DB provider
- Added support for Always On availability group in the Direct mode
- Added support for connection to the Azure database by using the "Redirect" connection policy in the Direct mode
- Fixed bug with restricting the list of procedures using PROCEDURE_TYPE condition in Metadata component
- Fixed bug with committing transaction on SQL Server 2000 in Direct mode

MySQL data provider

- Fixed bug with "Record changed by another user" error when LockMode <> ImNone and using encryption
- Fixed bug with truncation of milliseconds of DateTime fields in the Dump component
- Fixed bug with posting calculated field

PostgreSQL data provider

- Added support for PostGIS
- Improved TimeZoneOffset support for TPgTimeStamp
- Improved the Truncate method of the TPgLargeObject class is improved
- 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 specific option 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 with international characters in error messages

InterBase data provider

- Improved support for TIME/TIMESTAMP WITH TIME ZONE data types
- Added support for EXTENDED TIME/TIMESTAMP WITH TIME ZONE data types

- Added support for working with time zone data types when clients don't have the ICU library
- Added support for The `isc_dpb_session_time_zone`, `isc_dpb_set_bind`, `isc_dpb_decfloat_round`, and `isc_dpb_decfloat_traps` DPB
- Fixed bug with a "Too many Contexts of Relation/Procedures/Views" error when using "UPDATE OR INSERT" batch operations
- Fixed bug with using CAST in the SQL statement when UseUnicode set to True
- Fixed bug with reading GUID data of stored procedure in using the AsGuid property
- Fixed bug with processing GUID data in using in parameter
- Fixed bug with working with the transaction when LockMode <> ImNone
- Fixed bug with using the TUniAlerter component in WebBroker applications
- Fixed bug with "Invalid variant type" error when the DMLRefresh property is set to True

SQLite data provider

- Now the Direct mode is based on version 3.39.2 of the SQLite engine
- Fixed bug with mapping table columns of non-standard types to fields of the ftMemo type
- Fixed bug with mapping table columns of integer types to string fields

DBF data provider

- Added support for VisualFoxPro CANDIDATE indexes

NexusDB data provider

- Added support for NexusDB 4.60.01
- Added option to set Remote Thread Priority

MS Access data provider

- Fixed bug with processing UUID fields

New Features in UniDAC 9.2

- RAD Studio 11 Alexandria Release 1 is supported
- Lazarus 2.2.0 is supported
- Windows 11 is supported

- macOS Monterey is supported

PostgreSQL data provider

- The AddDelete specific option 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

SQLite data provider

- Now the Direct mode is based on version 3.37.2 of the SQLite engine

SQL Server data provider

- Passing the "client_interface_name" parameter to the server in the Direct mode is added

New Features in UniDAC 9.1

- The PoolId connection pool option is added

DBF data provider

- FSIZE SQL function is supported
- Autoinc data type for dBase tables is supported

PostgreSQL data provider

- PostgreSQL 14 is supported
- OUT parameters in stored procedures for PostgreSQL 14 are supported

InterBase data provider

- The WireCompression option for the Connection component is added

Nexus DB data provider

- Block fetch is supported
- Fetch performance is improved

New Features in UniDAC 9.0

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

InterBase data provider

- Firebird 4 is supported

SQLite data provider

- Added the IntegerAsLargeInt option for the Connection component

NexusDB provider

- NexusDB 4.50.27 is supported

Google BigQuery data provider

- Added the Google BigQuery provider

HubSpot data provider

- Added the HubSpot provider

New Features in UniDAC 8.4

- RAD Studio 10.4.2 Sydney is supported
- macOS 11 Big Sur is supported
- iOS 14 is supported
- Android 11 is supported
- Performance of batch operations is improved
- Performance of the FindFirst, FindNext, FindLast, and FindPrior methods is improved
- The UseUnicode option in the VirtualQuery component is added

Oracle data provider

- Oracle 21c is supported

- The PrefetchRows option in the Direct mode is supported
- Data fetch performance in the Direct mode is improved
- LOB read/write performance is improved

SQLServer data provider

- LOB read/write performance in the Direct mode is improved

PostgreSQL data provider

- PostgreSQL 13 is supported
- Work in a multi-threaded environment through a single connection is supported
- The MultipleConnections option in the Connection component is added

InterBase data provider

- Over-the-Wire (OTW) encryption is supported
- Automatic detection of computed fields when generating update statements is improved
- Memory consumption in batch operations is reduced

SQLite data provider

- The LockingMode specific option in the Connection component is added
- The Synchronous specific option in the Connection component is added
- The JournalMode specific option in the Connection component is added
- Performance with default values of the new options is significantly improved

ASE data provider

- The TextSize specific option is added

DBF data provider

- The IgnoreIndexErrors specific option in the Connection component is added
- Performance of reading and writing MEMO and BLOB values is improved
- Work with dBaseV and dBaseVII tables when the DBFFormat option set to dfAuto is improved

ODBC data provider

- Data fetch performance is improved
- LOB read/write performance is improved

New Features in UniDAC 8.3

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

Oracle data provider

- Oracle 20c is supported
- Connection via SSL protocol is supported
- Connection via SSH protocol is supported
- Connection via HTTP tunnel is supported

SQL Server data provider

- SQL Server 2019 is supported

DBF data provider

- Native dBase functions in a SQL statement are supported

ODBC data provider

- Error message retrieving from SQL Anywhere ODBC driver is improved

New Features in UniDAC 8.2

- RAD Studio 10.4 Sydney is supported
- Lazarus 2.0.8 is supported
- macOS 64-bit in Lazarus is supported

Oracle data provider

- Mapping the FLOAT Oracle data type to the ftNumber field is added

PostgreSQL data provider

- The Line geometric type is supported

DBF data provider

- The AllFieldsAsNullable specific option is added

NexusDB data provider

- Support for the Pipe protocol is added
- Support for the Secure Pipe protocol is added
- Support for the Secure TCP protocol is added
- Now an #INMEM alias is created automatically when it is specified in the Database property

New Features in UniDAC 8.1

- Android 64-bit is supported
- Lazarus 2.0.6 is supported
- Now Trial edition for macOS and Linux is fully functional

Oracle data provider

- Oracle 19c is supported
- Long database object names is supported

SQLServer data provider

- TLS 1.2 support in the Direct mode is added
- The connection option MultiSubnetFailover for the MSOLEDB provider is added
- Use of the Server property that contains Port in the Direct mode is added

MySQL data provider

- OpenSSL 1.1 library is supported

PostgreSQL data provider

- PostgreSQL 12 is supported
- OpenSSL 1.1 library is supported

Interbase data provider

- Interbase 2020 is supported
- Improved performance when using pooling

MongoDB data provider

- The LowerCaseObjectId specific option for the Connection component is added

DBF data provider

- The IdentifierCase specific option is added
- The cmUnsafe value for the ConnectMode specific option is added

New Features in UniDAC 8.0

- 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

New Features in UniDAC 7.5

- Lazarus 2.0.2 is supported
- The DefaultSortType property for TVirtualTable is added
- Performance of the SaveToFile/LoadFromFile methods of TVirtualTable is significantly increased

New Features in UniDAC 7.4

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

Oracle data provider

- Oracle 18c is supported
- Implicit result sets in Oracle 12 are supported

SQLServer data provider

- QuoteNames option in TUniLoader to escape field names is added

MySQL data provider

- Support for PAM and Windows authentications is added

InterBase data provider

- Possibility to write large blobs by pieces is added

PostgreSQL data provider

- PostgreSQL 11 is supported

SQLite data provider

- Support for the BreakExec method in the Query component is added

DBF data provider

- Detection of the file format when the DBFFormat option is set to dfAuto is improved
- Work with databases which contain a large number of files is improved

BigCommerce data provider

- OAuth authentication is supported

New Features in UniDAC 7.3

- Lazarus 1.8.4 is supported
- Performance of batch operations is improved
- Demo projects for IntraWeb 14 are added
- AutoOpenSources option for TVirtualQuery is added
- OfflineMode option for TVirtualQuery is added

Oracle data provider

- Now non-compiled stored procedures can be described in the Direct mode
- Performance of data fetching in the Direct mode is improved
- Performance of describing stored procedures in the Direct mode is improved
- Support for TIMESTAMP WITH TIMEZONE in the Direct mode is improved

SQLServer data provider

- MARS in TDS is supported
- NonBlocking mode in TDS is supported
- Query notifications in TDS are supported

MySQL data provider

- MySQL 8 is supported
- Support for sha2_password, caching_sha2_password authentications is added

InterBase data provider

- Now the "Data type is not supported" exception is not raised by the Query component when the DescribeParams property is set to True

PostgreSQL data provider

- Support for HTTP/HTTPS tunnel is added

SQLite data provider

- WAL in the Direct Mode for non-Windows platforms is supported

ASE data provider

- Retrieving the OUTPUT parameters is improved

MongoDB data provider

- The Decimal128 data type is supported
- Precompiled MongoDB client libraries are included in the Professional Edition
- Performance of fetching large documents is improved

DBF data provider

- Support for Clipper/Harbour is added
- Support for native indexes based on complex expressions is added
- Compatibility with Codebase is improved

ExactTarget data provider

- App center client authentication is supported

FreshBooks data provider

- FreshBooks new version is supported

Magento data provider

- Magento version 2.x is supported

NetSuite data provider

- Sandbox is supported

ZohoCRM data provider

- Domain is supported

New Features in UniDAC 7.2

- Lazarus 1.8 and FPC 3.0.4 are supported
- Support for custom constraints is added
- The UseBlankValues property for the Loader component is added

Redshift data provider

- Amazon Redshift provider is added

SQLServer data provider

- Windows authentication in the Direct mode is supported

MySQL data provider

- Support for backup/restore of triggers and stored procedures is added

InterBase data provider

- Loading of the default client library for 64-bit applications is improved

SQLite data provider

- Direct Mode in Lazarus is supported
- BIT type is supported

- The UnknownAsString dataset specific option that allows mapping fields of unknown type as ftString instead of ftMemo is added

DBF data provider

- Direct Mode in Lazarus is supported
- The IndexOnReading connection specific option that allows using local indexes on reading data is added

DB2 data provider

- Compatibility with DB2 version 11 is improved

New Features in UniDAC 7.1

- The performance of TVirtualQuery is significantly improved
- Application-defined functions in TVirtualQuery are supported
- Application-defined collations in TVirtualQuery are supported
- AutoInc fields in TVirtualTable are supported

Cloud data providers

- BigCommerce provider is added
- Dynamics CRM provider is added
- FreshBooks provider is added
- Magento provider is added
- MailChimp provider is added
- NetSuite provider is added
- QuickBooks provider is added
- Salesforce provider is added
- Salesforce Marketing Cloud provider is added
- SugarCRM provider is added
- Zoho CRM provider is added

Oracle data provider

- Oracle 12c connection modes (SYSBACKUP, SYSDG, SYSKM) in the Direct mode are supported
- OS authentication in the Direct mode is supported
- NChar literal replacement is supported
- CLOB parameters behavior when UnicodeEnvironment=True is improved

MySQL data provider

- Azure Database for MySQL is supported
- JSON data type is supported

InterBase data provider

- Support for Firebird on Android platform is added
- Support for Firebird 3 packages is added
- Aliases handling in the RETURNING clause is supported
- The WireCompression connection parameter for Firebird 3 is supported

PostgreSQL data provider

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

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.20.0
- Custom SQL aggregate functions are supported

DBF data provider

- The CodePage specific options are added
- The ConnectMode specific options are added

DB2 data provider

- The DECFLOAT data type is supported

New Features in UniDAC 7.2

- Lazarus 1.8 and FPC 3.0.4 are supported
- Support for custom constraints is added
- The UseBlankValues property for the Loader component is added

Redshift data provider

- Amazon Redshift provider is added

SQLServer data provider

- Windows authentication in the Direct mode is supported

MySQL data provider

- Support for backup/restore of triggers and stored procedures is added

InterBase data provider

- Loading of the default client library for 64-bit applications is improved

SQLite data provider

- Direct Mode in Lazarus is supported
- BIT type is supported
- The UnknownAsString dataset specific option that allows mapping fields of unknown type as ftString instead of ftMemo is added

DBF data provider

- Direct Mode in Lazarus is supported
- The IndexOnReading connection specific option that allows using local indexes on reading data is added

DB2 data provider

- Compatibility with DB2 version 11 is improved

New Features in UniDAC 7.1

- The performance of TVirtualQuery is significantly improved
- Application-defined functions in TVirtualQuery are supported

- Application-defined collations in TVirtualQuery are supported
- AutoInc fields in TVirtualTable are supported

Cloud data providers

- BigCommerce provider is added
- Dynamics CRM provider is added
- FreshBooks provider is added
- Magento provider is added
- MailChimp provider is added
- NetSuite provider is added
- QuickBooks provider is added
- Salesforce provider is added
- Salesforce Marketing Cloud provider is added
- SugarCRM provider is added
- Zoho CRM provider is added

Oracle data provider

- Oracle 12c connection modes (SYSBACKUP, SYSDG, SYSKM) in the Direct mode are supported
- OS authentication in the Direct mode is supported
- NChar literal replacement is supported
- CLOB parameters behavior when UnicodeEnvironment=True is improved

MySQL data provider

- Azure Database for MySQL is supported
- JSON data type is supported

InterBase data provider

- Support for Firebird on Android platform is added
- Support for Firebird 3 packages is added

- Aliases handling in the RETURNING clause is supported
- The WireCompression connection parameter for Firebird 3 is supported

PostgreSQL data provider

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

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.20.0
- Custom SQL aggregate functions are supported

DBF data provider

- The CodePage specific options are added
- The ConnectMode specific options are added

DB2 data provider

- The DECFLOAT data type is supported

New Features in UniDAC 7.0

- 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

Oracle data provider

- Oracle Encryption in the Direct mode is supported
- Oracle Data Integrity in the Direct mode is supported
- Oracle Cloud (DBaaS) in the Direct mode is supported
- Oracle 12c authentication in the Direct mode is supported
- SECUREFILE in the Direct mode is supported
- Prefetch LOBs for Oracle 11 and higher is supported
- EDITIONABLE and NONEDITIONABLE clause is supported

- The PrefetchLobSize option is added
- Now the Direct mode is based on the SQLite engine version 3.17.0
- Field size detecting for servers with multi-byte charset when UseUnicode=False is improved
- Now NUMBER data type without fixed scale has precision=39 and scale=39 instead of 38

Interbase data provider

- Possibility to manage batch operations using a transaction is added
- Possibility to obtain active transaction number using DBMonitor is added

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.17.0

NexusDB data provider

- Support for using ConnectionString is added
- Support for using the TfmtBCD fields is added
- Support for the SmartFetch mode is improved

MongoDB data provider

- New MongoDB provider is added

DBF data provider

- Direct mode is supported

New Features in UniDAC 6.4

- TVirtualQuery component is added
- TDADatasetOptions.InsertAllSetFields property is added

SQL Server data provider

- Support for IPv6 protocol in Direct Mode is added

New Features in UniDAC 6.3

- RAD Studio 10.1 Berlin is supported
- Lazarus 1.6 and FPC 3.0.0 is supported
- Support for the BETWEEN statement in TDADataset.Filter is added
- Performance of TDALoader on loading data from TDataSet is improved

Oracle data provider

- Transactions behavior when AutoCommit is disabled now is the same as in ODAC

SQLServer data provider

- Direct mode in TUniLoader is supported
- SmartFetch mode in Disconnected mode is supported

MySQL data provider

- Support for utf8mb4 charset is added
- SmartFetch mode in Disconnected mode is supported

PostgreSQL data provider

- PostgreSQL 9.5 is supported
- A MessageCharset option in connection specific options is added

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.12.0
- Support for URI filenames is added

Adaptive Server Enterprise data provider

- Direct mode is supported
- macOS is supported
- iOS is supported
- Android is supported
- Specific option HostName was renamed to ClientHostName

ODBC data provider

- An ability to select ODBC Driver Manager is added

MS Access data provider

- Possibility to select a driver is added

New Features in UniDAC 6.2

- RAD Studio 10 Seattle is supported
- INSERT, UPDATE and DELETE batch operations are supported
- Now Trial for Win64 is a fully functional Professional Edition

Oracle data provider

- Support for Offset is added for DML arrays
- Support for OraNet.PacketSize is added to improve performance in VPN and Wireless networks
- Now NULL and empty strings are different values for ftOraLob and ftOraClob parameters

MySQL data provider

- MariaDB Embedded is supported

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.8.11.1
- The EnableSharedCache specific option of the Connection component for non-Windows platforms is added

New Features in UniDAC 6.1

- RAD Studio XE8 is supported
- AppMethod is supported
- The ParamCheck option behavior is fixed

Oracle data provider

- Direct mode in Lazarus is supported
- Now the Direct mode is supplied as source code

- Support for Objects in the Direct mode is added
- Support for EZCONNECT in the Direct mode is added
- Support for fields with Cursor data type in the Direct mode is added
- Now statements with RETURN INTO clauses can return RowsAffected in the Direct mode

SQL Server data provider

- Direct mode in Lazarus is supported
- Now the Direct mode is supplied as source code
- Performance of connection establishing in the Direct mode is improved
- The specific option "OLEDBProvider" is renamed to "Provider"

InterBase data provider

- Firebird 3 support is added
- Firebird 3 BOOLEAN column type support is added

PostgreSQL data provider

- PostgreSQL 9.4 support is added

SQLite data provider

- Direct mode for macOS, iOS and Android platforms is supported
- Database encryption for macOS, iOS and Android platforms is supported
- Now the Direct mode is based on the SQLite engine version 3.8.9

ODBC data provider

- ODBC provider for Lazarus is added for Unix platforms

New Features in UniDAC 6.0

SQL Server data provider

- Direct Mode is supported
- macOS is supported
- iOS is supported

- Android is supported

InterBase data provider

- The QueryRowsAffected dataset specific option is added for increasing performance of update operations

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.8.7.1

NexusDB data provider

- Nexus Embedded support is added

ASE data provider

- Ability to set CharSet is added

DB2 data provider

- Support for 64-bit client is added

New Features in UniDAC 5.5

- RAD Studio XE7 is supported
- Lazarus 1.2.4 is supported
- New free Express edition is added
- Providers are added to the Standard edition and now it doesn't require other DAC products installation
- Demo projects for FastReport 5 are added
- SpecificOptions names and values validation are added
- The TCustomDADataset.GetKeyFieldNames method is added
- The ConstraintColumns metadata kind for the TDAMetadata component is added

Oracle data provider

- RAC server support is improved
- Support for WITH FUNCTION clause for Oracle 12c is added

- The HideRowId option is added

InterBase data provider

- The OldTransactionBehaviour global variable is added

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.8.6

ODBC data provider

- Fetch performance is improved
- Now the VarBytesAsBlob specific option is replaced with the VarBinaryAsBlob and LongVarBinaryAsBlob specific options
- Information about TypeInfo is added to ODBCMetaData

New Features in UniDAC 5.3

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

Oracle data provider

- DataTypeMapping conversion from XMLType to ftString is added
- DataTypeMapping conversion from Interval to ftString is added

SQL Server data provider

- SQL Server 2014 is supported

InterBase data provider

- TUniTransaction.OnCommitRetainig and TUniTransaction.OnRollbackRetainig events are added

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.8.4.3

ASE data provider

- The PrepareMethod option is added

New Features in UniDAC 5.2

- iOS in C++Builder XE5 is supported
- RAD Studio XE5 Update 2 is now required
- Now .obj and .o files are supplied for C++Builder
- Compatibility of migrating floating-point fields from other components is improved

Oracle data provider

- An ability to establish OCI and Direct connections in the same application is supported
- New Oracle 12c connection modes are added (SYSBACKUP, SYSDG, SYSKM)

SQLite data provider

- Direct mode for x64 platform is supported

New Features in UniDAC 5.1

- RAD Studio XE5 is supported
- Application development for Android is supported
- Lazarus 1.0.12 is supported
- Automatic checking for new versions is added
- Flexible management of conditions in the WHERE clause is added
- The possibility to use conditions is added
- Performance is improved
- IPv6 protocol support is added
- Migration from FIBPlus is added
- The possibility to use ranges is added

- The AutoCommit property for the Connection component 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
- 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
- ConnectionTimeout is now used when disconnecting after connection loss

Oracle data provider

- The UROWID data type is supported in the Direct mode

SQL Server data provider

- The CursorType specific option is added

MySQL data provider

- MariaDB is supported

InterBase data provider

- Now Params specific option values for TUniTransaction can be separated by a semicolon
- The ForceUsingDefaultPort global variable is added

PostgreSQL data provider

- PostgreSQL 9.3 is supported

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.8.0
- The AutoCommit and AutoCommitRowCount TUniLoader specific options

ODBC data provider

- The DefaultStrParamSize specific option is added

- An option that allows fetching VarBytes as BLOB is added
- ConnectionTimeout is now used when disconnecting after connection loss

MS Access data provider

- The ForceCreateDatabase option is added

NexusDB data provider

- NexusDB 3.12 is supported

New Features in UniDAC 5.0

- 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
- Migration from PgDAC and LiteDAC is added to the Migration Wizard
- Migration from AnyDAC and FireDAC is added to the Migration Wizard

Oracle data provider

- BINARY_DOUBLE & BINARY_FLOAT data types support in the Direct mode is added

MySQL data provider

- SSL support in macOS is fixed

InterBase data provider

- Application development for iOS using InterBase XE3 ToGo Edition is supported
- The DefaultTransaction property in TUniConnection is added
- The Params specific option in TUniTransaction is added

PostgreSQL data provider

- Now ErrorCode indicates a socket error code when a connection error appears
- SSL support in macOS is fixed

SQLite data provider

- Now the Direct mode is based on the SQLite engine version 3.7.16.2
- Now SQLite string data type without length is mapped as ftMemo instead of ftString
- Converter from Unix and Julian data formats to ftDateTime is added

ASE data provider

- The EncryptPassword option is added
- The DetectFieldsOnPrepare option is added

DB2 data provider

- XML fields support is added

New Features in UniDAC 4.6

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

SQLServer data provider

- The Port specific option that allows specifying the port number for connection is added

New Features in UniDAC 4.5

- Rad Studio XE3 is supported
- Windows 8 is supported

New Features in UniDAC 4.2

- Update 4 Hotfix 1 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
- Data Type Mapping support is added
- Data Encryption in a client application is added

- The TMSDecryptor component for data encryption is added
- Calling of the TCustomDASQL.BeforeExecute event is added

New Features in UniDAC 4.1

- Update 4 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
- macOS and iOS in RAD Studio XE2 is supported
- FireMonkey support is improved
- Lazarus 0.9.30.4 and FPC 2.6.0 are supported
- macOS in Lazarus is supported
- Linux x64 in Lazarus is supported
- FreeBSD in Lazarus is supported

Oracle data provider

- Oracle 11 Express Edition is supported
- Support for the NonBlocking option is added
- The QueryResultOnly option is added to TOraChangeNotification

PostgreSQL data provider

- PostgreSQL 9.1 is supported

SQLite data provider

- DateFormat and TimedFormat specific options are added in the SQLite data provider

NexusDB data provider

- Support of NexusDB 3.09 is added

New Features in Universal Data Access Components 4.00

- 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

MS Access data provider

- Exclusive access to databases in MSAccess provider is added

Adaptive Server Enterprise data provider

- Ability to set ApplicationName in the ASE provider is added
- The AnsiNull option in the ASE provider is added

New Features in Universal Data Access Components 3.70

- Lazarus 0.9.30 and FPC 2.4.2 is supported
- New DBF provider is added

Oracle data provider

- Oracle 9, Oracle 10, and Oracle 11 authentication in the Direct mode is supported
- Case sensitive login and password in the Direct mode is supported
- Unicode login and password in the Direct mode is supported
- Client Identifier in the Direct mode is supported
- Support of BLOB, CLOB, and NCLOB data types in TUniLoader is improved

PostgreSQL data provider

- Application Name connection option is supported
- Payload parameter for PostgreSQL notification is supported

SQL Server data provider

- Support for SQL Server Compact Edition 4.0 is added

SQLite data provider

- User-defined function for SQLite provider is supported
- Default UniNoCase collation for SQLite provider is added (the DefaultCollations specific option)
- Interface user-defined collation registration for SQLite provider is improved
- SQLite source version is fixed (missing .inc file is added)

Adaptive Server Enterprise data provider

- Support for the AnsiNull option is added

New Features in Universal Data Access Components 3.60

- NexusDB provider
- PostgreSQL 9.0 supported
- Improved performance in the PostgreSQL provider
- Encryption support in the SQLite provider
- Support for connection with using Service Name in the Direct mode in the Oracle provider
- Support for ASCII databases in the SQLite provider (the ASCIIDataBase specific option)

New Features in Universal Data Access Components 3.50

- Embarcadero RAD Studio XE supported
- TUniAlerter component
- Collation and UTF sorting support in the SQLite provider
- Support for dbMonitor 3
- Support for extended SQL for MS Access (set the ExtendedAnsiSQL specific option to 1)
- Support of ONLY lexeme in the FROM statement for PostgreSQL
- Ability to lock records in the CachedUpdate mode
- Ability to use Access system database added
- Ability to send call stack information to the dbMonitor component

- Now setting the SetFieldsReadOnly option to False makes all fields not readonly

New Features in Universal Data Access Components 3.00

- DB2, Microsoft Access, Advantage Database Server, Adaptive Server Enterprise, and other databases (using ODBC provider) support added
- Embarcadero RAD Studio 2010 supported

New Features in Universal Data Access Components 2.70

- SQLite support added

New Features in Universal Data Access Components 2.50

- **Unified SQL support**

Unified SQL allows to write truly database-independent SQL code. Unified SQL includes:

- *Macros* - in Unified SQL macros can evaluate to a different value depending on the provider used by the TUniConnection component.
 - *If* - for the purpose of extra flexibility Unified SQL supports conditional inclusion of SQL code into resulting statements using {if} directive. This allows to set different SQL for different DBMS.
 - *Functions* - introduce standard for calling common SQL functions. In run time function is transformed either to the corresponding native function, or to an equivalent expression.
 - *Literal* - provides universal syntax for date, time, and timestamp literals.
- **TUniLoader component**

serves for fast loading of data to the database. For each type of database server TUniLoader uses its specific interfaces for loading with maximum speed. For example, Oracle Direct Path Load interface is used for Oracle.

- **TUniDump component**

serves to store data from tables or editable views as a script and to restore data from a received script.

- **TUniConnection.AssignConnect method**

shares physical connection between several TUniConnection components

- Added support for Free Pascal under Linux
- Added NoPreconnect property to TUniScript for executing CONNECT and CREATE DATABASE commands
- Added DMLRefresh support in the PostgreSQL provider

New Features in Universal Data Access Components 2.00

- PostgreSQL support added

New Features in Universal Data Access Components 1.20

- Delphi 2009 and C++Builder 2009 supported
- Extended Unicode support for Delphi 2007 added (special Unicode build)
- Free Pascal 2.2 supported
- Powerful design-time editors implemented in Lazarus
- Completed with more comprehensive structured Help

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2 General Information

This section contains general information about Universal Data Access Components

- [Overview](#)

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.1 Overview

Universal Data Access Components (UniDAC) is a powerful library of nonvisual cross-database data access components for Delphi, C++Builder, and Lazarus (Free Pascal). The UniDAC library is designed to help programmers develop faster and cleaner cross-database applications. UniDAC is a complete replacement for standard database connectivity solutions and presents an efficient native alternative to the Borland Database Engine and dbExpress for access to Oracle, SQL Server, MySQL, InterBase, Firebird, SQLite, DB2, Microsoft Access, Advantage Database Server, Adaptive Server Enterprise, DBF, NexusDB, and other databases (using ODBC provider), as well as various Cloud services.

UniDAC is based on the well-known Data Access Components from Devart such as [ODAC](#), [SDAC](#), [MyDAC](#), [IBDAC](#), [PgDAC](#) and [LiteDAC](#). We have joined the experience of long-term successful development into one great product which provides unified access to popular databases such as Oracle, Microsoft SQL Server, MySQL, InterBase, Firebird, SQLite, DB2, Microsoft Access, Advantage Database Server, Adaptive Server Enterprise, DBF, NexusDB and other databases (using ODBC provider).

The UniDAC library is actively developed and supported by Devart Team. If you have questions about UniDAC, send us an email at support@devart.com or visit our [forum](#).

Advantages of UniDAC

UniDAC is very convenient in setup and usage. It provides transparent server-independent interface for working with different databases. Selected database provider ensures the best way to perform operations on the server.

Universal Data Access

UniDAC provides transparent server-independent interfaces for working with different databases, and lets you change the client engine for specific server type just by changing single connection option. It means that you can easily switch between database servers in your cross-database UniDAC-based application.

Server-Aware Providers

UniDAC chooses the best way specific to the server to perform most operations. Every UniDAC data provider uses server-specific native connectivity. All operations with data are performed by providers automatically considering peculiarities of the selected database server.

Access Cloud Services

UniDAC allows developing applications that work with data stored in such Cloud services as: BigCommerce, Dynamics CRM, FreshBooks, Google BigQuery, HubSpot, Magento, MailChimp, NetSuite, Salesforce, Salesforce MC, SugarCRM, QuickBooks, Zoho CRM. For this, it is enough to use UniDAC ODBC provider with any Devart ODBC drivers for Clouds.

Optimized Code

The goal of UniDAC is to enable developers to write efficient and flexible database applications. The UniDAC 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 UniDAC interface retains compatibility with standard VCL data access components like BDE. Existing BDE-based applications can be easily migrated to UniDAC and enhanced to take advantage of server-specific features.

Development and Support

UniDAC is a cross-database connectivity solution that has been actively developed and supported. UniDAC comes with full documentation, demo projects, and fast (usually within one business day) technical support by the UniDAC development team. Find out more about how to get help or submit feedback and suggestions to the UniDAC development team in [Getting Support](#).

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

Key Features

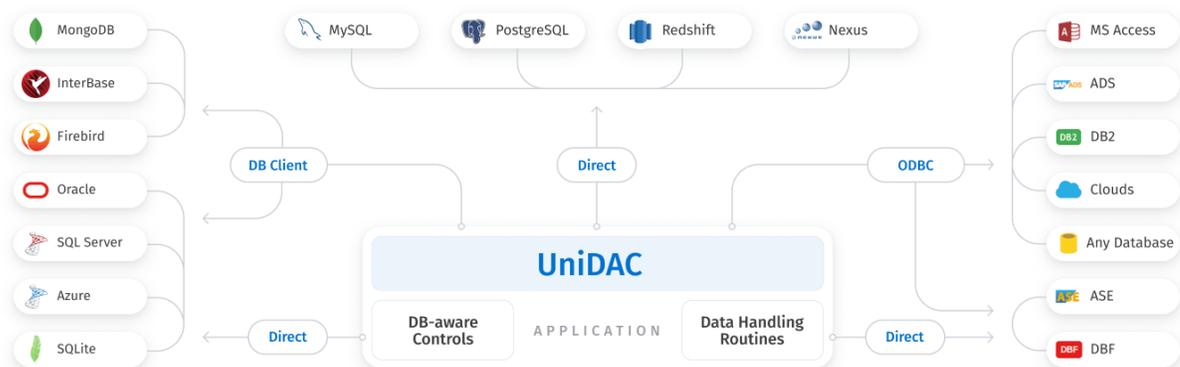
- Universal access to different database servers
- Support for most popular databases
- Full support for the latest server versions
- Support for the [latest IDE versions](#)
- VCL, LCL and FMX versions of library available
- High performance
- Easy to deploy
- Support of all standard and third-party data-aware controls
- Advanced connection management
- Flexible data updating
- [UniScript](#) component to execute scripts
- UniSQL for writing server-independent queries
- Ability of monitoring commands execution
- Advanced connection pooling
- Unicode and national char sets support
- Includes [database-independent data storage](#)
- [CachedUpdates](#) operation mode
- Local [sorting](#) and filtering by calculated and lookup fields
- [local master/detail](#) relationship
- Ability to [retrieve metadata information](#)

- Support for using [macros](#) in SQL
- Customizable [connection dialog](#)
- Advanced design-time editors
- A large amount of helpful [demo projects](#)
- Annual UniDAC [Subscription](#) with [Priority Support](#)
- Licensed royalty-free per developer, per team, or per site

The full list of UniDAC features are available in [Features](#).

How Does UniDAC Work?

UniDAC consists of two layers. The first layer is the general UniDAC Engine that provides the unified programming interface for the developer. The second layer is the data access layer, which consists of data access providers. These providers are intended for interacting between UniDAC Engine and database servers. Each data provider works with one specific database server. UniDAC structure overview is presented below:



UniDAC Structure Overview

There are two ways to install data access providers. The first way is to install the UniDAC Professional or UniDAC Standard edition. In this case all available providers are installed. The second way is to install UniDAC Engine with the UniDAC Express edition, and required data access providers with Data Access Components such as ODAC, SDAC, MyDAC, IBDAC, and PgDAC. Each DAC installs the corresponding data access provider for UniDAC. However, there is a slight difference between providers installed with UniDAC Professional

and providers installed with other DACs. Providers installed with UniDAC Professional include all server-specific functionality, while providers installed with DACs are just wrappers around DAC libraries. If both providers for a database server are installed, the provider installed with DAC will be used.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.2 Features

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)
- Access without using client library [Oracle, SQL Server, MySQL, PostgreSQL, SQLite, DBF]
- 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 support
- National charset support [Oracle, MySQL, InterBase, PostgreSQL]
- Unified SQL for writing server-independent queries

- Highly usable design time support
- Easy to deploy

Network and connectivity

- Disconnected Mode with automatic connection control for working with data offline
- Local Failover for detecting connection loss and implicitly reexecuting certain operations
- Ability to search for installed servers in a local network [SQL Server, MySQL, PostgreSQL]
- Connection timeout management [Oracle, SQL Server, MySQL, PostgreSQL, ODBC]
- Support for OS authentication
- Support for Proxy Authentication
- Support for the change expired password
- Support for both IPv6 and Ipv4 protocol

Compatibility

- Full support of the latest server versions
- Support for embedded server versions
- Compatible with Delphi 6, 7, C++Builder 6, Borland Delphi Studio 2006, Code Gear RAD Studio 2007, 2009, Embarcadero RAD Studio 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, Seattle, Berlin, Tokyo, Rio, Sydney, Alexandria
- Support for Lazarus 4.0 and FPC 3.2.2 for Windows, macOS and Linux (32-bit and 64-bit)
- Wide reporting component support, including support for InfoPower, ReportBuilder
- Support for 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

Server-specific features

Oracle

- Multiple Oracle Homes support
- Oracle sequence support

- Direct LOB access support
- Temporary LOB management routines
- Temporary LOBs for updating LOB fields
- OCI Connection Pooling, Proxy Session Pooling, and Statement Caching
- Oracle optimizer control
- CLIENT_IDENTIFIER support
- DBMS_ALERT support with the TUniAlerter component
- Secure connections with SSL, SSH, and HTTP tunneling
- Oracle package support
- Oracle 9i scrollable cursor support
- DML array operations support
- ProxySession support
- External Procedure support
- ROWID values retrieval
- Overloaded stored procedures support
- Support for WITH FUNCTION clause

SQL Server

- Possibility to change application name for a connection
- Possibility to change workstation identifier for a connection
- Configuration of OEM/ANSI character translation
- Enhanced support for SQL Server Compact Edition
- Enhanced support for User-defined Types of SQL Server
- Ability to lock records and tables

MySQL

- HANDLER syntax support
- Transaction isolation level support

- Possibility to retrieve last auto-incremented value
- Session identifier retrieval
- Server object information retrieval
- Row-level and table-level locking support
- Secure connections with SSL, SSH, and HTTP tunneling

InterBase/Firebird

- Advanced BLOB support
- Streaming (non-caching) BLOB access support
- Advanced generator support
- Advanced support for the character set OCTETS
- Support for the Firebird 2 EXECUTE BLOCK syntax
- Support for the Firebird 2 RETURNING clause
- Advanced locking for Firebird 2
- Automatic updates by DB_KEY unique field for Firebird 2
- Multiple transactions support with the TUniTransaction component
- InterBase events support with the TUniAlerter component
- Comprehensive array data type support
- Default value support for stored procedures
- InterBase services components for configuring server parameters and security
- Support for the Firebird 3 BOOLEAN datatype
- Support for the Firebird 2.1 trusted authentication
-
- Support for InterBase OTW encryption

PostgreSQL

- Advanced sequences support
- Advanced Large Objects support

- Ability to control Fetch block size
- Returning result sets from stored procedures
- Secure connections with SSL, SSH, and HTTP tunneling
- Notifications support with the TUniAlerter component
- Support for PostgreSQL Asynchronous Notification with the TUniAlerter component
- Supports the possibility of retrieving last inserted OID value
- Advanced errors support
- Support for the PostgreSQL notices

SQLite

- Support for all commonly used data types
- Support for autoincrement fields
- Possibility to retrieve last auto-incremented value
- SQLite database encryption in Direct mode using different encryption algorithms
- [Data Type Mapping](#)
- Support for automatic database creation on connect
- Support for Shared-Cache mode
- Support for SQLite user-defined functions
- Support for SQLite user-defined collations
- Support for SQLite extensions loading
- Support for SQLite R*Tree module
- Support for SQLite FTS3 and FTS4 extensions
- Support for multi-SQL statements executing

MongoDB

- Support for all commonly used data types
- Support for native MongoDB query and update commands syntax
- Support for displaying/modifying documents using regular data-aware controls like TDBGrid

- Support for simply modifying documents in code using "fluent" interface
- Support for reading/writing documents in the Extended JSON format
- Support for working with collections via regular SQL using VirtualDAC

DB2

- Advanced sequences support
- Schema and function path support

DBF

- Support for variety of database formats: dBaseIII-dBase10, dBase for Windows, HiPer-Six, FoxPro 2, Visual FoxPro
- Support for all native data types
- Support for native dBase functions
- Support for autoincrement fields
- Support for .dbt (dBase), .fpt (FoxPro) and .smt (HiPer-Six) MEMOs
- Support for .mdx (dBaseIV+) and .cdx (Visual FoxPro) indexes
- Support for table management commands: CREATE/DROP/PACK/ZAP/REINDEX TABLE, ALTER TABLE ADD/DROP/ALTER COLUMN
- Support for index management commands: CREATE/DROP INDEX

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

- High performance batch processing
- Intelligent fetch block size control
- Advanced connection pooling
- SmartFetch Mode enabling fast bi-directional navigation through large datasets

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 TUniQuery, TUniTable, and TUniStoredProc components
- Automatic record refreshing and locking
- Automatic query preparing
- Support for ftWideMemo field type in Delphi 2006 and higher
- Data Type Mapping
- Support for Data Encryption in a client application

Extended data access functionality

- Separate component for executing SQL statements
- Simplified access to table data with TUniTable component
- Ability to retrieve metadata information with TUniMetaData component
- BLOB compression support
- Support for using macros in SQL
- FmtBCD fields support
- Ability to customize update commands by attaching external components to TUniUpdateSQL objects

- Deferred detail DataSet refresh in master/detail relationships
- MIDAS technology support
- UniDataAdapter component for WinForms and ASP.NET applications
- Distributed transactions support with the TUniTransaction component [Oracle, SQL Server]
- Default value support for stored procedures
- RefreshQuick method [SQL Server, MySQL]
- Fast record insertion with TUniLoader component
- NonBlocking mode allows background executing and fetching data in separate threads
- LargeInt fields support
- Object-oriented building of SELECT statements

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 TUniDump component

Script execution

- Advanced script execution features with the TUniScript component
- Support for executing individual statements in scripts
- Support for executing huge scripts stored in files with dynamic loading
- Ability to use standard clients tool syntax in scripts

SQL execution monitoring

- Extended SQL tracing capabilities provided by the TUniSQLMonitor component and dbMonitor
- Borland SQL Monitor support

- Ability to send messages to dbMonitor from any point in your program

Visual extensions

- Includes the source code of enhanced TCRDBGrid data-aware grid control
- Customizable connection dialog

Design-time enhancements

- DataSet Manager tool to control DataSet instances in the project
- Advanced design-time component and property editors
- Automatic design-time component linking
- Easy migration from BDE and ADO with Migration Wizard
- More convenient data source setup with the TUniDataSource component
- Syntax highlighting in the design-time editors

Resources:

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

Licensing and support

- Included annual UniDAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.3 Requirements

The UniDAC's core itself has no specific system requirements.

To make an application with UniDAC Express Edition you need at least one of Data Access Components to be installed ([ODAC](#), [SDAC](#), [MyDAC](#), [IBDAC](#), [PgDAC](#), or [LiteDAC](#)).

Provider-specific requirements can be found in the corresponding article of the Provider-specific Notes section.

© 1997-2025
 Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.4 Compatibility

Database Server Compatibility

Database	Windows	macOS	Linux	iOS	iOS Simulator ARM 64-bit	Android
Oracle Servers: 23c, 21c, 19c, 18c, 12c, 11g, 10g, 9i, 8i, 8.0, including Oracle Express Edition 11g and 10g Clients: 23c, 21c, 19c, 18c, 12c, 11g, 10g, 9i, 8i, 8.0	✓	✓	✓	✓	✓	✓
Microsoft SQL Server Servers:	✓	✓	✓	✓	✓	✓

SQL Server 2022 (including Express edition) SQL Server 2019 (including Express edition) SQL Server 2017 (including Express edition) SQL Server 2016 (including Express edition) SQL Server 2014 (including Express edition) SQL						
---	--	--	--	--	--	--

Server 2012 (including Express edition) SQL Server 2008 R2 (including Express edition) SQL Server 2008 (including Express edition) SQL Server 2005 (including Express edition) SQL Server 2000 (including MSDE) SQL Server 7 SQL						
--	--	--	--	--	--	--

Server Compact 4.0, 3.5, 3.1 SQL Azure Clients: SQL OLE DB and SQL Native Client Microsoft SQL Azure Microsoft SQL Azure						
MySQL Servers: 9.0, 8.0, 6.0, 5.6, 5.5, 5.1, 5.0, 4.1, 4.0, and 3.23 Embedd ed servers: 9.0, 8.0,	✓	✓	✓	✓	✓	✓

<p>6.0, 5.6, 5.5, 5.1, 4.1, and 4.0</p> <p>MariaDB Versions since 5.x up to 12.x</p> <p>Microsoft Azure Database for MySQL Microsoft Azure Database for MySQL</p> <p>Amazon RDS for MySQL Amazon RDS for MySQL and Amazon Aurora</p> <p>Google Cloud for MySQL</p>						
--	--	--	--	--	--	--

Google Cloud for MySQL						
InterBase Versions from XE3 up to 2020	✓	✓	✓	✓	✓	✓
Versions since XE	✓	✓	✓			
Versions since 4.2	✓		✓			
Firebird versions 1.x, 2.x, 3.x, 4.x, and 5.x	✓	✓	✓		✓	
PostgreSQL versions from 8.0 up to 17						
Microsoft Azure Database for PostgreSQL Microsoft Azure Database	✓	✓	✓	✓	✓	✓

e for PostgreS QL Amazon RDS for Postgre SQL Amazon RDS for PostgreS QL and Amazon Aurora Google Cloud for Postgre SQL Google Cloud for PostgreS QL Heroku Postgre s Heroku Postgres						
SQLite Version 3.x	✓	✓	✓	✓	✓	✓
MongoD B Servers:	✓	✓	✓			

3.2 and higher Clients: 1.3.5 and higher						
Amazon Redshift Amazon Redshift	✓	✓	✓	✓	✓	✓
NexusDB Versions 4.75.xx	✓					
Microsoft Access Versions 95, 97, 2000, 2003, 2007, 2010, 2013, 2016, 2019	✓	✓	✓	✓	✓	✓
Microsoft Excel Versions from 97 up to 2021	✓	✓	✓	✓	✓	✓
Sybase Adaptive	✓	✓	✓	✓	✓	✓

Server Enterprise versions: 12.5.4 and higher						
Sybase Advantage Database Server versions: 8.0 and higher	✓					
DB2 versions: 8.0 and higher	✓					
DBF Formats: dBaselll- dBase10, dBase for Windows, HiPer- Six, FoxPro 2, Visual FoxPro	✓	✓	✓	✓	✓	✓
BigCommerce	✓					
Dynamic s CRM	✓					

FreshBooks	✓					
Google BigQuery	✓					
HubSpot	✓					
Magento	✓					
Mailchimp	✓					
NetSuite	✓					
QuickBooks	✓					
Salesforce	✓					
SalesforceMC	✓					
SugarCRM	✓					
Zoho CRM	✓					
Any database using ODBC provider	✓	✓ ¹	✓ ¹			

¹If ODBC driver is available for this platform.

IDE Compatibility

UniDAC is compatible with the following IDEs:

Embarcadero RAD Studio 13 Florence

- Embarcadero Delphi 13 Florence for Windows
- Embarcadero Delphi 13 Florence for macOS
- Embarcadero Delphi 13 Florence for Linux
- Embarcadero Delphi 13 Florence for iOS

- Embarcadero Delphi 13 Florence for Android
- Embarcadero C++Builder 13 Florence for Windows
- Embarcadero C++Builder 13 Florence for iOS
- Embarcadero C++Builder 13 Florence for Android

Embarcadero RAD Studio 12 Athens (Requires Release 1, Release 2, or Release 3)

- 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](#) 4.2 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.

Direct mode for Oracle, SQL Server and SAP Sybase ASE is available for all platforms and IDEs, and is distributed as obfuscated source code. SQLite Direct Mode is distributed as pre-compiled packages and available only in Delphi and C++Builder for all target platforms.

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.

Devart. All Rights Reserved.

2.5 Using Several DAC Products in One IDE

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

Packages:

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

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

© 1997-2025

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 Universal Data Access Components library. Click on the name of each component for more information. These components are added to the UniDAC page of the Component palette except for [TCRBatchMove](#) and [TVirtualTable](#) components. They are added to the Data Access page of the Component palette.

UniDAC component list

	TUniConnection	Lets you set up and control connections to different servers.
	TUniEncryptor	Represents data encryption and decryption in client application.
	TUniTransaction	Provides discrete transaction control over sessions. Can be used to manipulate both simple and distributed transactions for certain providers.

	TUniQuery	Uses SQL statements to retrieve data from tables and pass it to one or more data-aware components through a TDataSource object. This component provides a mechanism for updating data.
	TUniTable	Lets you retrieve and update data in a single table without writing SQL statements.
	TUniStoredProc	Executes stored procedures and functions. Lets you edit cursor data returned as parameter.
	TUniSQL	Executes SQL statements, and stored procedures, which do not return datasets.
	TUniScript	Executes sequences of SQL statements, and provides control over the execution process.
	TUniMetaData	Allows to retrieve embracing metadata on specified SQL object
	TUniUpdateSQL	Lets you tune update operations for a DataSet component.
	TUniDataSource	Provides an interface for connecting data-aware controls on a form and UniDAC dataset components.
	TUniLoader	Provides quick loading data to a database.
	TUniDump	Serves to store a database or its parts as a script and also to restore database from received script.
	TUniSQLMonitor	Interface for monitoring dynamic SQL execution.
	TUniConnectDialog	Allows you to build custom prompts for provider name, server name, port number, database, user name, and password.
	TUniAlerter	Used to send and receive database events.
	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.
	TVirtualQuery	Dataset that allows to use SQL statements to retrieve data from in-memory datasets or simultaneously from several different RDBMS'es.
	TCRBatchMove	Transfers data between all types of TDataSet descendants. This component is placed on the Data Access page of the Component palette.

UniDAC Database providers

	TAccessUniProvider	Links the Access provider to an application.
	TAdvantageUniProvider	Links the Advantage provider to an application.
	TASEUniProvider	Links the ASE provider to an application.
	TDB2UniProvider	Links the DB2 provider to an application.
	TDBFUniProvider	Links the DBF provider to an application.
	TExcelUniProvider	Links the Excel provider to an application.
	TInterBaseUniProvider	Links the InterBase provider to an application.
	TMongoDBUniProvider	Links the MongoDB provider to an application.
	TMySQLUniProvider	Links the MySQL provider to an application.
	TNexusDBUniProvider	Links the NexusDB provider to an application.
	TODBCUniProvider	Links the ODBC provider to an application.
	TOracleUniProvider	Links the Oracle provider to an application.
	TPostgreSQLUniProvider	Links the PostgreSQL provider to an application.
	TRedshiftUniProvider	Links the Amazon Redshift provider to an application.
	TSQLServerUniProvider	Links the SQL Server provider to an application.
	TSQLiteUniProvider	Links the SQLite provider to an application.

UniDAC Cloud providers

	TBigCommerceUniProvider	Links the BigCommerce provider to an application.
	TDynamicsCRMUniProvider	Links the Dynamics CRM provider to an application.
	TFreshBooksUniProvider	Links the FreshBooks provider to an application.
	TBigQueryUniProvider	Links the Google BigQuery provider to an application.
	THubSpotUniProvider	Links the HubSpot provider to an application.
	TMagentoUniProvider	Links the Magento provider to an application.
	TMailChimpUniProvider	Links the MailChimp provider to an application.
	TNetSuiteUniProvider	Links the NetSuite provider to an application.
	TQuickBooksUniProvider	Links the QuickBooks provider to an application.
	TSalesforceUniProvider	Links the Salesforce provider to an application.
	TSalesforceMCUniProvider	Links the Salesforce MC provider to an application.
	TSugarCRMUniProvider	Links the SugarCRM provider to an application.
	TZohoCRMUniProvider	Links Zoho CRM provider to an application.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.7 Hierarchy Chart

Many UniDAC classes are inherited from standard VCL/LCL classes. The inheritance hierarchy chart for UniDAC is shown below. The UniDAC classes are represented by

hyperlinks that point to their description in this documentation. A description of the standard classes can be found in the documentation of your IDE.

TObject

- | -TPersistent
 - | -TComponent
 - | | -TCustomConnection
 - | | | [-TCustomDAConnection](#)
 - | | | | [-TUniConnection](#)
 - | | -TDataSet
 - | | | [-TMemDataSet](#)
 - | | | | [-TCustomDADataSet](#)
 - | | | | | [-TCustomUniDataSet](#)
 - | | | | | | [-TUniQuery](#)
 - | | | | | | [-TUniStoredProc](#)
 - | | | | | | [-TUniTable](#)
 - | | | | | | [-TDAMetaData](#)
 - | | | | | | [-TUniMetaData](#)
 - | | | | | [-TVirtualTable](#)
 - | | -TDataSource
 - | | | [-TCRDataSource](#)
 - | | | | [-TUniDataSource](#)
 - | | -T:Devart.Dac.DADataAdapter
 - | | | -T:Devart.UniDac.UniDataAdapter
 - | | [-TCRBatchMove](#)
 - | | [-TCustomConnectDialog](#)
 - | | | [-TUniConnectDialog](#)
 - | | [-TCustomDASQL](#)
 - | | | [-TUniSQL](#)
 - | | [-TCustomDASQLMonitor](#)
 - | | | [-TUniSQLMonitor](#)
 - | | [-TDADump](#)
 - | | | [-TUniDump](#)

- | | [-TDALoader](#)
- | | | [-TUniLoader](#)
- | | [-TDAScript](#)
- | | | [-TUniScript](#)
- | | [-TDATransaction](#)
- | | | [-TUniTransaction](#)
- | | [-TDAAlerter](#)
- | | | [-TUniAlerter](#)
- | | [-TCREncryptor](#)
- | | | [-TUniEncryptor](#)
- | | [-TCustomDAUpdateSQL](#)
- | | | [-TUniUpdateSQL](#)
- | | [-TUniProvider](#)
- | [-TSharedObject](#)
- | | [-TBlob](#)
- | | | [-TCompressedBlob](#)
- | | | | [-TUniBlob](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

2.8 Editions

Universal Data Access Components comes in three editions: Express, Standard, and Professional.

The **Express** edition is free. It includes the UniDAC common engine, but does not include any data providers and additional components. UniDAC Express Edition supports only the following data providers: Oracle, SQL Server, MySQL, InterBase (Firebird), PostgreSQL, and SQLite, which are installed with ODAC, SDAC, MyDAC, IBDAC, PgDAC, and LiteDAC, respectively.

The **Standard** edition includes the UniDAC common engine and data providers.

The **Professional** edition shows off the full power of UniDAC, including mobile application development, data encryption, and additional components for working with databases, such

as TUniAlerter, TUniDump, TUniMetaData, TCRBatchMove, etc. Professional Edition also includes the Direct mode for Oracle, SQL Server, MySQL, and PostgreSQL, and static library linking – for SQLite. In addition, UniDAC Professional Edition includes the DataSet Manager tool, which is intended to organize datasets in your application.

You can get **Source Access** to UniDAC Professional Edition by purchasing a special UniDAC 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 source code of the Direct mode for Oracle and SQL Server is distributed obfuscated, and for SQLite – as precompiled packages.

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

UniDAC Edition Matrix

Features	Express	Standard	Professional
Direct Connectivity			
Oracle	OPT ¹	×	✓
SQL Sever	OPT ¹	×	✓
MySQL	OPT ¹	✓	✓
PostgreSQL	OPT ¹	✓	✓
SQLite	OPT ¹	×	✓
ASE (SAP Sybase Adaptive Server Enterprise)	×	×	✓
DBF	×	×	✓
Desktop Application Development			
Windows	OPT ¹	✓	✓
macOS	OPT ¹	×	✓

Linux	OPT ¹	×	✓
Mobile Application Development			
iOS	OPT ¹	×	✓
Android	OPT ¹	×	✓
Data Access Components			
Base Components:			
TUniConnection TUniQuery TUniSQL TUniTable TUniStoredProc TUniUpdateSQL TUniDataSource	✓	✓	✓
Script Executing TUniScript	✓	✓	✓
Transactions managing TUniTransaction	✓	✓	✓
Fast data loading into the server TUniLoader	OPT ¹	×	✓
Database Specific Components			
Messaging between sessions and applications TUniAlerter	OPT ¹	×	✓
Obtaining metadata about database objects TUniMetaData	OPT ¹	×	✓
Storing a database as a script TUniDump	✓	×	✓
Database Activity Monitoring			
Monitoring of per-component SQL execution TUniSQLMonitor	✓	✓	✓
Additional Components			
Advanced connection dialog TUniConnectDialog	✓	✓	✓
Data encryption and decryption TUniEncryptor	OPT ¹	×	✓

Data storing in memory table TVirtualTable	✓	✓	✓
Dataset that wraps arbitrary non-tabular data TVirtualDataSet	✓	✓	✓
SQL queries against TDataSet descendants TVirtualQuery	✗	✗	✓
Advanced DBGrid with extended functionality TCRDBGrid	✓	✓	✓
Records transferring between datasets TCRBatchMove	OPT ¹	✗	✓
Database Providers			
Access	✗	✓	✓
Advantage	✗	✓	✓
Amazon Redshift	✗	✓	✓
ASE	✗	✓	✓
DB2	✗	✓	✓
DBF	✗	✓	✓
Excel	✗	✓	✓
InterBase/Firebird	✗	✓	✓
MongoDB	✗	✓	✓
MySQL	✗	✓	✓
NexusDB	✗	✓	✓
ODBC	✗	✓	✓
Oracle	✗	✓	✓
PostgreSQL	✗	✓	✓
SQLite	✗	✓	✓
SQL Server	✗	✓	✓

Cloud Providers			
BigCommerce	×	✓ ³	✓ ³
Dynamics 365	×	✓ ³	✓ ³
FreshBooks	×	✓ ³	✓ ³
Google BigQuery	×	✓ ³	✓ ³
HubSpot	×	✓ ³	✓ ³
Magento	×	✓ ³	✓ ³
Mailchimp	×	✓ ³	✓ ³
NetSuite	×	✓ ³	✓ ³
QuickBooks	×	✓ ³	✓ ³
Salesforce	×	✓ ³	✓ ³
Salesforce MC	×	✓ ³	✓ ³
Sugar CRM	×	✓ ³	✓ ³
Zoho CRM	×	✓ ³	✓ ³
Design-Time Features			
Enhanced component and property editors	✓	✓	✓
Migration Wizard	✓	✓	✓
DataSet Manager	×	×	✓
Cross IDE Support			
Lazarus and Free Pascal Support	×	×	SRC ²

¹ Available only if included in the data provider edition.

² Available only in Professional Edition with Source Code.

³ The required ODBC driver is sold and distributed separately from UniDAC.

© 1997-2025

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 UniDAC 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 UniDAC 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 UniDAC. You can distribute UniDAC 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.

© 1997-2025

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 UniDAC and describes the UniDAC Priority Support program.

Support Options

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

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

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

UniDAC Priority Support

UniDAC Priority Support is an advanced product support service for getting expedited individual assistance with UniDAC-related questions from the UniDAC 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 [UniDAC Subscription](#).

To get help through the UniDAC 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 UniDAC Registration number.
- Full UniDAC edition name and version number. You can find both of these from the UniDAC | UniDAC About menu in the IDE.
- Versions of the 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.

© 1997-2025

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 Universal Data Access Components library. It gives a walkthrough for each part of the UniDAC usage process and points out the most relevant related topics in the documentation.

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

What is UniDAC?

Universal Data Access Components (UniDAC) is a component library that provides connectivity to Oracle, SQL Server, MySQL, InterBase, Firebird, PostgreSQL, SQLite, DB2, Microsoft Access, Advantage Database Server, Adaptive Server Enterprise, DBF, NexusDB, and other databases (using ODBC provider) for Delphi, C++Builder and Lazarus (FPC), and helps you develop fast cross-database applications with these environments.

Many UniDAC classes are based on VCL, LCL and FMX classes and interfaces. UniDAC is a complete replacement for [Borland Database Engine](#), provides native database connectivity, and is specifically designed as a universal interface to access different kinds of databases.

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

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

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

Installing UniDAC

To install UniDAC, complete the following steps.

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

For BDS 2006 and Turbo - **unidac100d10*.exe**

For Delphi 7 - **unid100d7*.exe**

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

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

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

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

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

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

Working with the UniDAC demo projects

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

To quickly get started working with UniDAC, launch and explore the introductory UniDAC demo project, *UniDACDemo*, from your IDE. This demo project is a collection of demos that show how UniDAC 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.

UniDACDemo Walkthrough

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

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

```
\Program Files\Devart\UniDAC for Delphi 2006\Demos\Win32\UniDACDemo  
\UniDACDemo.bdsproj
```

4. Select Run | Run or press F9 to compile and launch the demo project. *UniDACDemo* should start, and a full-screen UniDAC Demo window with a toolbar, an explorer panel, and a view panel will open. The explorer panel will contain a list of all demo sub-projects included in *UniDACDemo*, 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 a server. However, you will not be able to actually retrieve data from a server or execute commands until you connect to the database.

5. Click on the "Connect" button in the *UniDACDemo* toolbar. A Connect dialog box will open. Select the required provider from the list, and enter the connection parameters to connect to your server, and click "Connect" in the dialog box. Set of connection parameters depends on the selected provider.

Now you have a fully functional interface to your server. You will be able to go through the different demos, to browse tables, create and drop objects, and execute 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 *UniDACDemo*. If some of these objects already exist in the database you have connected to, an error with the error

message like the following will appear.

*"An error has occurred: ORA00955: name is already being used by an existing object. ...
Ignore this exception?"*

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

7. Choose a demo that demonstrates an aspect of working with UniDAC 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 TUniTable component, select the Table demo from the "Working with Components" folder. A simple 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 *UniDACDemo* toolbar to find out how the demo you have selected was implemented. The source code behind the demo project will appear in the view panel. Try to find the places where UniDAC components are used to connect to the database.
9. Click on the "Form as text" button in the *UniDACDemo* toolbar to view the code behind the interface to the demo. Try to find the places where UniDAC 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 the basic UniDAC components.

General demos

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

Server-specific demos

A collection of projects that demonstrate how to incorporate features of specific database servers.

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

Other UniDAC demo projects

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

Compiling and deploying your UniDAC project

Compiling UniDAC-based projects

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

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

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

Deploying UniDAC-based projects

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

- The Client software, if connecting not in the Direct mode.
- The UniDAC bpl files, if compiling with runtime packages.
- The UniDAC assembly files, if using VCL for .NET components.

If you are evaluating deploying projects with UniDAC 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 UniDAC Trial

Edition for C++Builder will only work if the C++Builder IDE is launched. More information about UniDAC Trial Edition limitations is provided [here](#).

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

Using the UniDAC documentation

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

The UniDAC documentation includes a detailed reference of all UniDAC components and classes. Many of the UniDAC components and classes inherit or implement members from other VCL and LCL 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 or LCL class a UniDAC component is inherited from, see the corresponding topic in your IDE documentation.

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

How to get help with UniDAC

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

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

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

Reserved.

3.1 Installation

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

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

Before installing UniDAC ...

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

So before installing a new version of UniDAC, uninstall any previous version of UniDAC 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 support@devart.com

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

Installed packages

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

Delphi/C++Builder Win32 project packages

Name	Description	Location
------	-------------	----------

dacXX.bpl	DAC run-time package	Delphi\Bin; Windows\System32
dacvclXX.bpl*	DAC VCL support package	Delphi\Bin
dcldacXX.bpl	DAC design-time package	Delphi\Bin
unidacXX.bpl	UniDAC run-time package	Delphi\Bin; Windows\System32
unidacvclXX.bpl*	VCL support package	Delphi\Bin
dclunidacXX.bpl	UniDAC design-time package	Delphi\Bin
XXproviderXX.bpl	UniDAC providers packages	Delphi\Bin; Windows\System32
crcontrolsXX.bpl	TCRDBGrid component	Delphi\Bin

Additional packages for using UniDAC managers and wizards

Name	Description	Location
datasetmanagerXX.bpl	DataSet Manager package	Delphi\Bin

Environment Changes

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

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

Delphi

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

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

C++Builder

C++Builder 6:

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

Options accessible from Project | Options | Directories/Conditionals.

C++Builder 2006, 2007:

- \$(BCB)\UniDAC\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)\UniDAC\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 UniDAC Installer performs C++Builder environment changes automatically for compiled versions of UniDAC.

Lazarus

The UniDAC installation program only copies UniDAC files. You need to install UniDAC packages to Lazarus IDE manually. Open %UniDAC%\Source\Lazarus1\dclunidac10.lpk (for Trial version %UniDAC%\Packages\dclunidac10.lpk) file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilt with UniDAC packages.

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

Installation of Additional Components and Add-ins

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.2 Migration Wizard

Note: Migration Wizard is only available for Delphi.

UniDAC Migration Wizard allows you to convert your BDE, IBX, ADO, dbGo, ODAC, SDAC,

MyDAC, IBDAC, PgDAC, LiteDAC, AnyDAC, FireDAC, and FIBPlus projects to UniDAC.

This wizard replaces the database components in a specified project (.dfm and .pas-files) with UniDAC components.

To convert a project, perform the following steps:

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

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

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

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

Your project will be restored to its previous state.

Reserved.

3.3 UniDAC Basics

- [Introduction](#)
- [Connecting to the Database](#)
- [Selecting Data](#)
- [Executing Queries](#)
- [Editing Data](#)
- [Executing Stored Procedures](#)
- [Creating Master/Detail Relations](#)
- [Unified SQL](#)

Introduction

Universal Data Access Components (UniDAC) is a powerful library of nonvisual cross-database data access components for Delphi, C++Builder and Lazarus(Free Pascal). The UniDAC library is designed to help programmers develop faster and cleaner cross-database applications. UniDAC is a complete replacement for standard database connectivity solutions and presents an efficient native alternative to the Borland Database Engine and dbExpress for access to Oracle, SQL Server, MySQL, InterBase, Firebird, SQLite, DB2, Microsoft Access, Advantage Database Server, Adaptive Server Enterprise, DBF, NexusDB, and other databases (using ODBC provider).

UniDAC is based on the well-known Data Access Components from Devart such as ODAC, SDAC, MyDAC, IBDAC, and PgDAC.

This article provides an overview of the concepts and tasks you will apply when you work with UniDAC.

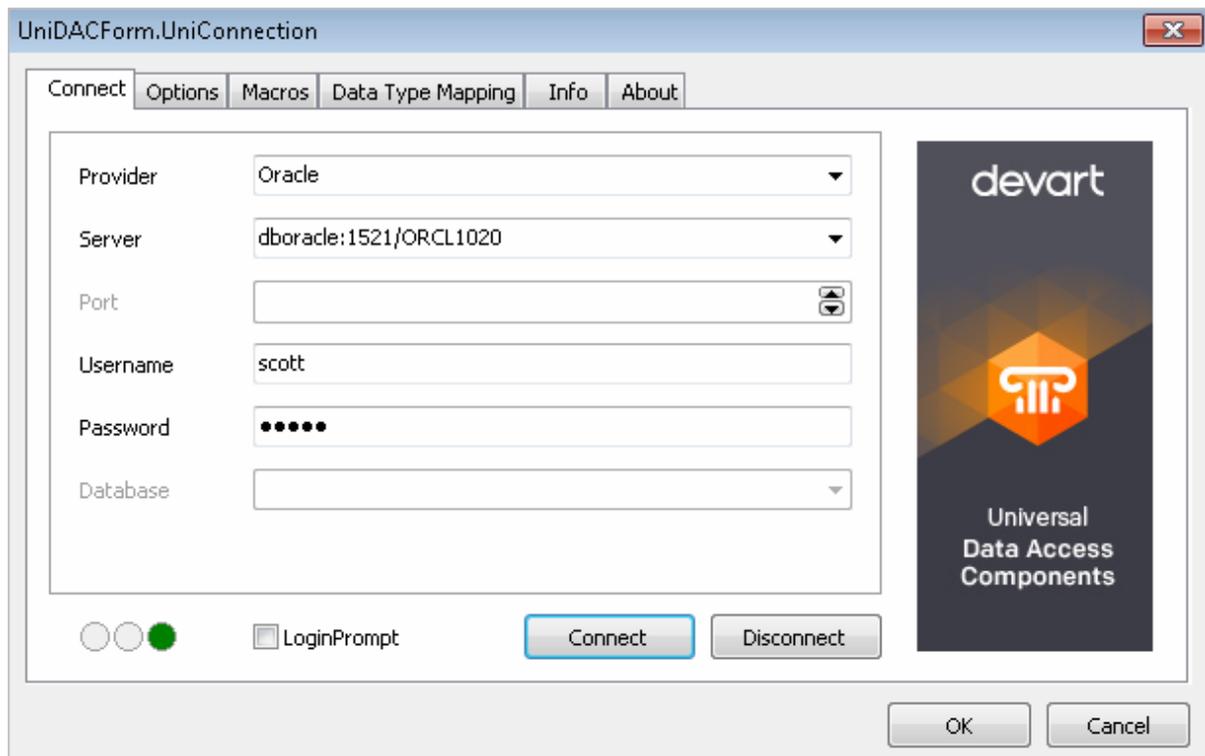


Connecting to the Database

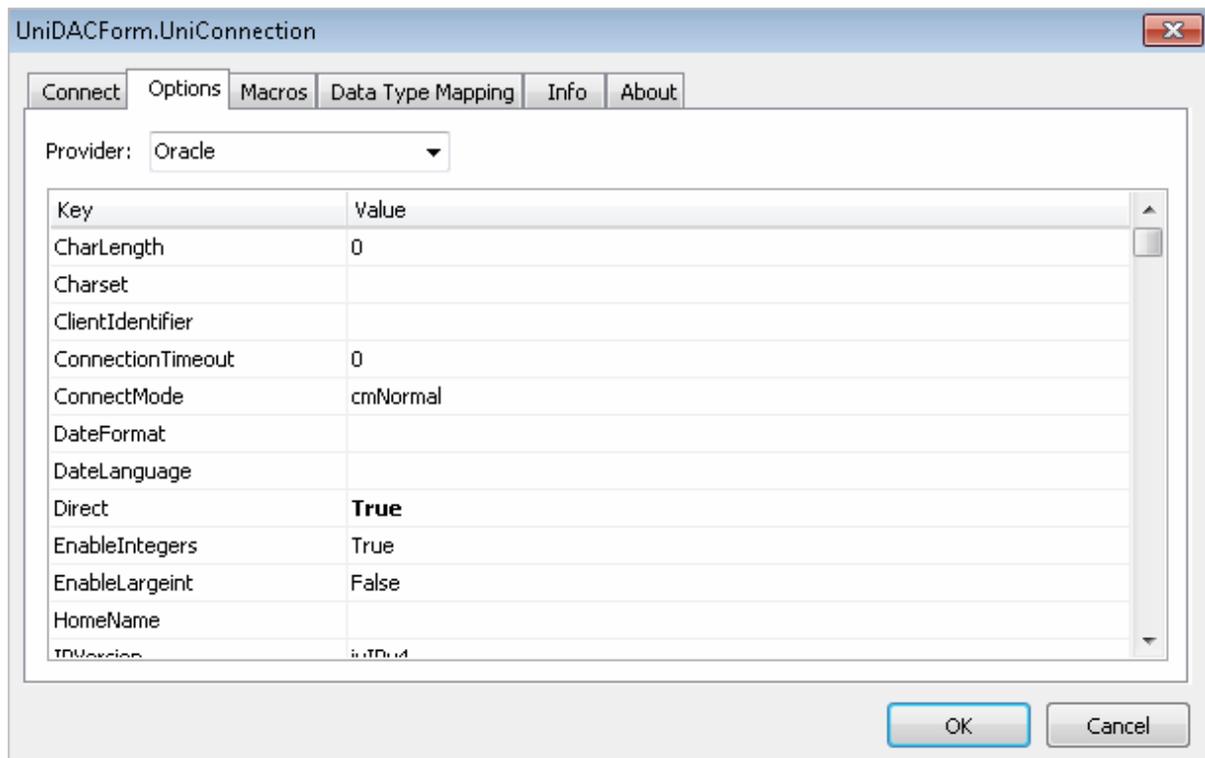
Connecting to the Database in Design-Time

For UniDAC component using you have to do following steps:

- Create an empty application that will be used to work with UniDAC components. Select **File | New | VCL Forms Application** from the Delphi menu.
- Find UniDAC page on the component palette and drop *TUniConnection* component on the form.
- Set the main properties of *TUniConnection* using TUniConnection editor. Double click the *TUniConnection* component on the form to open the editor.
- Select a provider name corresponding to your database from the **Provider** drop-down combobox. For example, select Oracle for connecting to an Oracle database.
- Enter the following connection parameters: user name, password, server, database, and port into the editor. Some of connection parameters are not used, depending on the selected provider. For Oracle you need to enter user name, password, and server, for example. **Server** is a TNS alias name of an Oracle database. You can select value for **Server** from the drop-down list or enter it manually.



- Click the **Connect** button. If the connection is established successfully the editor closes automatically.
- Open the editor again by double-clicking the *TUniConnection* component and select the **Options** page. Here you can enter some options specific to the provider. Schema is a useful option for an Oracle database. We will use objects from the "SCOTT" sample schema in this example. So, enter "SCOTT" as a value for **Schema**.



Connecting to the Database at Run-Time

Set the *TUniConnection* parameters and open it at run-time. The following example shows how to do this:

```
UniConnection1: TUniConnection;  
...  
UniConnection1.ProviderName := 'Oracle';  
UniConnection1.Username := 'scott';  
UniConnection1.Password := 'tiger';  
UniConnection1.Server := 'ORA1020';  
UniConnection1.SpecificOptions.Values['Schema'] := 'SCOTT';  
UniConnection1.Open;
```

Each line in the **SpecificOptions** property has the following format:

<OptionName>=<Value>. You can add options using the *Add* method:

```
UniConnection1.SpecificOptions.Add('Schema=SCOTT');
```

But it is better to use the **Values** property of *TStrings* because this property does not add a

new line if an option with the same name already exists. Instead it replaces the text after '=' with a new value.

To close the connection use the *Close* method:

```
UniConnection1.Close;
```

You should link all the providers that you use in the application. To link a provider, add its unit to the **USES** list. For Oracle add the *OracleUniProvider* to USES:

```
uses ..., OracleUniProvider;
```

The provider unit can be easily added by help of the UniDAC Providers palette page. Select this page, find the **OracleUniProvider** component and drop it on the form. IDE will add the corresponding unit to **USES** automatically if it is not added yet.

Selecting Data

The *TUniQuery* and *TUniTable* components allow you to select data. To do it, drop *TUniQuery* component into the form. For data selecting you have to establish a connection to the database. You need to set the **Connection** property for most components. If there is a *TUniConnection* component into the form, UniDAC automatically sets the **Connection** property to this component.

For the *TUniQuery* you need to set the SQL property. Double click the *TUniQuery* component to open the *TUniQuery* editor. On the first page of the editor you can enter the text for the SQL property.

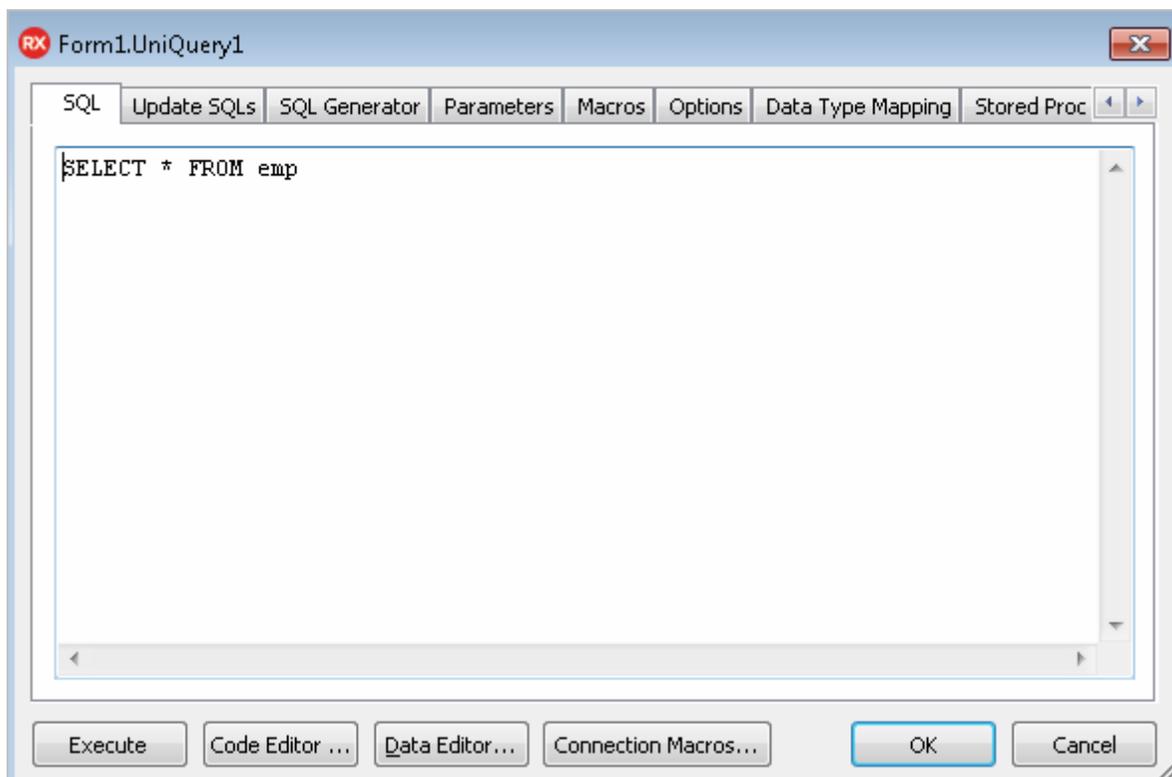
TUniSQL component is used to execute queries without recordset. The *TUniSQL* is not a *TDataSet* descendant like *TUniQuery*. *TUniSQL* is a simple component that provides the best performance.

It is used in the same way as the *TUniQuery*. If you want to define SQL and parameters - use *TUniSQL* editor at design-time. You can define SQL and parameters at run-time too. To execute query you have to assign a value for the SQL property and call the *Execute* method.

If you connect to the SCOTT sample schema, you can enter:

```
SELECT * FROM emp
```

to select data from the EMP table.



Click the **OK** button to save changes and close the editor. To execute the query, you can change the *Active* property to True in Object Inspector, or call the *Open* method in your program:

```
UniQuery1: TUniQuery;  
...  
UniQuery1.Connection := UniConnection1;  
UniQuery1.SQL.Text := 'SELECT * FROM emp';  
UniQuery1.Open;
```

The Displaying Data

Drop *TDataSource* and *TDBGrid* components into the form to see data from *TUniQuery*. You can use standard *TDataSource* from the Data Access palette page or *TUniDataSource* component from the UniDAC page. These components have same functionality but *TUniDataSource* sets the *DataSet* property automatically.

Set the *DataSet* property of *TDataSource* to *UniQuery1* (if it is not set automatically). Then set the *DataSource* property of *TDBGrid* to *DataSource1*. If the **Active** property of *UniQuery* is **True**, *DBGrid* will display data.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	15.02.1981 22:00:00	1677	300	30
7521	WARD	SALESMAN	7698	22.02.1981	1290	500	30
7566	JONES	MANAGER	7839	02.04.1981	2975		20
7654	MARTIN	SALESMAN	7698	28.09.1981	1250	1400	30
7698	BLAKE	MANAGER	7839	01.05.1981	2850		30
7782	CLARK	MANAGER	7839	09.06.1981	2450		10
7788	SCOTT	ANALYST	7566	19.04.1987	3000		20
7839	KING	PRESIDENT		17.11.1981	5000		10
7844	TURNER	SALESMAN	7698	08.09.1981	1500	0	30
7876	ADAMS	CLERK	7788	23.05.1987	1100		20
7900	JAMES	CLERK	7698	03.12.1981	950		30

To close the *TUniQuery* use its *Close* method or set its **Active** property to **False**.

UniQuery with data always has a current record. Current record is changed while you move across the *DBGrid*.

Current record can be changed programmatically by help of the *First*, *Last*, *Next*, *Prior*, *Locate*, and *LocateEx* methods of the *TUniQuery*.

Working with Files

The *TUniQuery* has a *Fields* collection containing one *TField* object for each field in your query. You can get a reference to the *TField* object by field number or by using *FieldByName* method:

```
UniQuery1.Fields[0];  
UniQuery1.FieldByName('EMPNO');
```

TField object can read data from the current record. Use a **Value** property of *TField* or typed properties like **AsInteger**, **AsString**, etc.

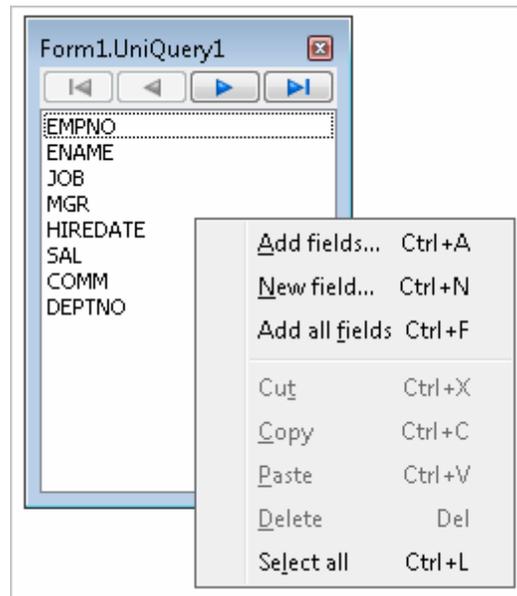
For example, you can copy data from the *TUniQuery* to a *TMemo* component using the following code:

```
var  
  Empno: integer;  
  Ename: string;  
begin  
  Memo1.Lines.Clear;  
  UniQuery1.Open;  
  UniQuery1.First;  
  while not UniQuery1.Eof do begin  
    Empno := UniQuery1.FieldByName('EMPNO').AsInteger;  
    Ename := UniQuery1.FieldByName('ENAME').AsString;  
    Memo1.Lines.Add(IntToStr(Empno) + ' ' + Ename);  
    UniQuery1.Next;  
  end;  
  UniQuery1.Close;  
end;
```

The *Next* method sets the *Eof* property of *TUniQuery* to True if it cannot move to the next record because there are no more records.

The *TUniQuery* creates and destroys fields dynamically when you open and close the query. Sometimes you need to create persistent fields generated with the form. To create persistent fields, right click *TUniQuery* component and select **Fields Editor** from the context menu.

Fields Editor window will be opened. Right click inside the **Fields Editor** window and select **Add all fields** from the menu. Now you will see the list of fields in the window.



Fields are created as the components on the form. IDE adds corresponding variable of form class for each field. You can rewrite the previous code example using the persistent field variables:

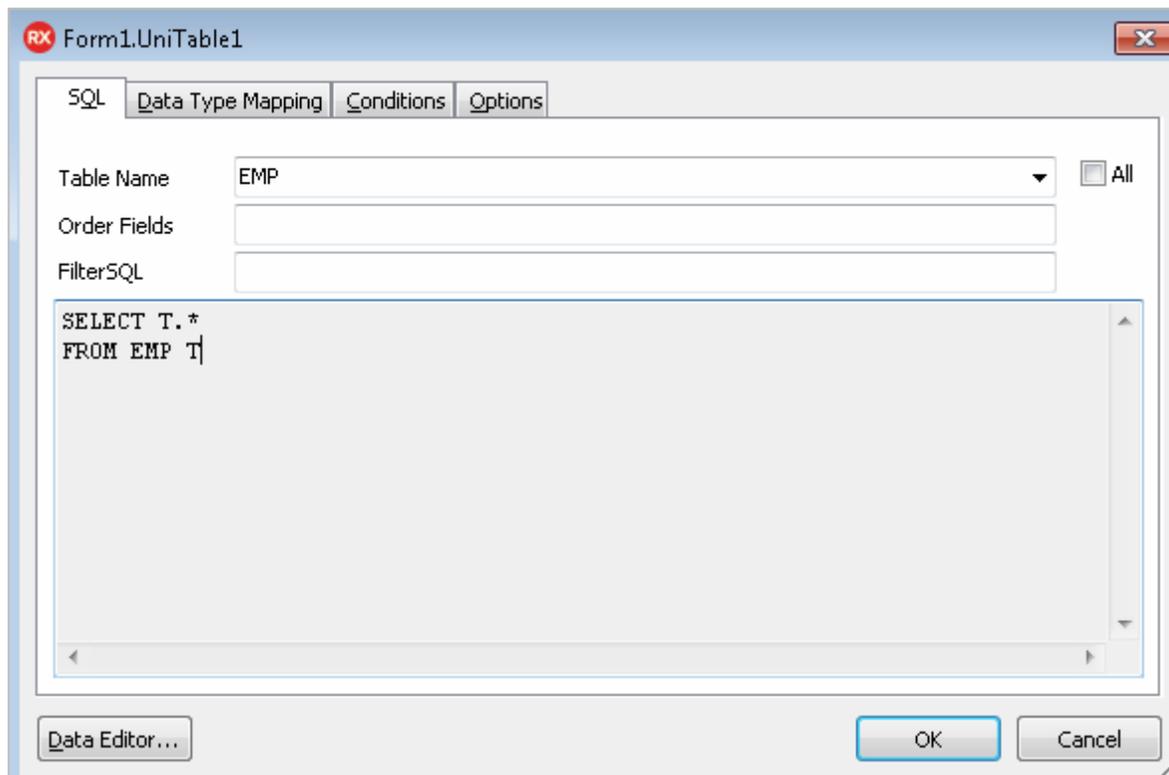
```

...
while not UniQuery1.Eof do begin
    Empno := UniQuery1EMPNO.AsInteger;
    Ename := UniQuery1ENAME.AsString;
    Memo1.Lines.Add(IntToStr(Empno) + ' ' + Ename);
    UniQuery1.Next;
end;
...

```

We recommend use *TUniTable* to select data from one table. You don't need to write SQL statement for *TUniTable*. You set the *TableName* property and *TUniTable* automatically generates SQL statement to get data from this table.

Drop the **TUniTable** into the form and double-click the component to open *TUniTable* editor. You can enter value for the *TableName* property and for *OrderFields* and *FilterSQL* properties in the editor.



When *OrderFields* and *FilterSQL* properties are empty, *TUniTable* generates simple SQL statement like

```
SELECT * FROM emp
```

If you set values for *OrderFields* or *FilterSQL*, corresponding ORDER BY or WHERE clauses will be added to the statement.

Executing Queries

TUniQuery can be used not only for selecting data but for executing any queries supported by database server.

For example, you can change records in the EMP table by using the *TUniQuery* with UPDATE statement. Drop the *TUniQuery* component on the form and double click it to open the editor. Enter the following text for SQL:

```
UPDATE emp SET sal = sal + 1 WHERE empno = 10
```

The query can be executed at design-time from the editor using the **Execute** button. To execute the query at run-time, call the *Execute* method of *TUniQuery*.

```
UniQuery1.Execute;
```

Parameters

Queries don't use fixed values in "SET" or "WHERE" clause in general. For example, your program can get the new values for "SAL" and "EMPNO" fields from the user.

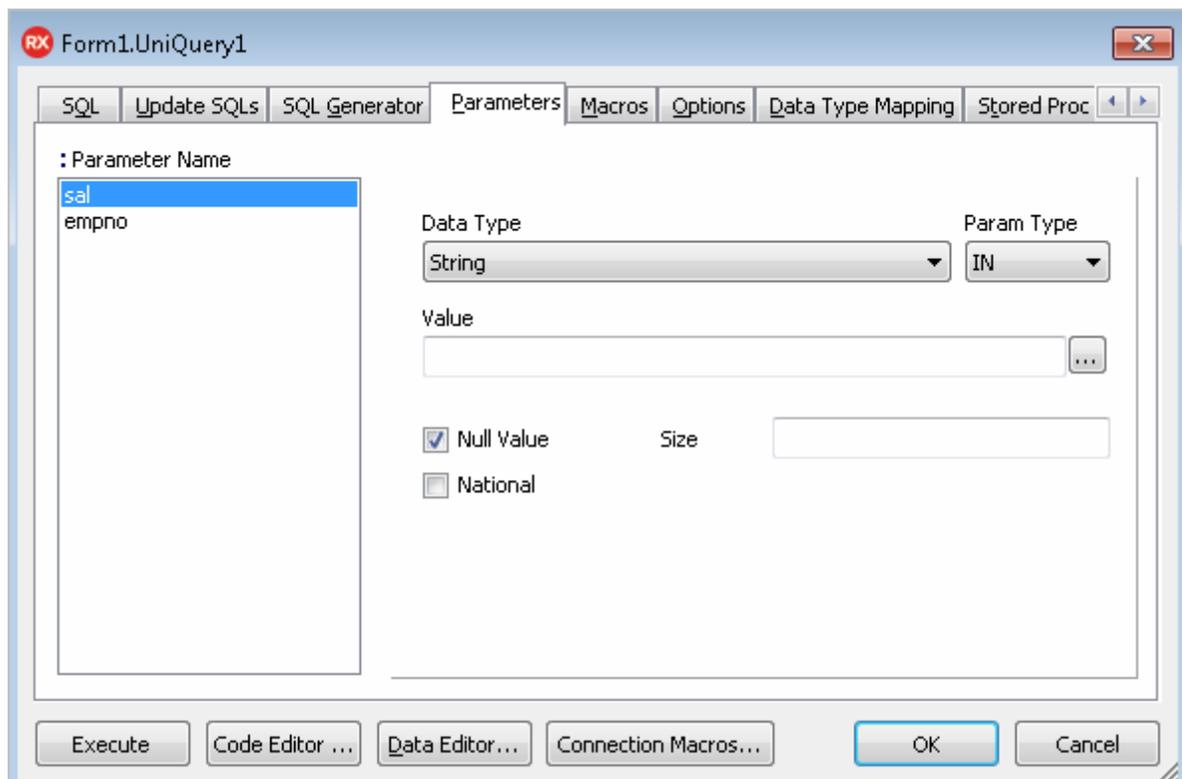
You can use parameters for this purpose:

```
UPDATE emp SET sal = :sal WHERE empno = :empno
```

Parameters are marked using ':' (colon) and parameter name.

Values of the parameters can be set at run-time, and the server replaces parameter names with the values during the query execution.

After the query with parameters was defined into the SQL tab of the *TUniQuery* editor, go to the **Parameters** tab. Here you have to set *DataType* and *ParamType* for each parameter



At run-time you can access the parameters by number or by name using the Params collection of *TUniQuery*.

```
UniQuery2.Params[0];  
UniQuery2.ParamByName('SAL');
```

Use the following code to execute query with parameters:

```
UniQuery2.ParamByName('SAL').AsFloat := 100;  
UniQuery2.ParamByName('EMPNO').AsInteger := 10;  
UniQuery2.Execute;
```

Each parameter is substituted only by single value in the SQL statement.

Macros

Any part of statement (table name, for example) can be changed dynamically with macros. The macros are marked with '&' (ampersand) and macro name:

```
SELECT * FROM &macro1
```

The macros are accessed by number or name from the Macros collection of *TUniQuery* component in your program code.

```
UniQuery3.Macros[0];  
UniQuery3.MacroByName('MACRO1');
```

The value of a macro can be set by the **Value** property of a *TMacro*. For example:

```
UniQuery3.MacroByName('MACRO1').Value := 'emp';
```

or

```
UniQuery3.MacroByName('MACRO1').Value := 'emp ORDER BY ename';
```

Editing Data

All of the datasets components described above are editable. Call the *Edit* method to begin editing. Call the *Post* or *Cancel* method to finish editing. If you call *Post*, the changes are passed to the database server. If you call *Cancel*, changes will be revoked.

```
UniQuery1.Edit;  
UniQuery1.FieldByName('HIREDATE') := Now;  
UniQuery1.FieldByName('SAL') := 1000;  
UniQuery1.Post;
```

Database Controls like TDBGrid or TDBEdit allow user for data editing.

- Run the test application.

- You can edit any cell in DBGrid linked to *TUniQuery*. The *Edit* method called automatically, when editing starts. The *Post* method is called, when another record is selected. To cancel your changes in the current record, press the ESC key.

A new record can be inserted by the *Insert* or *Append* method. The *Append* method adds record to the end of dataset. The *Insert* method inserts record in the current position. After one of these methods is called, you should assign values to the fields and call the *Post* method:

```
UniQuery1.Append;  
UniQuery1.FieldName('EMPNO') := -1;  
UniQuery1.FieldName('ENAME') := 'NEW EMP';  
UniQuery1.FieldName('HIREDATE') := NOW;  
UniQuery1.FieldName('SAL') := 2000;  
UniQuery1.Post;
```

To delete record in the current position, call the *Delete* method.

UniDAC executes "INSERT", "UPDATE", or "DELETE" statement to apply changes to the database.

Debugging

UniDAC can show SQL statements in dialog window before execution. Set the *Debug* property of *TUniQuery* to True to see SQL statements of your query. For profiling in real-time you have to add the **UniDacVcl** unit to the USES list. Then run the application. You see the SELECT statement at startup. Try to edit a record, add a new record, and delete this record. You will see the corresponding update statements in the Debug window.

Updating table property

If more than one table is specified in the query, UniDAC allows you to update data only in one table. Fields from other tables become read-only. For example, change the *SQL* property of *UniQuery1* to the following:

```
SELECT e.*, d.dname  
FROM emp e INNER JOIN dept d ON e.deptno = d.deptno
```

Now you can edit all the fields except the last field DNAME.

UpdatingTable property contains a name of the table that will be updated.

UniDAC uses the first table specified after "SELECT" or the first table pointed after "FROM" as default updating table, depending from the current data provider.

If your query contains several tables, it is recommended to always set the *UpdatingTable* property to the table you want to edit.

General field information

UniDAC requires information about key fields of the updating table to generate "WHERE" clause of "UPDATE" and "DELETE" statements. Some servers like SQL Server return this information when a query is executed. Oracle and other database servers do not return information about key fields, so UniDAC performs the additional query to the database to get key fields. You can set the KeyFields property of **TUniQuery** manually. In this case an additional query is not executed.

Complex queries

If you set a complex query to the *SQL* property, UniDAC may not be able to generate the correct update statements. Or you need custom SQL statements to apply changes to the database (for example, you can apply changes using stored procedures instead of "INSERT", "UPDATE", and "DELETE" statements). You can use the *SQLInsert*, *SQLUpdate*, and *SQLDelete* properties of *TUniQuery* to set custom update statements. If you double-click one of these properties in Object Inspector, the **Update SQLs** page of the *TUniQuery* editor is opened.

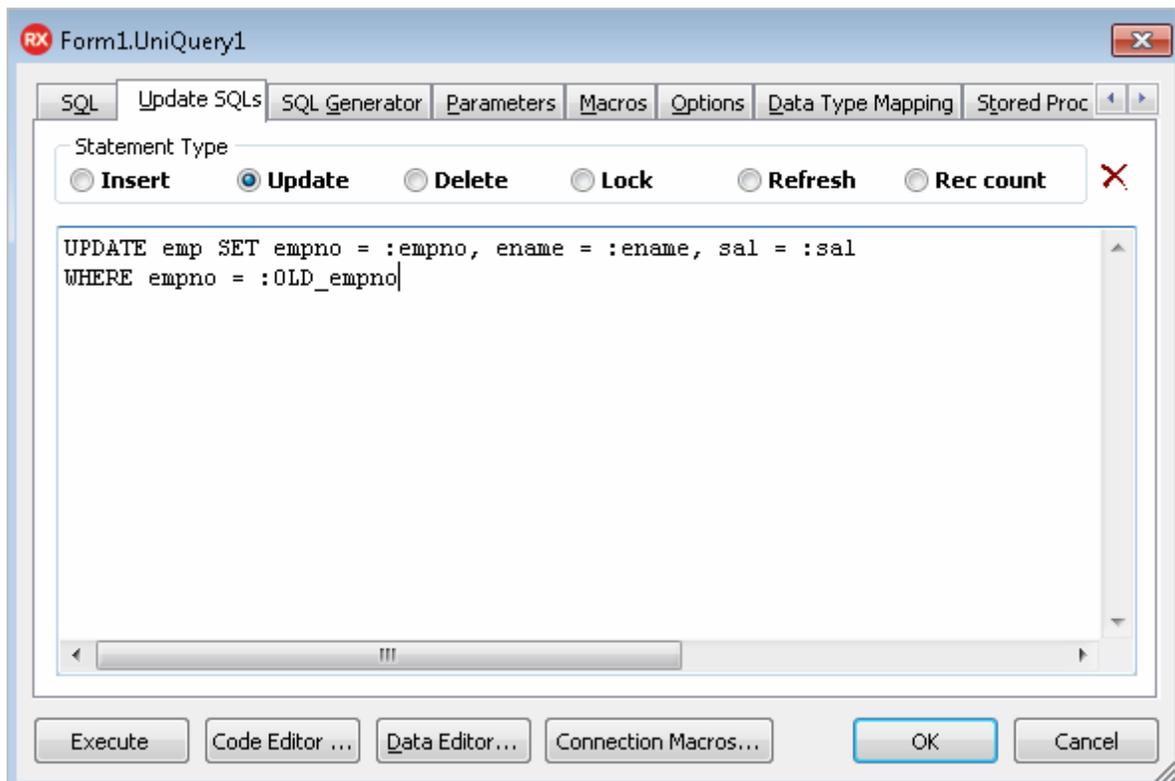
A field value in the update queries can be referenced by the parameter with the same name as field name. For example, use the following statement in the *SQLUpdate* property to save changes to "ENAME" and "SAL" fields.

```
UPDATE emp SET ename = :ename, sal = :sal  
WHERE empno = :empno
```

Old parameters

You can reference to an old value of the field by adding "OLD_" prefix to the parameter name. For example, if user can change value of EMPNO field, you need to use the old value of this field in the "WHERE" condition:

```
UPDATE emp SET empno = :empno, ename = :ename, sal = :sal  
WHERE empno = :OLD_empno
```



SQL generator

For simple SQL-queries *SQL properties* can be updated automatically on the **SQL generator** tab. Go to the **SQL Generator** page of the query editor. If your query has several tables in the "FROM" clause, select table to update in the **Table Name** combobox. You can select statement types to be generated, key fields, and data fields.

Click the **Generate SQL** button. The update statements are generated and the editor changes the current page to **Update SQLs**. Now you can make changes in the generated statements.

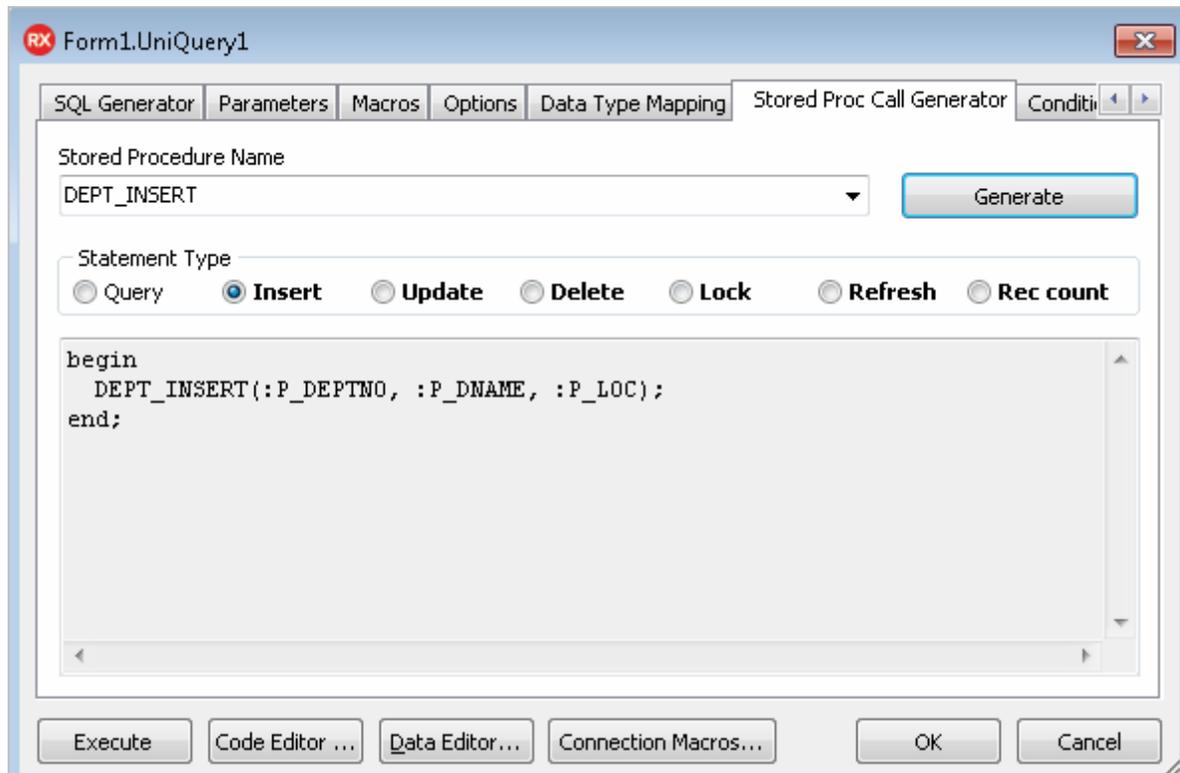
Using stored procedures

Stored procedure can be used in the update statements. The procedure for insert is similar to following (example for Oracle):

```
CREATE OR REPLACE PROCEDURE DEPT_INSERT
(pDNAME VARCHAR, pLOC VARCHAR)
AS
BEGIN
INSERT INTO DEPT (DNAME, LOC) VALUES (pDNAME, pLOC);
```

```
END;
```

An SQL statement for stored procedure call can be written manually or created by generator. Go to the **Stored Proc Call Generator** page, select the stored procedure name, select the statement type and click the **Generate** button.

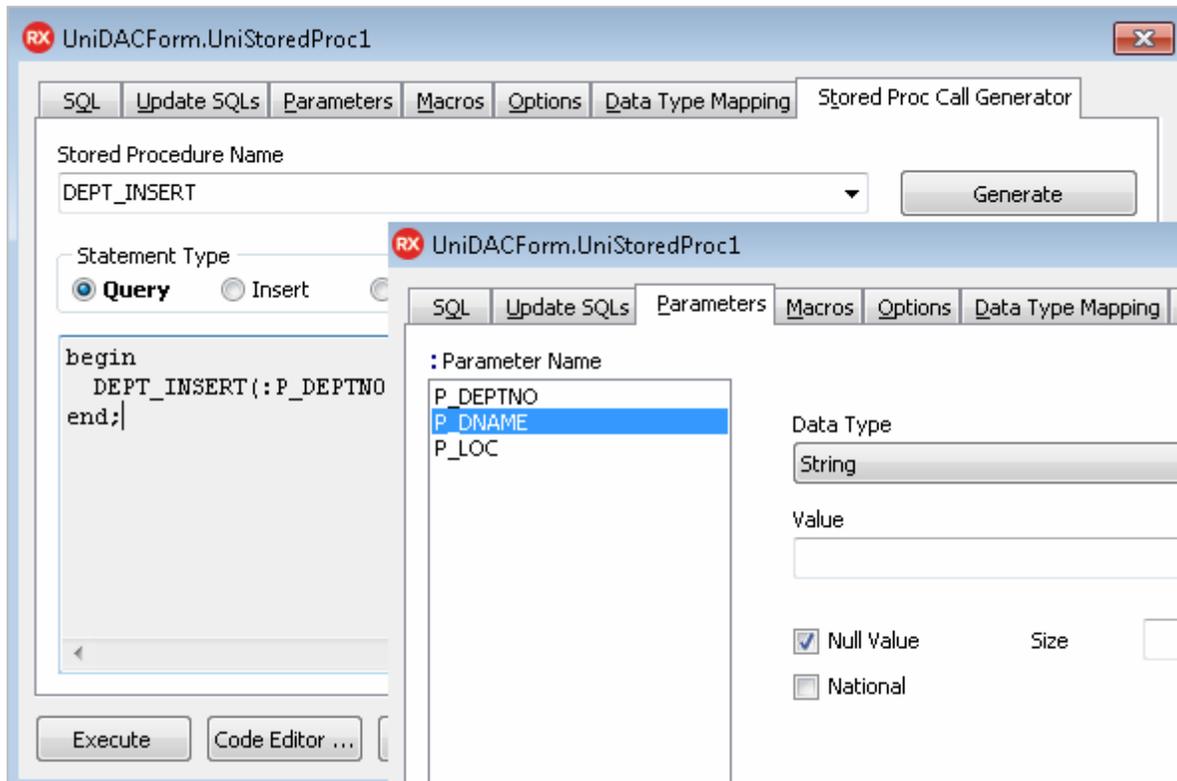


Executing Stored Procedures

TUniStoredProc allows you to execute a stored procedure.

- Drop *TUniStoredProc* on the form and double-click it. *TUniStoredProc* editor will be opened.
- Enter the stored procedure name or select it from the list. For example, you can select "EMP_INS" procedure from the previous topic.
- When you move focus to another control or press the **Create SQL** button (📄), the editor creates SQL statement for calling the procedure. You can see it in the box below the stored procedure name.

- If the procedure has parameters, they will be added to the generated SQL statement and to the Params property.



To call the procedure at run-time use the *Execute* method. You may also set the stored procedure name and generate SQL statement for calling the stored procedure at run-time. Call the *PrepareSQL* method to generate SQL statement for stored procedure. After that Params collection is filled, and you can assign values to the parameters.

```
UniStoredProc1.StoredProcName := 'DEPT_INSERT';
UniStoredProc1.PrepareSQL;
UniStoredProc1.ParamByName('PDNAME').AsString := 'DEPT 1';
UniStoredProc1.ParamByName('PLOC').AsString := 'California';
UniStoredProc1.Execute;
```

Creating Master/Detail Relations

Imagine that you have two tables, and second table has a field (foreign key) that references the primary key of the first table. For example, the "SCOTT" sample schema in the Oracle database has "DEPT" and "EMP" tables. "DEPT" contains the list of departments, and "EMP" contains the list of employees. "DEPT" table has DEPTNO primary key. "EMP" also has the

DEPTNO field. This field references the "DEPT" table and contains a number of the department where an employee works.

If you have two *TUniQuery* or *TUniTable* components, you can link them in a master/detail relation. The detail dataset shows only records corresponding to the current record in the master dataset.

For example, drop two *TUniTable* components on the form. Set the **Name** property of the first table to "DeptTable", and **TableName** property to "Dept". Set the **Name** property of the second table to "EmpTable", and **TableName** property to "Emp". Set the **Active** property of both tables to True.

Drop two **TUniDataSource** components on the form, set their names to "DeptDS" and "EmpDS", and link them to the corresponding tables. Then drop two TDBGrid components and link them to the corresponding data sources.

Set the **MasterSource** property of *EmpTable* to "DeptDS". Double-click the **MasterFields** property of *EmpTable* in Object Inspector. It will open the editor for linking fields between details and master. Select the DEPTNO field in both left and right list and click the **Add** button. Click the **OK** button to close the dialog.

Now *EmpTable* displays only employees from the current department. If you change the current record in *DeptTable*, *EmpTable* is automatically refreshed and displays another employees.

When you set **MasterSource** for *TUniTable* or *TUniQuery*, its **SQL** is automatically modified. Fields that you linked are added to the **WHERE** condition:

```
SELECT * FROM EMP
WHERE DEPTNO = :DEPTNO
```

The parameter value is set from the corresponding field of the master dataset, then the query is executed. When you change the current record in the master, the parameter value in the detail is changed, and the detail query is reexecuted.

Text parameters, corresponding to the master fields, can be added to the SQL text manually. In this case you don't need to set the **MasterFields** property, just set the **MasterSource** property. UniDAC sets values for parameters automatically if the master dataset has fields with the same name.

When the current record in the master is changed, the detail query is reexecuted each time.

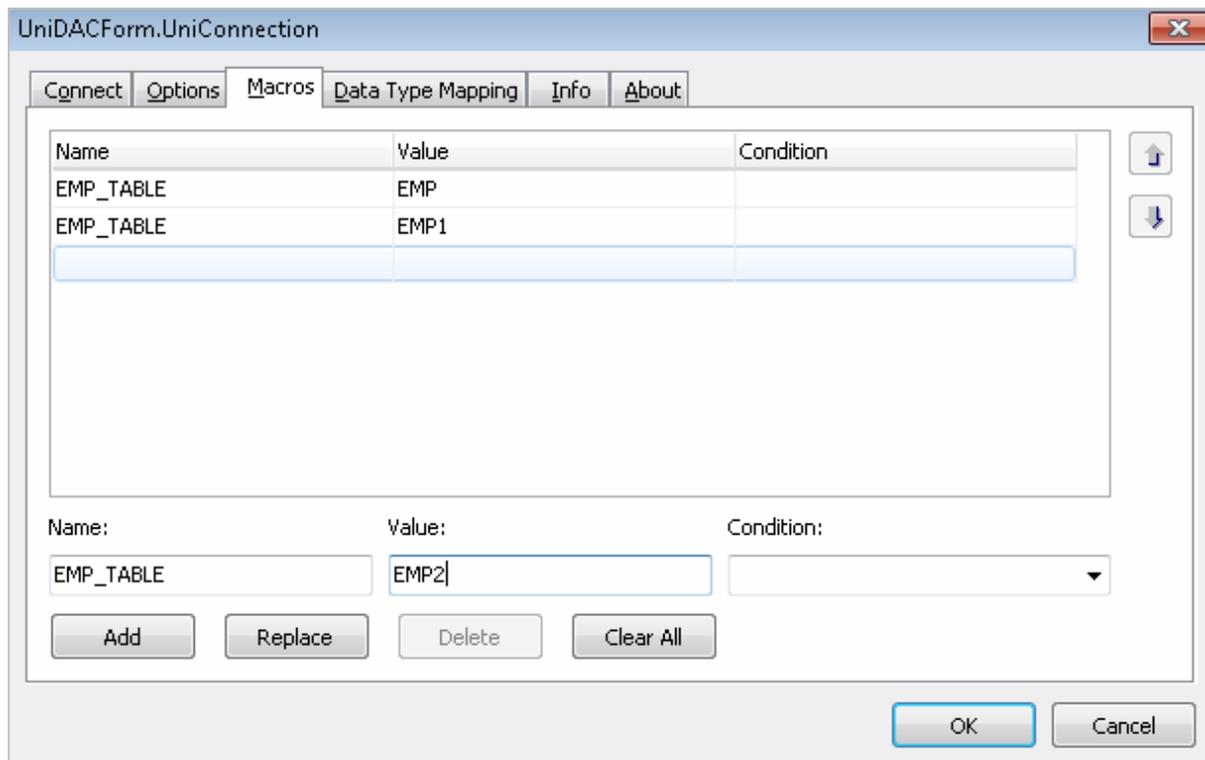
You can avoid this by using local master/detail. Set **Options.LocalMasterDetail** to True for *TUniTable* or *TUniQuery*. In this case parameters are not added to the detail query. This query is executed only one time and returns all records. UniDAC filters these records locally to display only records corresponding to the master record.

Unified SQL

Unified SQL includes special directives, macros, literals, and functions. You can use Unified SQL to write SQL statements that are independent from used provider and database. There are several ways to do it. First way is using connection macros and IF directive. UniDAC automatically defines the macro that corresponds to the selected provider in this way. For example, if you select Oracle provider, **Oracle** macros is defined. If you want to use "EMP1" table for Oracle and "EMP2" table for SQL Server, you can assign the following to the **SQL** property of *TUniQuery*:

```
{if ORACLE}
SELECT * FROM EMP1
{else}
{if SQLSERVER}
SELECT * FROM EMP2
{else}
SELECT * FROM EMP
{endif}
{endif}
```

To define macros at design-time, open the *TUniConnection* editor and select **Macros page**. Fill **Name** and **Value** boxes at the bottom of the page. Then press the **Add** button. You can use the added macro in IF directive or directly in SQL statements.



For example, if you define macro "EMP_TABLE" with value "EMP", you can write the following SQL statement:

```
SELECT * FROM {EMP_TABLE}
```

The several macros with the same name but different value and conditions can be defined. Condition is the name of another macro. If the macro, specified in condition, is enabled, the current macro is also enabled and its value replaces the macro name in SQL statements. If the macro specified in condition is not enabled, the current macro is not enabled also.

The macros corresponding to the providers in **Condition** can be used. For example, you can add two more macros with name "EMP_TABLE": one with Value = EMP1 and Condition = ORACLE, another with Value = EMP2 and Condition = SQLSERVER. In this case the query

```
SELECT * FROM {EMP_TABLE}
```

is equivalent for the query with IF directives from the first example.

The Macros collection of *TUniConnection* can be used for macros adding at run-time:

```
UniConnection1.Macros.Add('EMP_TABLE', 'EMP');
UniConnection1.Macros.Add('EMP_TABLE', 'EMP1', 'ORACLE');
UniConnection1.Macros.Add('EMP_TABLE', 'EMP2', 'SQLSERVER');
```

Unified SQL defines unified literals for date, time and timestamp values. For example:

```
SELECT * FROM emp WHERE HIREDATE > {date '1982-01-15'}
```

For Oracle, this statement is converted to the following:

```
SELECT * FROM emp WHERE HIREDATE > TO_DATE('1982-01-15', 'YYYY-MM-DD')
```

Unified SQL supports also functions. Functions are marked in SQL statements using 'fn' keyword. For example,

```
SELECT {fn TRIM(ENAME)} FROM emp
```

evaluates to

```
SELECT TRIM(ENAME) FROM emp
```

it is the counterpart in the DBMS. But in MS SQL Server there is no single corresponding function, so the expression evaluates to

```
SELECT LTRIM(RTRIM(ENAME)) FROM emp
```

The treated article presented general definition of UniDAC components and their usage. For detailed information please look UniDAC documentation. The UniDAC documentation includes useful articles and a detailed reference of all UniDAC components and classes.

If you want to download trial version of UniDAC, please visit <https://www.devart.com/unidac/download.html>. For information about getting the UniDAC, visit the [How to Order](#) section. If you have a question about UniDAC or any other Devart product, contact sales@devart.com.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

3.4 Demo Projects

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

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

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

Where are the UniDAC demo projects located?

In most cases all UniDAC demo projects are located in "%UniDAC%\Demos\".

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

```
|--UniDACDemo [The main UniDAC demo project]
|--Performance [Demo project, that compares performance of UniDAC
with another components (BDE, ADO, dbExpress)]
|--ThirdParty
|  |-- [A collection of demo projects on integration with third-
party components]
|--Miscellaneous
    |-- [Some other demo projects on design technologies]
```

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

Note: This documentation describes ALL UniDAC demo projects. The actual demo projects you will have installed on your computer depend on your UniDAC version, UniDAC 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 UniDAC demo projects

To explore a UniDAC demo project,

1. Launch your IDE.
2. In your IDE, choose File|Open Project from the menu bar.
3. Find the directory you installed UniDAC 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* file for more details.

The executed version of the demo will contain a sample application written with UniDAC or a navigable list of samples and sample descriptions. To properly use each sample, you will need to connect to a working server.

The included sample applications are fully functional. To use the demos, you have to first set up a connection to a server. 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 necessary database objects. When you are done with a demo, click "Drop" to remove all objects used for the demo from your database.

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

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

Demo project descriptions

UniDACDemo

UniDACDemo is one large project which includes two collections of demos.

Working with components

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

General demos

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

UniDACDemo can be opened from %UniDAC%\Demos\UniDACDemo\unidacdemo.dpr (.bdsproj, or .dproj). The following table describes all demos contained in this project.

Working with Components

Name	Description
ConnectDialog	Demonstrates how to customize the UniDAC connect dialog . Changes the standard UniDAC 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 UniDAC 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.
Query	Demonstrates working with TUniQuery , which is one of the most useful UniDAC components. Includes many TUniQuery usage scenarios. Demonstrates how to execute queries, edit data, and export it to XML files, shows how to perform local filtering, demonstrates several different kinds of record locking and refreshing, and working with FetchAll mode. Note: This is a very good introductory demo. We recommend starting here when first becoming familiar with UniDAC.
Sql	Uses TUniSQL to execute SQL statements. Demonstrates how to work with parameters in SQL, prepare SQL statements, and create stored procedures calls by UniDAC means.
StoredProc	Uses TUniStoredProc to access editable recordsets in the client application returned from a stored procedure.
Table	Demonstrates how to use TUniTable to work with data from a single table on the server without manually writing any SQL queries. Performs server-side data sorting and filtering and retrieves results for browsing and editing.
Transaction	Demonstrates the main approaches for setting up distributed transactions with the TUniTransaction component. Shows how to manage transactions, tune the transaction isolation level, and select the coordinator for a distributed transaction.
UpdateSQL	Demonstrates using the TUniUpdateSQL component to customize update commands. Lets you optionally use TUniSQL and TUniQuery 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 the CachedUpdates mode, including highlighting uncommitted changes, managing transactions, and committing changes in a batch.
FilterAndIndex	Demonstrates UniDAC'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 UniDAC 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.
Pictures	Uses UniDAC functionality to work with BLOB fields and graphics. The sample demonstrates how to retrieve binary data from database and display it on visual components. Sample also shows how to load and save pictures to files and to the database.
Text	Uses UniDAC functionality to work with text. The sample demonstrates how to retrieve text data from 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

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

Location	Name	Description
ThirdParty	FastReport	Demonstrates how UniDAC can be used with FastReport components. This project consists of two parts. The first part is several packages that integrate UniDAC components into the FastReport editor. The second part is a demo application

		that lets you design and preview reports with UniDAC technology in the FastReport editor.
	InfoPower	Uses InfoPower components to display recordsets retrieved with UniDAC. This demo project displays an InfoPower grid component and fills it with the result of a UniDAC query. Shows how to link UniDAC data sources to InfoPower components.
	IntraWeb	A collection of sample projects that show how to use UniDAC components as data sources for IntraWeb applications. Contains IntraWeb samples for setting up a connection, querying a database and modifying data and working with CachedUpdates and MasterDetail relationships.
	QuickReport	Lets you launch and view a QuickReport application based on UniDAC. This demo project lets you modify the application in design-time.
	ReportBuilder	Uses UniDAC data sources to create a ReportBuilder report that takes data from a database. Shows how to set up a ReportBuilder document in design-time and how to integrate UniDAC components into the Report Builder editor to perform document design in run-time.
Miscellaneous	CBuilder	A general demo project about how to create UniDAC-based applications with C++Builder. Lets you execute SQL scripts and work with result sets in a grid. This is one of the two UniDAC demos for C++Builder.
	DII	Demonstrates creating and loading DLLs for UniDAC-based projects. This demo project consists of two parts - a UniDll project that creates a DLL of a form that sends a query to the server and displays its results,

		and a UniExe project that can be executed to display a form for loading and running this DLL. Allows you to build a dll for one UniDAC-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.
	Midas	Demonstrates using MIDAS technology with UniDAC. This project consists of two parts: a MIDAS server that processes requests to the database and a thin MIDAS client that displays an interactive grid. This demo shows how to build thin clients that display interactive components and delegate all database interaction to a server application for processing.
	VirtualTable eCB	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. This is one of the two demo projects for C++Builder.

© 1997-2025

Devart. All Rights Reserved.

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

3.5 Deployment

UniDAC applications can be built and deployed with or without runtime libraries. You can dynamically load packages by choosing `Link with runtime packages` in the `Project Options` dialog box. In earlier versions of Delphi, the option is called `Build with runtime packages`.

Deploying Windows Applications Built Without Runtime Packages

It is not required to deploy any files with a UniDAC application built without runtime packages, provided you have UniDAC Standard or Professional Edition. Make sure that your application does not use runtime packages (`Link with runtime packages` is not selected).

Trial Limitation Warning

If you are deploying a UniDAC Trial application, these UniDAC libraries and their dependencies are required even when your application is built without runtime packages (XX depends on the version of your IDE).

<code>dacXX.bpl</code>	mandatory
<code>unidacXX.bpl</code>	mandatory
<code>rtlXX.bpl</code>	mandatory
<code>dbrtlXX.bpl</code>	mandatory
<code>vcldbXXX.bpl</code>	mandatory

Deploying Windows Applications Built with Runtime Packages

You can build applications with runtime packages by selecting `Link with runtime packages` in `Project Options` before compiling the application. You must deploy these BPL files with your application (XX depends on the version of your IDE or the name of the provider).

<code>dacXX.bpl</code>	mandatory
<code>unidacXX.bpl</code>	mandatory
<code>XXproviderXX.bpl</code>	for each necessary provider
<code>dacvc1XX.bpl</code>	if an application uses the <code>UniDacVc1</code> unit
<code>unidacvc1XX.bpl</code>	if an application uses the <code>UniDacVc1</code> unit
<code>dacfmxXX.bpl</code>	if an application uses the <code>UniDacFmx</code> unit

unidacfmXX.bpl	if an application uses the UniDacFmx unit
XXdacXX.bpl	for each necessary provider (only in Express Edition)
crcontrolsXX.bpl	if an application uses the CRDBGridcomponent
tdsproviderXX.bpl	if an application connects in the Direct mode
odbcproviderXX.bpl	if an application connects through an ODBC driver
vqueryXX.bpl	if an application connects to DBF files

If you have UniDAC Express Edition, you must deploy `xdacXX.bpl` with your application because in Express Edition, `XXproviderXX.bpl` is just a wrapper around `xdacXX.bpl`. It is not required to deploy the `xdacXX.bpl` file with the application if you have UniDAC Professional or Standard Edition because the `XXproviderXX.bpl` file already includes the functionality of `XdacXX.bpl`.

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4 Using UniDAC

This section describes the basics of using Universal Data Access Components

- [Connecting to Database](#)
- [Updating data with UniDAC](#)
- [Master/Detail Relationships](#)
- [Data Types](#)
- [Data Type Mapping](#)
- [Data Encryption](#)
- [Working in an Unstable Network](#)
- [Disconnected Mode](#)
- [Increasing Performance](#)

- [Macros](#)
- [Connection Pooling](#)
- [DataSet Manager](#)
- [Network Tunneling](#)
- [Executing Stored Procedures](#)
- [Transactions](#)
- [Unified SQL](#)
- [DBMonitor](#)
- [Writing GUI Applications with UniDAC](#)
- [Compatibility with Previous Versions](#)
- [64-bit Development with Embarcadero RAD Studio XE2](#)
- [Database Specific Aspects of 64-bit Development](#)
- [Demo Projects](#)
- [Deployment](#)
- [C++ Builder Development for Android and iOS](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.1 Connecting to Database

This topic describes the procedure of connecting to databases with different providers, and meaning of connection parameters.

- [Common connection properties](#)
 - [Provider](#)
 - [Username and Password](#)
 - [Server](#)
 - [Database](#)
 - [Port](#)
- [Provider-specific properties](#)

- [Oracle](#)
- [SQL Server](#)
- [MySQL](#)
- [InterBase](#)
- [PostgreSQL](#)
- [SQLite](#)

Common connection properties

Each database server requires its own set of parameters for connection (username, password, etc.). Some of the parameters are the same for several servers, but the parameter meaning may vary depending on the server. UniDAC provides all types of parameters for supported database servers. If a parameter is not used for a certain provider, it will be disabled in the connection dialog and not used for connection. UniDAC supports the following parameters:

Provider

This is the first parameter that should be set. It specifies the provider that will be used for connection, and other parameters that will be available.

Username and Password

These properties are used for each database provider to authenticate the client application.

Server

Commonly this property is used to provide the name or the IP address of the computer in the network on which the database server is located. If the Server property is empty for SQL Server, MySQL, and InterBase providers, UniDAC will try to connect to localhost.

- [Oracle](#) - in the Client mode you should specify the server name which appears in the *tnsnames.ora* configuration file. You can also set the [HomeName](#) option to specify which of the installed clients to use in the Client mode.

If you are connecting to the Oracle server in the [Direct mode](#), value of the Server property should be assigned in special format: *Host:Port:SID*. Host is the server's IP address or DNS name, Port is the port number that the server listens to, SID is the Oracle System

Identifier of the server.

- SQL Server - you should specify the computer name or IP address of the computer in the network which is running SQL Server. If your SQL Server uses a port different from the default one, you can connect to it specifying the port number in the following way:
HostName,PortNumber.
- ASE, MySQL, and PostgreSQL - you should specify the computer name or IP address of the computer in the network which is running database server.
- ODBC - you should specify ODBC data source name (DSN), name of a file with data source information (File DSN), or ODBC connection string
- DB2 - you should specify the database name to the Server property

Database

This property is used for Access, Advantage, SAP Sybase ASE, DBF, InterBase, MySQL, NexusDB, PostgreSQL, SQL Server, and SQLite providers. It specifies initial database for the connection. On SAP Sybase ASE, MySQL, and SQL Server the Database value can be changed when the connection is active without reconnect. If the Database is not assigned, the behaviour of UniDAC will depend on the selected provider:

- MySQL - the current database will not be selected. It means that you will need to explicitly specify the database name in your queries.
- SQL Server and ASE - the default database for the current SQL Server login will be used as a default database for the connection. For connecting to SQL Server Compact Edition this property is used to provide the database file name.

Port

This property is used for SAP Sybase ASE, MySQL, and PostgreSQL providers. It specifies the port number for TCP/IP connection.

- MySQL - The default value is 3306.
- PostgreSQL - The default value is 5432.
- ASE - The default value is 5000.

Provider-specific properties

Along with the connection options described above, there are several specific options that manage connection behaviour for each provider. These options are described in the Provider-specific Notes articles for each provider: [Oracle](#), [SQL Server](#), [MySQL](#), [InterBase](#), [PostgreSQL](#), and [SQLite](#). Open the article that corresponds to the provider you are interested in, and find the specific options description for TUniConnection in the article. Several important specific connection options will be described below.

Oracle

With the Oracle provider you can connect to the server in two modes: the Client mode, and the Direct mode. Connecting in the Client mode requires Oracle client to be installed on the client computer. Connecting in the Direct mode does not require Oracle client, however, this mode has certain limitations. For more information, refer to the [Connecting in Direct mode](#) section in the article Using UniDAC with Oracle.

SQL Server

The SQL Server provider can connect through one of the three client types that can be changed using the [OLEDBProvider](#) specific option of TUniConnection. By default this option is set to prAuto. This value means that the provider will try to open the SQL Native Provider first. If this provider is not available, the OLE DB provider will be opened. In order to connect to SQL Server Compact Edition, the OLEDBProvider option must be set to prCompact. This value gives effect to all specific options which names start with Compact. The version of SQL Server Compact Edition to be used should be specified in the [CompactVersion](#) specific option. By default version of SQL Server Compact Edition will be chosen in accordance with the database file version. If the file does not exist, or the file is not a valid database file, the CompactVersion option will be used to determine which server version to load.

MySQL

The MySQL provider can connect to MySQL server directly or using the client library *libmysqld.dll*. This behaviour is controlled by the [Direct](#) specific option. By default, Direct is set to True. If you switch Direct to False, you will need to deploy *libmysqld.dll* with your application.

In order to connect to a database with MySQL Embedded server, you should switch the value of the [Embedded](#) specific option to True. Its default value is False. If Embedded is set to True, the value of Direct is ignored. The Embedded Server library with the share directory should be

deployed with the application. The path to data should be specified in the configuration file of Embedded Server.

InterBase

The InterBase provider can connect to the server through such network protocols as TCP/IP, NetBEUI, and SPX. The network protocol that will be used for the connection can be specified with the [Protocol](#) specific option.

PostgreSQL

The PostgreSQL provider connects to PostgreSQL server directly and does not use the PostgreSQL client library.

SQLite

The SQLite provider can connect to DB using SQLite client library SQLite3. You can use either an external SQLite3 library or embedded SQLite3 engine. This behaviour is controlled by the option. By default [Direct](#) is set to False and in this case the SQLite provider searches a client library in directories specified in the PATH environment variable. SQLite can create the database file automatically if it does not exist. For this the [ForceCreateDatabase](#) specific option should be used.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.2 Updating data with UniDAC

This topic describes common approaches to data edit with dataset components of UniDAC.

- [Automatic data updating](#)
- [Extended setup of data updating](#)
- [Caching updates](#)
- [Default values/expressions](#)
- [Autoincrement values generating](#)
- [Getting newest data on time](#)

Automatic data updating

TUniTable, TUniQuery, and TUniStoredProc are UniDAC components that allow retrieving and editing data. To edit data with each of the components, specify key field names in the [KeyFields](#) property. If KeyFields is an empty string, Oracle, PostgreSQL, InterBase, SQLite, and all ODBC-based providers will try to request information about primary keys from the server sending an additional query (this may negatively affect the performance). SQL Server and MySQL providers will use the metainformation sent by the server together with data. The SQL Server provider has the [UniqueRecords](#) option that allows automatically requesting primary key fields from the table if they were omitted in the query.

If the dataset to be opened has no fields that uniquely identify a record, this problem can be solved with Oracle, Firebird 2.0, PostgreSQL, and SQLite servers by the server means. With the Oracle and SQLite servers you should add the RowID column to your query. With Firebird 2.0 - DB_KEY. With PostgreSQL server OID column can be used as key field if your table is created with OIDs. More information about these fields you will find in the documentation of the correspondent server.

Extended setup of data updating

For a dataset having data from several tables, only one table will be updatable by default. You should specify the table name to be updatable in the [UpdatingTable](#) property, otherwise the table to which belongs the first field in the field list will be updatable. If the [SetFieldsReadOnly](#) option is set to True (by default), fields that are not used in automatically generated update SQL statements are marked read-only. With the Oracle, PostgreSQL, and all ODBC-based providers for complicated queries (statements that use multiple tables, Synonyms, DBLinks, aggregated fields) we recommend to keep the [ExtendedFieldsInfo](#) option enabled.

If Insert/Post, Update, or RefreshRecord operation has affected more than one record, UniDAC raises an exception. To suppress such exceptions, you should set the [StrictUpdate](#) option to False.

For more flexible control over data modifications you can fill update SQL statements. They are represented by the [SQLInsert](#), [SQLUpdate](#), [SQLDelete](#), and [SQLRefresh](#) properties and are executed automatically on Insert/Post, Edit/Post, Delete, and Refresh operations. At design-time you can generate default update SQL statements at the SQL Generator tab in component editor. The generated statements can be modified corresponding your needs. But if the update queries are generated dynamically for each record, only changed values are sent to the server.

For some particular cases this functionality is not enough. It can be extended with the [TUniUpdateSQL](#) component. TUniUpdateSQL allows associating a separate TUniSQL/TUniQuery/TUniStoredProc component for each update operation.

Caching updates

UniDAC allows caching updates at the client (so-called [Cashed Updates](#) mode), and then post all updates in a batch. It means that changes are not reflected at the server immediately after calling Post or Delete. All cached changes are posted to the server after calling the [ApplyUpdates](#) method. The [UpdateBatchSize](#) option lets setting up the number of changes to be posted at the same time.

Default values/expressions

If you have defined default values or expressions for columns in a database table, you can setup UniDAC so that it requests these expressions from the server. These expressions will be assigned to the DefaultExpression property of TField objects. If the DefaultExpression values have already been filled, they are replaced. This behaviour is controlled by the [DefaultValues](#) option, which is disabled by default.

Autoincrement values generating

When editing a dataset, it is often convenient not to fill key field values manually but automatically generate them. There are three ways to do it.

The first way, the most usable one, is to use server means for automatic generating of the key field values.

SQL Server, MySQL, and SQLite allow defining autoincrement columns in the table. This does not require additional handling at the client. For SAP Sybase ASE, Oracle, PostgreSQL, and InterBase providers it is necessary to specify the [KeySequence](#) ([KeyGenerator](#) for InterBase) specific option. Automatically generated values are reflected in the dataset automatically.

The second way is to generate and fill the key field value in the BeforePost event handler. As a rule this way requires executing a query to retrieve some information from the server. So this way may be useful only in some particular cases.

The third way is to create the AFTER INSERT trigger that fills the field with the appropriate value. But there is a problem with returning the value generated by the trigger. Although this

problem can be solved (see the next paragraph in this topic), this approach is considered nonoptimal. So try choosing another approach if possible.

However, retrieving generated values can be disabled for SQL Server provider with the [QueryIdentity](#) specific option. This should increase performance of records inserting.

Getting newest data on time

For certain situations UniDAC allows automatically refreshing records in the dataset in order to keep their values up-to-date.

With [RefreshOptions](#) you can make UniDAC refresh the current record before editing, after inserting or deleting. It is done by executing an additional query.

The DMLRefresh option allows refreshing the current record after insert or update similarly to RefreshOptions, but it works in a different way. This allows achieving higher performance than with RefreshOptions. DMLRefresh is not supported by the MySQL, SQLite, and ODBC-based providers.

If you want to control which fields of the current record need to be refreshed after insert or update, you should do the following: define in your update queries output parameters with names that correspond the field names in your dataset, and set the ReturnParams option to True. After the update query has been executed, dataset reads values of the output parameters and puts them into fields with the correspondent names.

© 1997-2025

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 UniDAC 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.

UniDAC provides two ways to bind tables. First code example shows how to bind two TCustomUniDataSet components into MD relationship via parameters.

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

```
// 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.4 Data Types

This topic describes in what way server data types are mapped to the Delphi field types and demonstrates common approaches for working with large data types.

The table below represents the server data types mapped to certain Delphi field types by default. There are several options that change the default mapping. These changes are reflected in the table as footnotes.

Delp hi Type	Orac le Type s	SQL Serv er Type s	MyS QL Type s [1]	Inter Base Type s	Post greS QL Type s	SQLit e Type s	ODB C Type s	DB2 Type s	Acce ss Type s	Adva ntage Type s	SAP Syba se ASE Type s	Nexu sDB
ftSm alli nt	NUMB ER(p , 0) [2] (p < 5)	SMAL LINT	TINY INT(M) (M > 1) SMAL LINT	SMAL LINT	SMAL LINT	TINY INT SMAL LINT	SQL_ SMAL LINT	SMAL LINT	SMAL LINT	SHOR T	SMAL LINT	SHOR TINT / SMAL LINT
ftWo rd	-	TINY INT	TINY INT(M) UNSI GNED (M > 1) SMAL LINT UNSI GNED YEAR	-	-	-	SQL_ TINY INT	-	BYTE	-	TINY INT	WORD / BYTE / TINY INT
ftIn tege r	NUMB ER(p , 0) [2] (4 < p < 10)	INT	MEDI UMIN T MEDI UMIN T UNSI GNED INT	INTE GER	INTE GER	INTE GER INT	SQL_ INTE GER	INTE GER	INTE GER	INTE GER	INT	INTE GER, AUTO INC, RECR EV
ftLa rgei nt	NUMB ER(p , 0) [2] (9 < p < 19)	BIGI NT	BIT INT UNSI GNED BIGI NT BIGI NT UNSI GNED	BIGI NT	BIGI NT	BIGI NT	SQL_ BIGI NT	BIGI NT	-	-	BIGI NT	LARG EINT / DWOR D
ftFl	NUMB	DECI	DECI	NUMB	DECI	DECI	SQL_	DECI	DECI	DECI	DECI	FLOA

oat	ER (p, s) [2] BINARY FLOAT (FLOAT) BINARY DOUBLE	MAL (p, s) [3] FLOAT REAL	MAL (p, s) [3] FLOAT DOUBLE PRECISION	ER (p, s) [3] FLOAT DOUBLE PRECISION	MAL [3] REAL DOUBLE PRECISION	MAL (p, s) [3] FLOAT DOUBLE PRECISION	DECIMAL (p, s) SQL_NUMERIC (p, s) SQL_REAL SQL_FLOAT SQL_DOUBLE	MAL (p, s)	MAL (p, s) DOUBLE	MAL (p, s) [3] FLOAT DOUBLE MONEY SMALLMONEY	T, DOUBLE PRECISION, EXTENDED	
ftBCD	NUMBER (p, s) [2] (p < 15) and (s < 5)	DECIMAL (p, s) [3] (p < 15) and (s < 5)	DECIMAL (p, s) [3] (p < 15) and (s < 5)	DECIMAL (p, s) [3] (p < 15) and (s < 5)	DECIMAL [3]	DECIMAL [3]	SQL_DECIMAL SQL_NUMERIC	DECIMAL	DECIMAL	DECIMAL CURDOUBLE MONEY	DECIMAL [3] MONEY SMALLMONEY	DECIMAL
ftFMTBcd	NUMBER (p, s) [2] (14 < p < 39) and (4 < s < 39)	DECIMAL (p, s) [3] (14 < p < 39) and (4 < s < 39)	DECIMAL (p, s) [3] (14 < p < 39) and (4 < s < 39)	DECIMAL (p, s) [3] (14 < p < 39) and (4 < s < 39)	DECIMAL [3]	DECIMAL [3]	SQL_DECIMAL SQL_NUMERIC	DECIMAL	DECIMAL	DECIMAL CURDOUBLE MONEY	DECIMAL [3] MONEY SMALLMONEY	-
ftCurrency	-	MONEY SMALLMONEY	-	-	MONEY	MONEY	-	-	-	-	-	MONEY

ftBoolean	-	BIT	TINY INT [4] BOOLEAN [4] BOOLEAN [4]	BOOLEAN	BOOLEAN	BOOLEAN	SQL_BIT	-	BOOLEAN	LOGICAL	BIT	BOOLEAN
ftString	VARCHAR2 NVARCHAR2 VARCHAR CHAR NCHAR RAW [5] INTERVAL DAY TO SECOND INTERVAL DAY TO MONTH ROWID UROWID	CHAR VARCHAR	CHAR VARCHAR ENUM SET BINARY [6] VARIABLE [6]	CHAR VARCHAR	CHAR VARCHAR	CHAR VARCHAR	SQL_CHAR SQL_VARCHAR	CHAR VARCHAR	TEXT	CHAR CICH VARCHAR	CHAR VARCHAR NCHAR NVAR CHAR	VARCHAR, NULL STRING, SHORT STRING, CHAR , SINGLE CHAR
ftWideString	See note [7]	NCHAR NVARCHAR	See note [7]	See note [7]	See note [7]	See note [7]	SQL_WCHAR SQL_WVARCHAR CHAR Also See note	GRAPHIC VARGRAPHIC Also See note	See note [7]	See note [7]	UNICODE UNICODE ARCHAR Also See note	NSINGLE CHAR, NCHAR, NVARCHAR CHAR

							note [7]	[7]			[7]	
ftMemo	LONG Also see note [8]	TEXT NTEXT [9]	TINY TEXT TEXT MEDIUMTEXT LONGTEXT	BLOB TEXT	TEXT	TEXT CLOB	SQL_LONG_VARCHAR	LONG_VARCHAR CLOB	MEMO	MEMO	TEXT	TEXT CLOB
ftWideMemo	See note [10]	NTEXT [11]	See note [10]	See note [10]	See note [10]	See note [10]	SQL_WLONGVARCHAR CHAR See note [10]	LONG_VARCHAR DBLOB See note [10]	See note [10]	See note [10]	UNIT EXT Also See note [10]	NCLOB
ftOraclob	CLOB NCLOB	-	-	-	-	-	-	-	-	-	-	NCLOB
ftBlob	LONG RAW	IMAGE	TINY BLOB BLOB MEDIUMBLOB LONG Spatial Data Types	BLOB BINARY	BYTE A	BLOB	SQL_LONG_VARCHAR INAR Y	LONG_VARCHAR FOR BIT DATA BLOB	-	BLOB	IMAGE	BLOB , IMAGE
ftOrablob	BLOB	-	-	-	LARGE OBJECT	-	-	-	-	-	-	-
ftBytes	-	BINARY TIMESTAMP	BINARY	-	-	-	SQL_BINARY	CHAR FOR BIT DATA	-	RAW	BINARY	BYTE ARRAY
ftVar	RAW	VARB	VARB	CHAR	-	BINA	SQL_	VARC	-	VARB	VARB	-

rBytes		INAR Y	INAR Y	VARC HAR (CHA RSET = OCTE TS)			RY VARB INAR Y	VARB INAR Y	HAR FOR BIT DATA		INAR Y	INAR Y	
ftDate	-	-	DATE	DATE	DATE	DATE		SQL_ TYPE_ _DAT _E	DATE	-	DATE	-	DATE
ftDateTime	DATE	DATE	DATE TIME	TIME STAMP	TIME STAMP	TIME STAMP DATE TIME	SQL_ TYPE_ _TIM _ESTAMP	TIME STAMP	DATE		TIME STAMP	DATE	DATE TIME
ftTime	-	-	TIME	TIME	TIME	TIME	SQL_ TYPE_ _TIM _E	TIME	-		TIME	-	TIME
ftTimeStamp	TIME STAMP TIME STAMP WITH TIME ZONE	-	-	-	-	-	-	-	-	-	-	-	-
ftCursor	REF CURSOR	-	-	-		REFC URSO R	-	-	-	-	-	-	-
ftGuid	-	UNIQ UEID ENTI FIER	-	-	-	-	-	-	-	-	-	-	GUID
ftVariant	-	SQL_ VARI _ANT	-	-	-	-	-	-	-	-	-	-	-
NOT SUPP ORTE D	BFIL E OBJE CT XML	CURS OR XML TABL E	-	-	-	-		SQL_ TYPE_ _UTC _DATE _TIME	-	-	-	-	-

							SQL TYPE _UTC TIME SQL INTE RVAL SQL GUID						
--	--	--	--	--	--	--	---	--	--	--	--	--	--

[1] - If the FieldsAsString option is True, all fields except BLOB and TEXT fields are mapped to ftString

[2] - The Oracle provider maps the NUMBER data type with different precision and scale to certain Delphi types depending on the provider options in the following way:

1. if scale equals zero, provider checks values of the specific options to choose the correct Delphi type in the following order:

- 1.1 field precision is less or equal PrecisionSmallint (default is 4) - uses ftSmallint;
- 1.2 field precision is less or equal PrecisionInteger (default is 9) - uses ftInteger;
- 1.3 field precision is less or equal PrecisionLargeInt (default is 18) - uses ftLargeint;

2. if scale is greater than zero, the appropriate Delphi type is chosen using the following sequence of rules:

- 2.1 field precision is less or equal PrecisionFloat (default is 0) - uses ftFloat;
- 2.2 EnableBCD is True and field precision, scale is less or equal PrecisionBCD (default is 14,4) - uses ftBCD;
- 2.3 EnableFMTBCD is True and field precision, scale is less or equal PrecisionFMTBCD (default is 38,38) - uses ftFMTBCD;
- 2.4 uses ftFloat.

[3] - The appropriate Delphi type is chosen using the following sequence of rules:

1. EnableBCD is True and field precision, scale is less or equal 14,4 - uses ftBCD.
When using InterBaseUniProvider, set the SimpleNumericMap option to False;
2. EnableFMTBCD is True - uses ftFMTBCD;
3. uses ftFloat.

[4] - If the EnableBoolean option is True

[5] - If the RawAsString option is True

[6] - If the BinaryAsString is True

[7] - If the UseUnicode option is True, all server types mapped to ftString will be mapped to ftWideString.

[8] - If the LongStrings option is False, and the field length is greater than 255, all server types mapped to ftString will be mapped to ftMemo.

[9] - For all Delphi versions prior to BDS 2006.

[10] - If the UseUnicode option is True, in BDS 2006 and later versions all server types mapped to ftMemo will be mapped to ftWideMemo.

[11] - For BDS 2006 and higher IDE versions.

Working with large objects

Server field types used to store large objects (BLOB, LOB, TEXT, etc.) are represented in Delphi as TBlobField and TMemoField. The TWideMemoField field was added in Delphi 2006.

- TBlobField is used to store binary objects.
- TMemoField is used to store single-byte and multibyte character data using database character set.
- TWideMemoField is used to store Unicode (UTF-16) data.

Generally there is no difference in working with these three field types in UniDAC. The Pictures and Text demos demonstrate working with datasets that contain TBlobField and TMemoField. If you want to insert a BLOB value into a table directly (without opening a dataset), please take a look at the example below. It demonstrates inserting a new record into the UniDAC_BLOB table with the TUniSQL component:

```
UniSQL.SQL.Text := 'INSERT INTO UniDAC_BLOB(ID, Title, Picture) VALUES (1, ' +  
UniSQL.ParamByName('BLOBValue').LoadFromFile('world.bmp', ftBlob);  
UniSQL.Execute;
```

If a BLOB value must be formed in your program, without using a file, and inserted into a field, you can use the LoadFromStream method:

```
var  
  Stream: TStringStream;  
begin  
  Stream := TStringStream.Create('');
```

```
try
  Stream.WriteString('The first line' + #13#10);
  Stream.WriteString('The second line');
  UniSQL.SQL.Text := 'INSERT INTO UniDAC_Text(ID, Title, TextField) VALUES
  UniSQL.ParamByName('TEXTValue').LoadFromStream(Stream, ftMemo);
  UniSQL.Execute;
finally
  Stream.Free;
end;
```

A BLOB values can be retrieved from the server in two ways. The first way is using a SELECT query from the table containing a BLOB field:

```
UniQuery.SQL.Text := 'SELECT TextField FROM UniDAC_Text WHERE ID = 1';
UniQuery.Open;
(UniQuery.FieldByName('TextField') as TBlobField).SaveToFile('A_file_name');
UniQuery.Close;
```

The second way is to use output parameters like in the following example. Note that the query may differ depending on your database server.

```
UniSQL.SQL.Text := 'SELECT :TEXTValue = TextField FROM UniDAC_Text WHERE ID
UniSQL.ParamByName('TEXTValue').ParamType := ptOutput;
UniSQL.Execute;
ShowMessage(UniSQL.ParamByName('TEXTValue').AsString);
```

See Also

- [TUniBlob](#)
- Pictures demo
- Text demo

© 1997-2025

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 is not supported, UniDAC automatically sets the correspondence between the DB data types and Delphi field types. DB data types can be manually mapped to Delphi field types using the constants for [Oracle](#), [InterBase/Firebird](#), [MySQL](#), [PostgreSQL](#), [SQL Server](#), [SQLite](#), [SQLite](#), [MS Access](#), [Advantage DB Server](#), [SAP ASE](#), [DB2](#), [DBF](#), [MongoDB](#), [NexusDB](#), [ODBC](#).

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

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

And Data Type Mapping should be used so that:

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

The above in the form of a table:

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

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

```
var
  DBType: word;
```

```

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

```

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

```

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

```

Rules order

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

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

```

CREATE TABLE NUMBER_TYPES
(
  ID NUMBER NOT NULL,
  VALUE1 NUMBER(5,2),
  VALUE2 NUMBER(10,4),
  VALUE3 NUMBER(15,6),
  CONSTRAINT PK_NUMBER_TYPES PRIMARY KEY (id)
)

```

TBCDField should be used for NUMBER(10,4), and TFMTBCDField - for NUMBER(15,6)

instead of default fields:

Oracle 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:

```
OraSession.DataTypeMap.Clear;
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, 9, r1Any, r1Any, ftF1
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, r1Any, 0, 4, ftBC
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, r1Any, 0, r1Any, ftFM
```

it will lead to the following result:

Oracle 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:

```
OraSession.DataTypeMap.Clear;
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, r1Any, 0, r1Any, ftFM
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, r1Any, 0, 4, ftBC
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, 9, r1Any, r1Any, ftF1
```

it will lead to the following result:

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

This happens because the rule

```
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, r1Any, 0, r1Any, ftFMTBCD
```

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

When using Data Type Mapping, first matching rule is searched for each type, and it is used

for mapping. In the second example, the first set rule appears to be the first matching rule for all three types, and therefore the ftFMTBCD type will be used for all fields in Delphi.

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

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

Defining rules for Connection and Dataset

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

For example, such table is created in SQL Server:

```
CREATE TABLE person
(
  id          INT          NOT NULL ,
  firstname  VARCHAR(20)   NULL   ,
  lastname   VARCHAR(30)  NULL   ,
  gender_code VARCHAR(1)   NULL   ,
  birth_dttm DATETIME     NULL   ,
  CONSTRAINT pk_person PRIMARY KEY CLUSTERED (id ASC) ON [PRIMARY]
)
GO
```

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:

```
MSConnection.DataTypeMap.Clear;
MSConnection.DataTypeMap.AddDBTypeRule(msDateTime, 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:

```
MSQuery.DataTypeMap.Clear;
MSQuery.DataTypeMap.AddDBTypeRule(msDateTime, 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:

```
MSConnection.DataTypeMap.Clear;  
MSConnection.DataTypeMap.AddDBTypeRule(msDateTime, ftDate);  
MSQuery.DataTypeMap.Clear;  
MSQuery.DataTypeMap.AddDBTypeRule(msDateTime, ftDateTime);
```

In this case, in all DataSets for the DATETIME type fields with the ftDate type will be created, and for MSQuery - 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 INT NOT NULL AUTO_INCREMENT,  
  name CHAR(50) NOT NULL,  
  guid CHAR(38),  
  PRIMARY KEY (id)  
) ENGINE=MyISAM;
```

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:

```
MyQuery.DataTypeMap.Clear;  
MyQuery.DataTypeMap.AddDBTypeRule(myChar, ftGUID);
```

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

```
MyQuery.DataTypeMap.AddFieldNameRule('guid', ftGuid);
```

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

Ignoring conversion errors

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

For example:

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:

```
IBCConnection.DataTypeMap.AddDBTypeRule(ibcVarchar, 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.

UniDAC and Data Type Mapping

When using UniDAC, there often occurs a hard-to-solve situation, when two similar types from the DB have different types in Delphi. For greater clarity, there are examples below.

e.g. there is a project, which works with two DBs: Oracle and SQL Server. There is such table created in each DB:

Oracle:

```
CREATE TABLE ITEM_INFO
(
  ID NUMBER NOT NULL,
  CODE VARCHAR2(10) NOT NULL,
  DESCRIPTION NVARCHAR2(250),
  CONSTRAINT PK_ITEM_INFO PRIMARY KEY (id)
)
```

SQL Server:

```
CREATE TABLE item_info
(
  id INT NOT NULL ,
  code VARCHAR(10) NOT NULL ,
  description NVARCHAR(250) NULL ,
  CONSTRAINT pk_item_info PRIMARY KEY CLUSTERED (id ASC)
ON [PRIMARY]
)
GO
```

The problem is due to that, when working with Oracle with the enabled UseUnicode option, both CODE and DESCRIPTION fields will have the ftWideString type, and if the UseUnicode option is disabled, both fields will have the ftString type. For SQL Server, the CODE field will always be ftString, and the DESCRIPTION field will always be ftWideString. This problem arises especially sharply when attempting to create persistent fields, because in this case, when working with one of the providers, an error will always occur. Formerly, the only way to

avoid the error was to refuse using of persistent fields in such situations.

For the time being, this problem can be solved rather easily. Data Type Mapping can be set for the Oracle provider:

```
UniConnection.DataTypeMap.Clear;  
UniConnection.DataTypeMap.AddDbTypeRule(oraVarchar2, ftString);  
UniConnection.DataTypeMap.AddDbTypeRule(oraNVarchar2, ftwideString);
```

Or Data Type Mapping can be set for SQL Server:

```
// for useUnicode = True in the Oracle data provider  
UniConnection.DataTypeMap.Clear;  
UniConnection.DataTypeMap.AddDbTypeRule(msVarchar, ftwideString);
```

or:

```
// for useUnicode = False in the Oracle data provider  
UniConnection.DataTypeMap.Clear;  
UniConnection.DataTypeMap.AddDbTypeRule(msNVarchar, ftString);
```

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.6 Data Encryption

UniDAC 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 varbinary IDENTITY (1,1) NOT NULL PRIMARY KEY,  
ENAME varbinary (2000),  
HIREDATE varbinary (200),  
SAL varbinary (200),  
FOTO VARBINARY);
```

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.

```
UniQuery.SQL.Text := 'SELECT * FROM EMP';  
UniQuery.Encryption.Encryptor := UniEncryptor;  
UniQuery.Encryption.Fields := 'ENAME, HIREDATE, SAL, FOTO';  
UniEncryptor.Password := '11111';  
UniQuery.DataTypeMap.AddFieldNameRule ('ENAME', ftString);  
UniQuery.DataTypeMap.AddFieldNameRule ('HIREDATE', ftDateTime);  
UniQuery.DataTypeMap.AddFieldNameRule ('SAL', ftFloat);  
UniQuery.Open;
```

© 1997-2025

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), UniDAC can implicitly perform the following operations:

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

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

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

See Also

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

Reserved.

4.8 Disconnected Mode

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

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

In disconnected mode a connection is opened for executing requests to the server (if it was not opened already) and is closed automatically if it is not required any more. If the connection was explicitly opened (the [Connect](#) method was called or the Connected property was explicitly set to True), it does not close until the [Disonnect](#) 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.UniDac.TUniQuery.LockMode](#)
- [TCustomDACConnection.Pooling](#)
- [TCustomDACConnection.Connect](#)
- [TCustomDACConnection.Disconnect](#)
- [Working in unstable network](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.9 Batch Operations

Data amount processed by modern databases grows steadily. In this regard, there is an acute problem – database performance. Insert, Update and Delete operations have to be performed as fast as possible. Therefore Devart provides several solutions to speed up processing of huge amounts of data. So, for example, insertion of a large portion of data to a DB is supported in the [TUniLoader](#). Unfortunately, [TUniLoader](#) 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 [TUniLoader](#), 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

For Oracle:

```
CREATE TABLE BATCH_TEST  
(
```

```

ID      NUMBER(9,0),
F_INTEGER NUMBER(9,0),
F_FLOAT  NUMBER(12,7),
F_STRING  VARCHAR2(250),
F_DATE   DATE,
CONSTRAINT PK_BATCH_TEST PRIMARY KEY (ID)
)

```

For MS SQL Server:

```

CREATE TABLE BATCH_TEST
(
  ID      INT,
  F_INTEGER INT,
  F_FLOAT  FLOAT,
  F_STRING  VARCHAR(250),
  F_DATE   DATETIME,
  CONSTRAINT PK_BATCH_TEST PRIMARY KEY (ID)
)

```

For PostgreSQL:

```

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)
)

```

For InterBase:

```

CREATE TABLE BATCH_TEST
(
  ID      INTEGER NOT NULL PRIMARY KEY,
  F_INTEGER INTEGER,
  F_FLOAT  FLOAT,
  F_STRING  VARCHAR(250),
  F_DATE   DATE
)

```

For MySQL:

```

CREATE TABLE BATCH_TEST
(
  ID      INT,
  F_INTEGER INT,
  F_FLOAT  FLOAT,
  F_STRING  VARCHAR(250),
  F_DATE   DATETIME,
  CONSTRAINT PK_BATCH_TEST PRIMARY KEY (ID)
)

```

For SQLite:

```
CREATE TABLE BATCH_TEST
(
  ID          INTEGER,
  F_INTEGER  INTEGER,
  F_FLOAT    FLOAT,
  F_STRING   VARCHAR(250),
  F_DATE     DATETIME,
  CONSTRAINT PK_BATCH_TEST PRIMARY KEY (ID)
)
```

Batch operations execution

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

```
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
```

```

UniQuery1.SQL.Text := 'INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, :F_INTEGER, :F_INTEGER, :F_INTEGER)';
// define the parameter types passed to the query :
UniQuery1.Params[0].DataType := ftInteger;
UniQuery1.Params[1].DataType := ftInteger;
UniQuery1.Params[2].DataType := ftFloat;
UniQuery1.Params[3].DataType := ftString;
UniQuery1.Params[4].DataType := ftDateTime;
// specify the array dimension:
UniQuery1.Params.ValueCount := 1000;
// populate the array with parameter values:
for i := 0 to UniQuery1.Params.ValueCount - 1 do begin
  UniQuery1.Params[0][i].AsInteger := i + 1;
  UniQuery1.Params[1][i].AsInteger := i + 2000 + 1;
  UniQuery1.Params[2][i].AsFloat := (i + 1) / 12;
  UniQuery1.Params[3][i].AsString := 'Values ' + IntToStr(i + 1);
  UniQuery1.Params[4][i].AsDateTime := Now;
end;
// insert 1000 rows into the BATCH_TEST table
UniQuery1.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:

```
UniQuery1.Execute(1000);
```

2×500 rows:

```

// insert first 500 rows
UniQuery1.Execute(500, 0);
// insert next 500 rows
UniQuery1.Execute(500, 500);

```

500 rows, then 300, and finally 200:

```

// insert 500 rows
UniQuery1.Execute(500, 0);
// insert next 300 rows starting from 500
UniQuery1.Execute(300, 500);
// insert next 200 rows starting from 800
UniQuery1.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
  UniQuery1.SQL.Text := 'UPDATE BATCH_TEST SET F_INTEGER=:F_INTEGER, F_FLOAT=:F_FLOAT';
  // define parameter types passed to the query:
  UniQuery1.Params[0].DataType := ftInteger;
  UniQuery1.Params[1].DataType := ftFloat;
  UniQuery1.Params[2].DataType := ftString;
  UniQuery1.Params[3].DataType := ftDateTime;
  UniQuery1.Params[4].DataType := ftInteger;
  // specify the array dimension:
  UniQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to 1000 - 1 do begin
    UniQuery1.Params[0][i].AsInteger := i - 2000 + 1;
    UniQuery1.Params[1][i].AsFloat := (i + 1) / 100;
    UniQuery1.Params[2][i].AsString := 'New Values ' + IntToStr(i + 1);
    UniQuery1.Params[3][i].AsDateTime := Now;
    UniQuery1.Params[4][i].AsInteger := i + 1;
  end;
  // update 1000 rows in the BATCH_TEST table
  UniQuery1.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
  UniQuery1.SQL.Text := 'DELETE FROM BATCH_TEST WHERE ID=:ID';
  // define parameter types passed to the query:
  UniQuery1.Params[0].DataType := ftInteger;
  // specify the array dimension
  UniQuery1.Params.ValueCount := 1000;
  // populate the arrays with parameter values
  for i := 0 to 1000 - 1 do
    UniQuery1.Params[0][i].AsInteger := i + 1;
  // delete 1000 rows from the BATCH_TEST table
  UniQuery1.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:

DAC Name	Operation Type	25 000 records	
		Standard Operation (sec.)	Batch Operation (sec.)
ODAC / UniDAC (with OracleUniProvider)	Insert	17.64	0.59
	Update	18.28	1.20
	Delete	16.19	0.45
LiteDAC / UniDAC (with SQLiteUniProvider)	Insert	2292	0.92
	Update	2535	2.63
	Delete	2175	0.44
PgDAC / UniDAC (with PostgreSQLUniProvider)	Insert	346.7	1.69
	Update	334.4	4.59
	Delete	373.7	2.05
IBDAC / UniDAC (with InterBaseUniProvider)	Insert	55.4	3.03
	Update	81.9	3.58
	Delete	61.3	0.91
MyDAC / UniDAC (with MySQLUniProvider)	Insert	1138	11.02
	Update	1637	26.72
	Delete	1444	17.66
SDAC / UniDAC (with SQLServerUniProvider)	Insert	19.19	3.09
	Update	20.22	7.67
	Delete	18.28	3.14

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
  UniQuery1.Params.ParamByName('ID')[i].AsInteger := i + 1;
  UniQuery1.Params.ParamByName('F_INTEGER')[i].AsInteger := i + 2000 + 1;
  UniQuery1.Params.ParamByName('F_FLOAT')[i].AsFloat := (i + 1) / 12;
  UniQuery1.Params.ParamByName('F_STRING')[i].AsString := 'Values ' + IntToStr(i);
  UniQuery1.Params.ParamByName('F_DATE')[i].AsDateTime := Now;
end

```

```
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.**

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.10 Increasing Performance

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

Connect

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

Execute

If your application executes the same query several times, you can use the [TCustomDADataset.Prepare](#) method or set the [TDADatasetOptions.AutoPrepare](#) property to increase performance. For example, it can be enabled for Detail dataset in Master/Detail relationship or for update objects in [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 TUniSQL component is more preferable than [TUniQuery](#). 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.11 Using 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](#) .

When a connection component disconnects from the database the connection actually remains active and is placed into the pool. When this or another connection component connects to the database it takes a connection from the pool. Only when there are no connections in the pool, new connection is established.

Connections in the pool are validated to make sure that a broken connection will not be returned for the [TCustomDACConnection](#) component when it connects to the database. The pool validates connection when it is placed to the pool (e. g. when the [TCustomDACConnection](#) component disconnects). If connection is broken it is not placed to the pool. Instead the pool frees this connection. Connections that are held in the pool are validated every 30 seconds. All broken connections are freed. If you set the [PoolingOptions.Validate](#) to `True`, a connection also will be validated when the [TCustomDACConnection](#) component connects and takes a connection from the pool. When some network problem occurs all connections to the database can be broken. Therefore the pool validates all connections before any of them will be used by a [TCustomDACConnection](#) component if a fatal error is detected on one connection.

The pool frees connections that are held in the pool during a long time. If no new connections are placed to the pool it becomes empty after approximately 4 minutes. This pool behaviour is intended to save resources when the count of connections in the pool exceeds the count that is needed by application. If you set the [PoolingOptions.MinPoolSize](#) property to a non-zero

value, this prevents the pool from freeing all pooled connections. When connection count in the pool decreases to [MinPoolSize](#) value, remaining connection will not be freed except if they are broken.

The [PoolingOptions.MaxPoolSize](#) property limits the count of connections that can be active at the same time. If maximum count of connections is active and some [TCustomDAConnection](#) component tries to connect, it will have to wait until any of [TCustomDAConnection](#) components disconnect. Maximum wait time is 30 seconds. If active connections' count does not decrease during 30 seconds, the [TCustomDAConnection](#) component will not connect and an exception will be raised.

You can limit the time of connection's existence by setting the [PoolingOptions.ConnectionLifeTime](#) property. When the [TCustomDAConnection](#) component disconnects, its internal connection will be freed instead of placing to the pool if this connection is active during the time longer than the value of the [PoolingOptions.ConnectionLifeTime](#) property. This property is designed to make load balancing work with the connection pool.

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

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

See Also

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

© 1997-2025
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 UniDAC 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;
```

UniDAC 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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

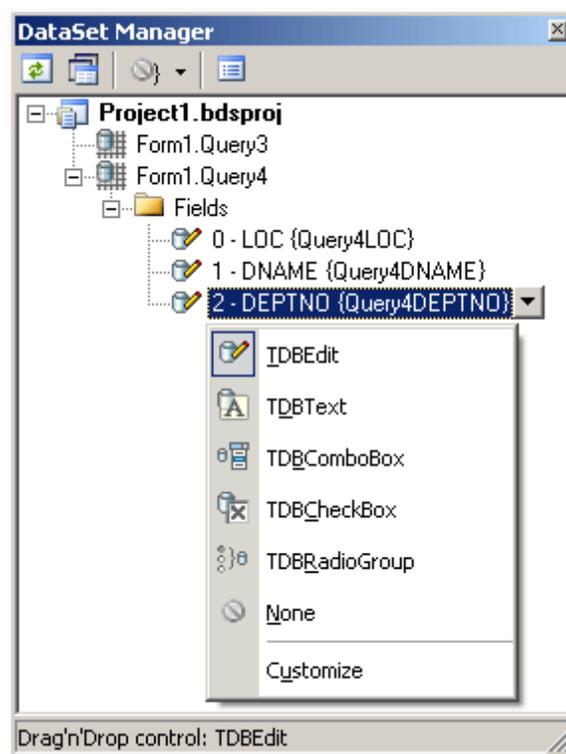
[DAC Forum](#)

[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.

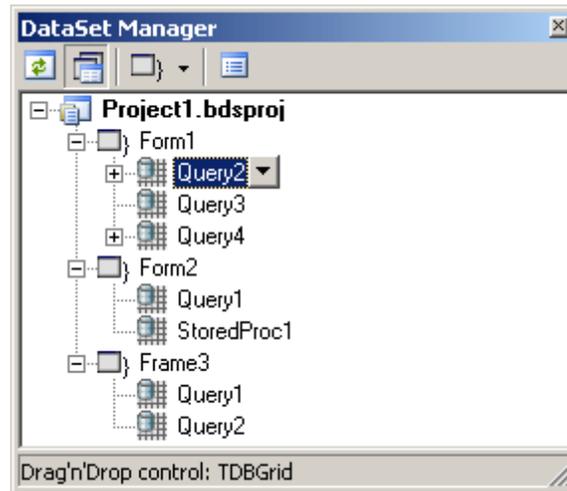
Opening the DataSet Manager window

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.

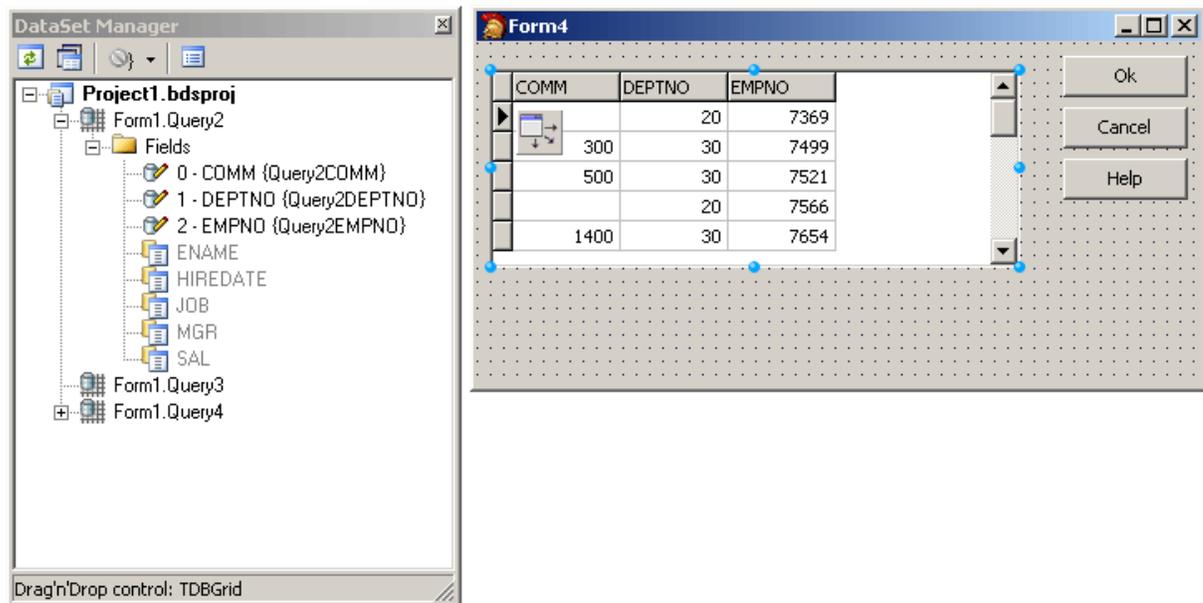
DataSets can be grouped by form or connection. To change DataSet grouping click the *Grouping mode* button or click a down. You can also change grouping mode by selecting required mode from the DataSet Manager window popup menu.



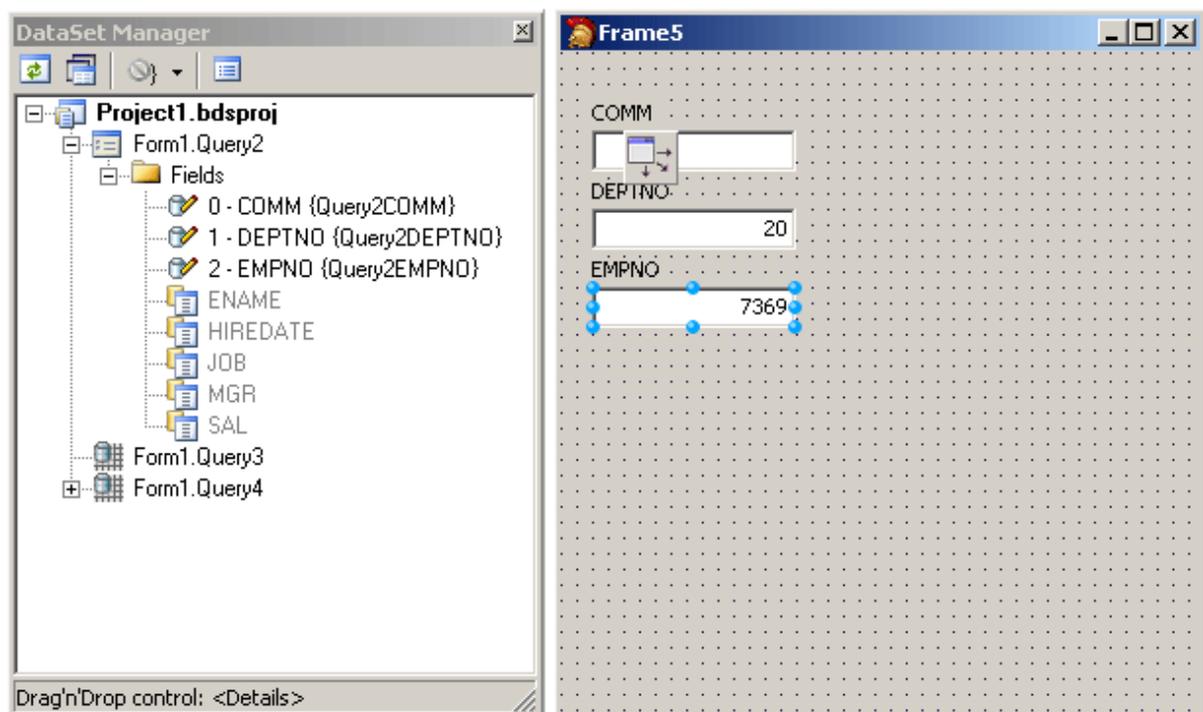
Creating Data-bound Controls

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.

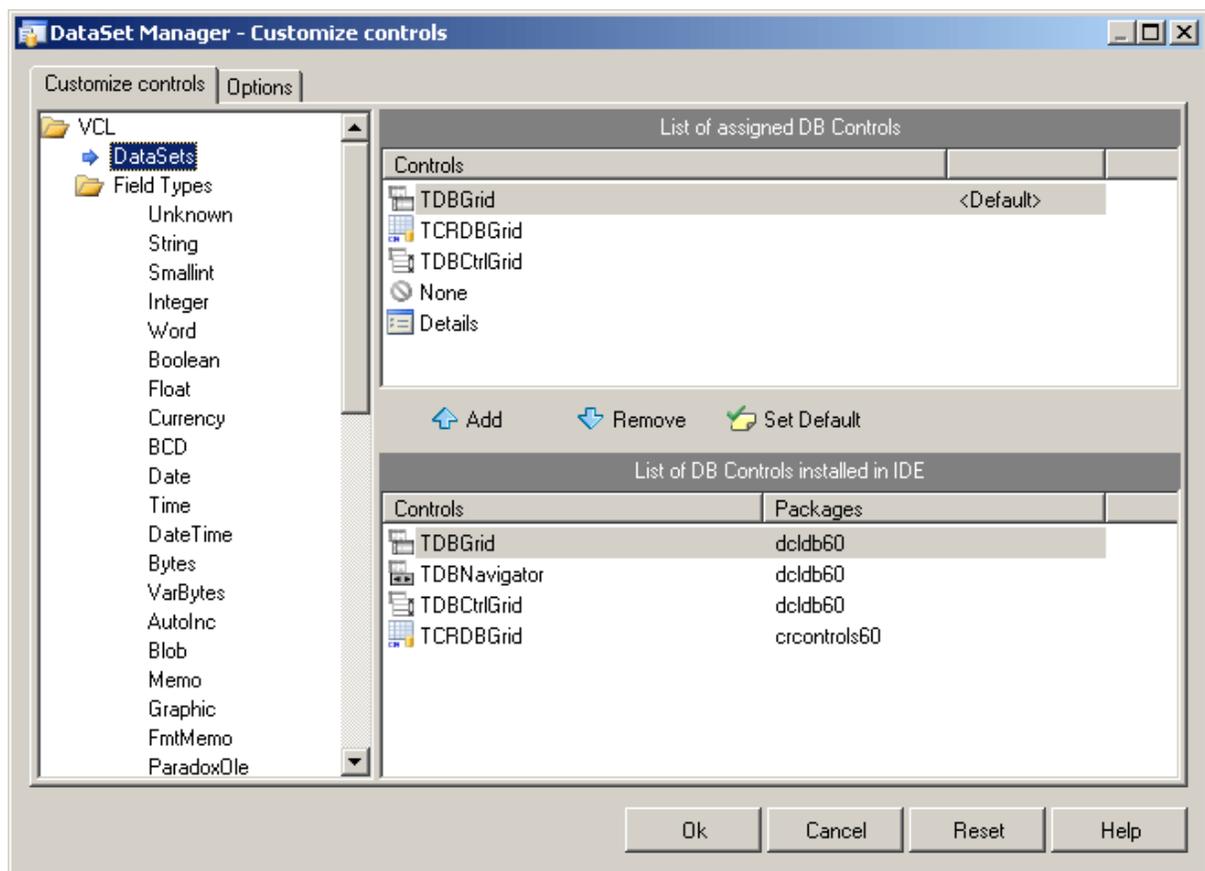


To use Details layout choose Details from the DataSet node drop-down control list in the DataSet Manager window. Then select required controls in the drop-down control list for each DataSet field. DataSet fields must be created. After setting required options you can drag the DataSet to the form from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

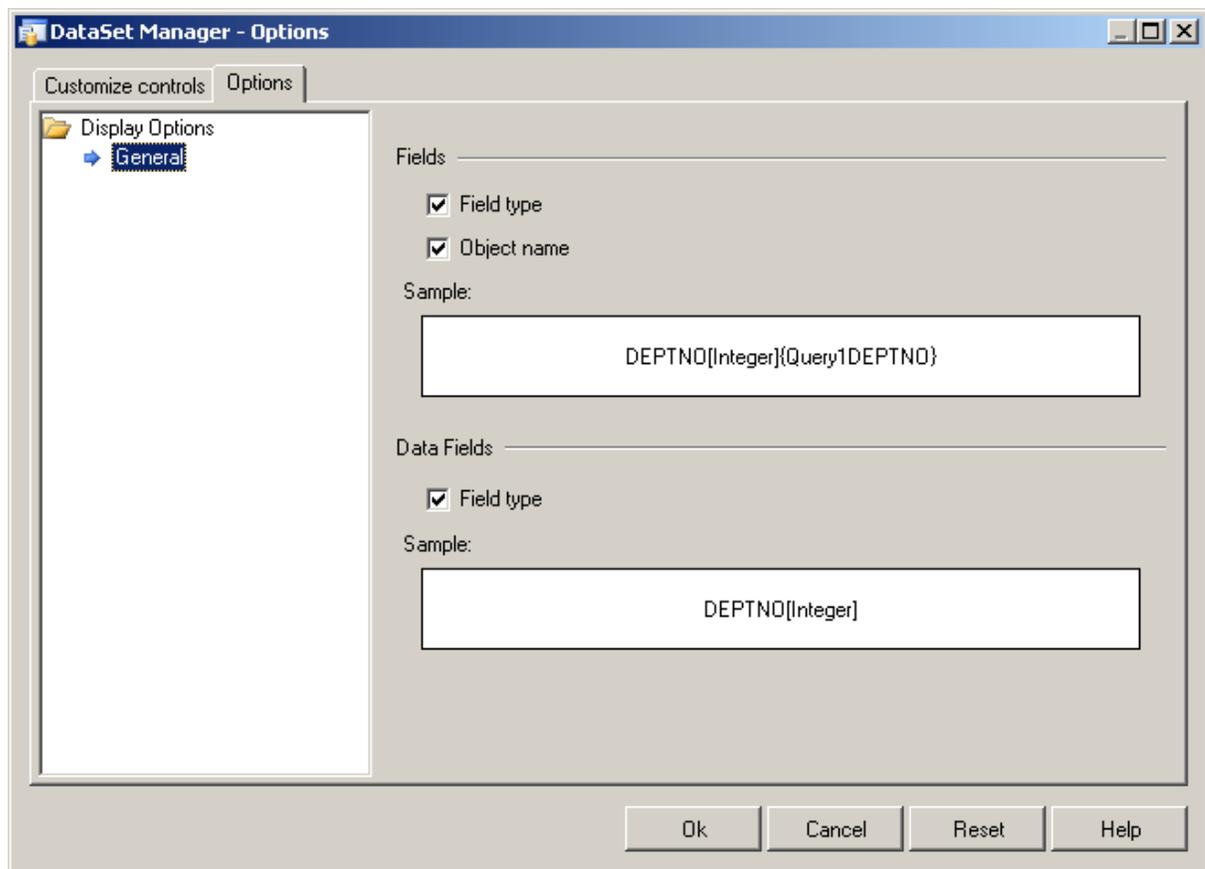
DataSet Manager allows you to create and remove TField objects. DataSet must be active to work with its fields in the DataSet Manager. You can add fields, based on the database table columns, create new fields, remove fields, use drag-n-drop to change fields order.

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.

Devart. All Rights Reserved.

4.14 Executing Stored Procedures

This topic describes approaches for executing stored procedures with UniDAC.

- [What component to choose?](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery](#)
 - [TUniStoredProc](#)
- [Usage of stored procedure parameters](#)
 - [Parameter types](#)
 - [Passing default parameter values](#)

Stored procedures in UniDAC can be executed with one of the following components: [TUniConnection](#), [TUniSQL](#), [TUniQuery](#), [TUniStoredProc](#). Below you will find the description of working with stored procedure using these components starting with the simplest approach.

TUniConnection

The simplest way to execute a stored procedure is the TUniConnection component, but it has several limitations. TUniConnection does not have properties like SQL, StoredProcName, or Params. So you will need to provide stored procedure name and parameter values each time you need to execute it. TUniConnection does not support output parameters, however you can get a result parameter from a function. Also TUniConnection does not support preparation. Stored procedures are executed with the [ExecProc](#) and [ExecProcEx](#) methods.

Therefore, if you need to execute a stored procedure that returns neither record set nor output parameters only once, the TUniConnection component is an optimal choice.

TUniSQL

TUniSQL is a separate component dedicated to execute commands that do not return record sets. It has no data storage, therefore it consumes a bit less memory than TUniQuery or TUniStoredProc and works a bit faster. To execute a stored procedure, an appropriate

command must be assigned to the SQL property of TUniSQL. It can be assigned manually, or created with the [CreateProcCall](#) method.

The CreateProcCall method accepts a stored procedure name, gets the description of a stored procedure from the server, and generates SQL command with parameters. The generated command is automatically assigned to the [SQL](#) property. Parameters can be accessed both at design time and run time using properties such as Params, ParamByName, etc.

Comparing to the previous method of stored procedures execution, TUniSQL supports all kinds of parameters (INPUT, OUTPUT, etc.). For repeatable executions of a stored procedure, you do not need to pass a SQL command on each execution. It is stored in the SQL property.

Each command of TUniSQL can be [prepared](#). In some cases preparation improves performance of execution.

TUniSQL is a powerful component that is an appropriate choice for a stored procedure that does not return result sets, needs to be executed multiple times, or returns output parameters.

TUniQuery

One more component that lets you execute stored procedures is [TUniQuery](#). In addition to the abilities provided by TUniSQL, TUniQuery allows to obtain record sets from stored procedures and modify them. If a stored procedure returns multiple record sets, all of them can be accessed sequentially. The Open method opens the first record set. The [OpenNext](#) method closes the current record set and opens the next one. If the server has sent enough meta-information about the query, obtained dataset will be editable. Otherwise to get an editable dataset you should setup properties such as [SQLDelete](#), [SQLInsert](#), and others properly.

The TUniQuery is a good choice for executing stored procedures that return record sets.

TUniStoredProc

TUniStoredProc is a component designed specially for working with stored procedures. If you want to execute a stored procedure, just assign its name to the [StoredProcName](#) property, call [PrepareSQL](#) to describe parameters, assign parameter values, and call [Execute](#). If the stored procedure has no input or input/output parameters to be assigned, call to the PrepareSQL method is not necessary. Other than that TUniStoredProc is similar to

TUniQuery. It supports result sets, output parameters, preparation, and can be initialized by the CreateProcCall method.

TUniStoredProc is the most convenient component for working with stored procedures that covers all necessary functionality.

There are several notes concerning parameters of stored procedures.

Parameter types

UniDAC supports four parameter types: input, output, input/output, and result.

TUniConnection can pass values of the input parameters to the server, and get the result value from a function. If a parameter value is not assigned, the default value will be provided if possible. If an unassigned parameter has no default value, an error will be raised.

TUniSQL, TUniQuery, and TUniStoredProc components can handle all of these parameter types. If an input parameter value is not assigned with one of these components, the NULL value will be passed as a parameter value. Assigning of output and result parameter values has no effect as they are not passed to the server on execution, and after execute they will be replaced with values returned from the server.

Passing default parameter values

Some stored procedures may have default values for parameters. If you want to pass a default parameter value to a stored procedure, you should do the following:

- with TUniConnection call the [ExecProcEx](#) method omitting the names and values of the parameters to be initialized with their default values;
- with TUniConnection call the [ExecProc](#) method omitting values of the last parameters to be initialized with their default values;
- with other components set the Bound property of the parameter to be initialized with its default value to False.

If a parameter value in TUniSQL, TUniQuery, or TUniStoredProc is not assigned or cleared, the NULL value will be passed as a parameter value. It is not the same as assigning a default value.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.15 Transactions

This topic describes how transaction support is implemented in UniDAC. So, you should be pretty familiar with transactions to understand how to control them with UniDAC.

The local transactions are managed by the TUniConnection component with [StartTransaction](#), [Commit](#), [Rollback](#), and other methods. Each time you are about to start a transaction, you should check whether it is active. You can do this using the [InTransaction](#) property. Call to StartTransaction when the transaction is already active will cause an exception. Here is a short example that demonstrates the general approach for working with local transactions:

```
if not UniConnection.InTransaction then
  UniConnection.StartTransaction;
try
  // Do some actions with database. For example:
  UniSQL1.Execute;
  UniSQL2.Execute;
  // Commit the current transaction to reflect changes in database if no
  UniConnection.Commit;
except
  // Rollback all changes in database made after StartTransaction if an e
  UniConnection.Rollback;
end;
```

After you have activated a transaction, all operations, including dataset opening, will be performed within the context of the current transaction until you commit or rollback it. If no transactions were started, changes performed by each operation are reflected in database right after the operation is completed (so-called AutoCommit mode). When using InterBase provider, please pay attention to the AutoCommit property. The AutoCommit property has the True value by default that leads to automatically execution of CommitRetaining or RollbackRetaining when there is any data modification. By setting the property to False, you will get rid of this behavior, however, you will have to manage the transactions by yourself. The TUniConnection.AutoCommit property has a higher priority than the specific option "AutoCommit" of datasets (TUniQuery, TUniTable). If the TUniConnection.AutoCommit property is set to False, all transactions can be committed only explicitly (despite of the specific option "AutoCommit" value of a dataset). If you want most datasets to automatically commit transactions, and for some of them to control transactions manually, you should set the TUniConnection.AutoCommit property to True, and only for datasets with manual transaction control, set the specific option "AutoCommit" value to False.

The behaviour of each explicitly started transaction can be customized with parameters

passed to the overloaded `StartTransaction` method. You can specify the isolation level for the transaction and whether this transaction will be editable. There is a more detailed description of these parameters in the [StartTransaction](#) topic.

UniDAC also supports working with Savepoints. The [Savepoint](#) method lets you to define a named savepoint within a transaction. You can use the savepoint name in the [RollbackToSavepoint](#) method to rollback changes in the database to the actual state at the point of time the savepoint was made. Call to `RollbackToSavepoint` keeps the current transaction active.

The [CommitRetaining](#) and [RollbackRetaining](#) methods are similar to `Commit` and `Rollback`, but they keep the current transaction active. It means that you will not need to call `StartTransaction` to keep working in transaction like you do with the `Commit` and `Rollback` methods. Functionality of `CommitRetaining` and `RollbackRetaining` is supported by InterBase/Firebird/Yaffil servers. For other servers this functionality is emulated by subsequent call to `StartTransaction` after `Commit` or `Rollback`.

InterBase-like servers support several simultaneous active transactions within a single connection and require a transaction to be active when opening a cursor. You should not take care of this, as UniDAC encapsulates these peculiarities letting you work in a way similar to the way of working with other database servers. If you want to involve abilities of InterBase servers to run parallel transactions, you should place several [TUniTransaction](#) components onto the form and setup properties of [TCustomUniDataSet](#) descendants such as [Transaction](#) and [UpdateTransaction](#) with these components. The `Transaction` and `UpdateTransaction` properties are used only for the InterBase provider. For other providers these properties are ignored.

UniDAC uses MTS to manage distributed transactions with Oracle and Microsoft SQL Server connections. Distributed transactions are controlled by the `TUniTransaction` component. You can add connections to a distributed transaction context using the [AddConnection](#) method. The MTS distributed transaction coordinator allows mixing connections both to different servers and different server kinds.

```
begin
  UniConnection1.Connect;
  UniConnection2.Connect;
  UniTransaction.AddConnection(UniConnection1);
  UniTransaction.AddConnection(UniConnection2);
  UniTransaction.StartTransaction;
  UniSQL1.Connection := UniConnection1;
  UniSQL2.Connection := UniConnection2;
```

```
try
  UniSQL1.Execute;
  UniSQL2.Execute;
  UniTransaction.Commit;
except
  UniTransaction.Rollback;
end;
end;
```

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.16 Unified SQL

One of the most crucial problems in programming applications for several databases is that SQL syntax can be different in many situations. This article demonstrates how UniDAC helps to overcome this issue.

Database applications operate data using SQL statements. Unless entered directly by the user, the statements can be constructed in one of two ways, either hard-coded during development, or constructed at run time. The first way is very convenient for developer, while the second way is far more flexible. UniDAC allows to take best from both approaches: you can hard-code SQL statements that are transformed into appropriate syntax in run time.

- [General Information](#)
- [Macros](#)
- [Conditional Execution \(IF\)](#)
- [Literals and Identifiers](#)
- [Comments](#)
- [SQL Functions](#)
- [Macros Reference](#)

General Information

Universal capabilities of UniDAC are based on the following features:

- **Macros** that have values specific for different databases (providers). In addition to predefined macros you can define your own.
- Set of automatically mapped **functions**.

- Unified standard of **literals**.

Knowing this, you can write truly database-independent SQL code interpreted in run time.

Macros

UniDAC offers two approaches to working with macros: Connection Macros and DataSet Macros. They differ by the way they are defined and by the way they are indicated in the SQL query text.

DataSet Macros are defined by "&MacroName" and affect only the specified dataset.

Connection Macros are defined by "{MacroName}" and affect all associated datasets.

Lets make more detailed analysis of TUniConnection.Macros. You can work with it in the traditional way:

```
if UniConnection.ProviderName = 'Oracle' then
  UniConnection.MacroByName('tablename').Value := 'dept'
else
if UniConnection.ProviderName = 'MySQL' then
  UniConnection.MacroByName('tablename').Value := 'test.dept';
```

Or you can use predefined approach.

Macro is a set of name, condition and value. Macro evaluates to its value if underlying condition is enabled, or to an empty string if the condition is not enabled. Conditions are enabled or disabled depending on a provider used by the TUniConnection component. For example, if you use the Oracle provider, ORACLE macro will be enabled.

Consequently, all macros that base on *Oracle* conditions return their value when used in SQL statements; all other macros return empty string.

For list of available conditions (in other words, predefined macros) refer to the [Macros Reference](#).

From API point of view, macros are represented as [TUniMacro](#) class. Collections of macros are organized into [TUniMacros](#), which can be accessed through the [Macros](#) property of [TUniConnection](#). Each connection has individual set of macros.

The following examples demonstrate usage of macros:

```
UniConnection.Provider = 'MySQL';
...
UniConnection.Open;
UniConnection.Macros.Add('tablename', 'test.dept', 'MySQL');
```

```
UniQuery.SQL.Text := 'SELECT Count(*) FROM {tablename}';  
UniQuery.Open;
```

Now suppose we need to do the same on an Oracle server. Due to usage of UniSQL the only thing to add is another macro:

```
UniConnection.Provider = 'Oracle';  
...  
UniConnection.Open;  
UniConnection.Macros.Add('tablename', 'test.dept', 'MySQL');  
UniConnection.Macros.Add('tablename', 'dept', 'Oracle');  
UniQuery.SQL.Text := 'SELECT Count(*) FROM {tablename}';  
UniQuery.Open;
```

As you see, it is very easy to control SQL statements transformation. Now let's take a look at another example that demonstrates a whole pack of important features:

```
UniConnection.Macros.Add('tablename', 'emp', '');  
//For MySQL, prepend database name  
UniConnection.Macros.Add('tablename', 'test.emp', 'MySQL');  
//Limit records count where it is easy (MySQL and PostgreSQL)  
UniConnection.Macros.Add('limit', 'LIMIT 0,5', 'MySQL');  
UniConnection.Macros.Add('limit', 'LIMIT 5 OFFSET 0', 'PostgreSQL');  
//Define default FROM clause  
UniConnection.Macros.Add('from', 'FROM {tablename}', '');  
//If the limit macro is defined, add extra clause  
UniConnection.Macros.Add('from', 'FROM {tablename} {limit}', 'limit');  
//Define query that uses the macro  
UniQuery.SQL.Text := 'SELECT EName, Job, Sal {from}';  
UniQuery.Open;
```

Supposed that in this sample connection is made to MySQL server, the executed statement would be

```
SELECT EName, Job, Sal FROM emp LIMIT 0,5
```

Note: you can use [DBMonitor](#) application to see what your query turns into on execution.

A step-by-step analysis of the sample reveals following important notes:

1. If a macro has blank condition, it is always evaluated.
2. Macro with enabled condition overrides macro with blank condition.
3. Conditions are case-insensitive.
4. You can use your own macros as conditions.
5. You can use macros as part of the value of other macros.

You can add any text after macros name inside braces. This text is added to final SQL statement if macro's condition is enabled. For example:

```
UniConnection.Macros.Add('schema', 'test', 'MySQL');
```

```
UniQuery.SQL.Text := 'SELECT * FROM {schema .}emp';
UniQuery.Open;
```

In this example a dot is added only when SCHEMA macro is enabled.

UniDAC has set of useful predefined macros that help you write universal statements. Please refer to [Macros Reference](#) for more information.

Conditional Execution (IF)

For the purpose of extra flexibility UniSQL supports conditional inclusion of SQL code into resulting statements. This is as simple as that:

```
{if my_macro} STATEMENT_1 {else} STATEMENT_2 {endif}
```

If macro *my_macro* is defined, the *STATEMENT_1* is returned, otherwise *STATEMENT_2* is the result of the expression. For instance:

```
{if Oracle}
SELECT * FROM dept
{else}
SELECT * FROM test.dept
{endif}
```

The {else} clause can be omitted. Here is a bit more sophisticated example:

```
SELECT {if Oracle}RowId, {endif} DeptNo, DName FROM dept
```

Note that you can use nested {if...} constructs to continue branching. Also you can use [predefined](#) macros.

Literals and Identifiers

UniDAC provides universal syntax for dates, timestamps and quoted identifiers. Its usage is similar to usage of macros. Note that this functionality is not available for OLE DB, ODBC, and DB2 data providers.

Date and time constants

In date/time constants parts of date are separated with hyphen, time parts are separated with colon, and space is expected between the two parts. The following table illustrates date/time format:

Literal type	Format	Example
date	yyyy-mm-dd	{date '2006-12-31'}
time	hh:mm:ss	{time '23:59:59'}

timestamp	yyyy-mm-dd hh:mm:ss	{timestamp '2006-12-31 23:59:59'}
-----------	---------------------	-----------------------------------

The following SQL statement:

```
SELECT * FROM emp WHERE HIREDATE>{date '1982-01-15'}
```

in MySQL evaluates to

```
SELECT * FROM emp WHERE HIREDATE>CAST('1982-01-15' AS DATETIME)
```

and in Oracle it turns to

```
SELECT * FROM emp WHERE HIREDATE>TO_DATE('1982-01-15', 'YYYY-MM-DD')
```

Universal quoting of identifiers

All database servers support quoting for identifiers that contain special symbols like spaces or dots. UniDAC allows to wrap identifiers universally so that quotation is appropriate for every database server. Use the following syntax:

```
"identifier"
```

For example, expression *"table1"."field1"* turns into *"table1"."field1"* in Oracle and PostgreSQL, into *[table1].[field1]* in MS SQL Server, and into *`table1`.`field1`* in MySQL server. Do not confuse with single quotes, which are intended to wrap string constants.

Comments

Comments are inserted in UniSQL with two hyphens (comments out the text till the end of current line). For multiline comment, wrap it into */*...*/* sequences. Example:

```
--This is a single-line comment
/*This one
 spans over
 several lines*/
```

SQL Functions

UniDAC introduces standard for calling common SQL functions. This is set of function names with fixed meaning. In run time the function is transformed either to corresponding native function, or to equivalent expression (for example, several functions). The construct syntax is

```
{fn Function_Name(parameter1 [,parameter2 ... ])}
```

For example, the following fragment

```
SELECT {fn TRIM(ENAME)} FROM emp
```

evaluates to

```
SELECT TRIM(ENAME) FROM emp
```

in MySQL, because there is the counterpart in the DBMS. But in MS SQL Server there is no single corresponding function, so the expression evaluates to

```
SELECT LTRIM(RTRIM(ENAME)) FROM emp
```

The following table lists unified functions and describes them briefly.

Function name	Description
System routines	
USER	Returns current user name.
String routines	
CHAR_LENGTH(string_exp)	Returns length of string expression in characters.
LOCATE(string_exp1, string_exp2)	Finds first occurrence of substring <i>string_exp1</i> in string expression <i>string_exp2</i> .
SUBSTRING(string_exp, start, length)	Returns substring from specified string <i>string_exp</i> .
CONCAT(string_exp1, string_exp2)	Concatenates several string expressions.
CHAR(code)	Converts integer values into characters.
TRIM(string_exp)	Removes leading and trailing spaces from a string.
UPPER(string_exp)	Returns <i>string_exp</i> , with all letters uppercase.
LOWER(string_exp)	Returns <i>string_exp</i> , with all letters lowercase.
Number routines	
TRUNCATE(numeric_exp, integer_exp)	Returns <i>numeric_exp</i> truncated to <i>integer_exp</i> places right of the decimal point.
CEILING(numeric_exp)	Returns the smallest integer value not less than <i>numeric_exp</i> .
Date and time routines	
CURRENT_DATE	Returns date part of current timestamp, that is, year, month and day.
YEAR(date_exp)	Extracts year part of a timestamp.
MONTH(date_exp)	Extracts month part of a timestamp.
DAY(date_exp)	Extracts day part of a timestamp.
DATEADD(datepart, number, date)	Returns a new datetime value based on adding an interval to the specified <i>date</i> .

	The interval is formed as <i>number of datepart</i> units. The following example adds two years to HireDate field: SELECT {fn DATEADD(year,2,HireDate)} FROM emp
DATEDIFF (datepart, startdate, enddate)	Returns the number of date and time boundaries crossed between two specified dates.
Conversion routines	
TODATE(string exp)	Converts value to date format.
TOCHAR(any type exp)	Converts value to string format.
TONUMBER(string exp)	Converts value to number format.

Macros Reference

The following table enumerates names of predefined macros that are enabled depending on DBMS server connected and provider used.

Provider	Macro name
Adaptive Server Enterprise	ASE
Advantage Database Server	Advantage
DB2	DB2
InterBase	InterBase
Microsoft Access	Access
MySQL	MySQL
ODBC	ODBC
Oracle	Oracle
PostgreSQL	PostgreSQL
SQLite	SQLite
SQL Server	SQLServer
DBF	DBF
NexusDB	NexusDB

There are also predefined macros that help to solve most common differences in SQL syntax. The following table enumerates them and gives translation for some databases.

Macro name	VARCHAR	DOUBLE	DATETIME	PROVIDER
Remarks	Evaluates to database type	Evaluates to database type	Evaluates to database type	Evaluates to the name of currently

	that represents string values. Used mainly in CAST expressions.	that represents floating point values. Used mainly in CAST expressions.	that represents date and time values. Used mainly in CAST expressions.	used provider
Adaptive Server Enterprise	VARCHAR	FLOAT	DATETIME	ASE
Advantage	VARCHAR	DOUBLE	TIMESTAMP	Advantage
DB2	VARCHAR	DOUBLE	TIMESTAMP	DB2
InterBase	VARCHAR	DOUBLE PRECISION	TIMESTAMP	InterBase
Microsoft Access	VARCHAR	DOUBLE	DATE	Access
MySQL	VARCHAR	DOUBLE	DATETIME	MySQL
ODBC	VARCHAR	DOUBLE	TIMESTAMP	ODBC
Oracle	VARCHAR2	NUMBER	DATE	Oracle
PostgreSQL	VARCHAR	DOUBLE PRECISION	TIMESTAMP	PostgreSQL
SQLite	VARCHAR	DOUBLE PRECISION	TIMESTAMP	SQLite
SQL Server	VARCHAR	FLOAT(53)	DATETIME	SQL Server
DBF	VARCHAR	DOUBLE	DATE	DBF
NEXUS	VARCHAR	DOUBLE	DATETIME	NexusDB

[Working with Macros](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.17 DBMonitor

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

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 [TUniSQLMonitor](#) 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.18 Writing GUI Applications with UniDAC

UniDAC 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 UniDacVcl unit in your application. This feature is needed for writing console applications.

Delphi and C++Builder

By default UniDAC does not require Forms, Controls and other GUI related units. Only [TUniConnectDialog](#) components require the Forms unit.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.19 Syntax Highlighting in Component Editors

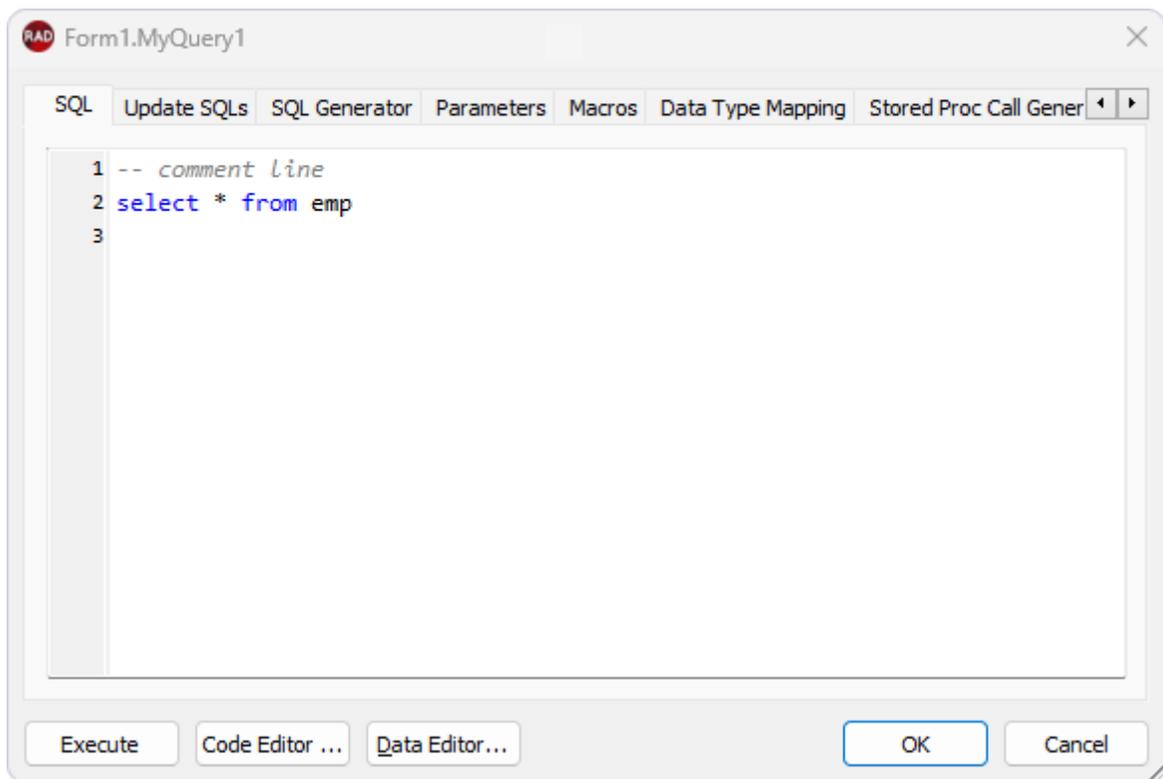
UniDAC supports syntax highlighting in component editors at design time. This topic explains how to enable or disable this feature in UniDAC component editors.

- [Enable syntax highlighting](#)
 - [Delphi](#)
 - [Lazarus](#)
- [Disable syntax highlighting in Delphi](#)

Enable syntax highlighting

Delphi

To enable syntax highlighting for Delphi, install the freeware SynEdit component set. You can download SynEdit from the [official website](#) or install it using the [GetIt Package Manager](#). Once installed, syntax highlighting will be enabled automatically in all component editors.



Lazarus

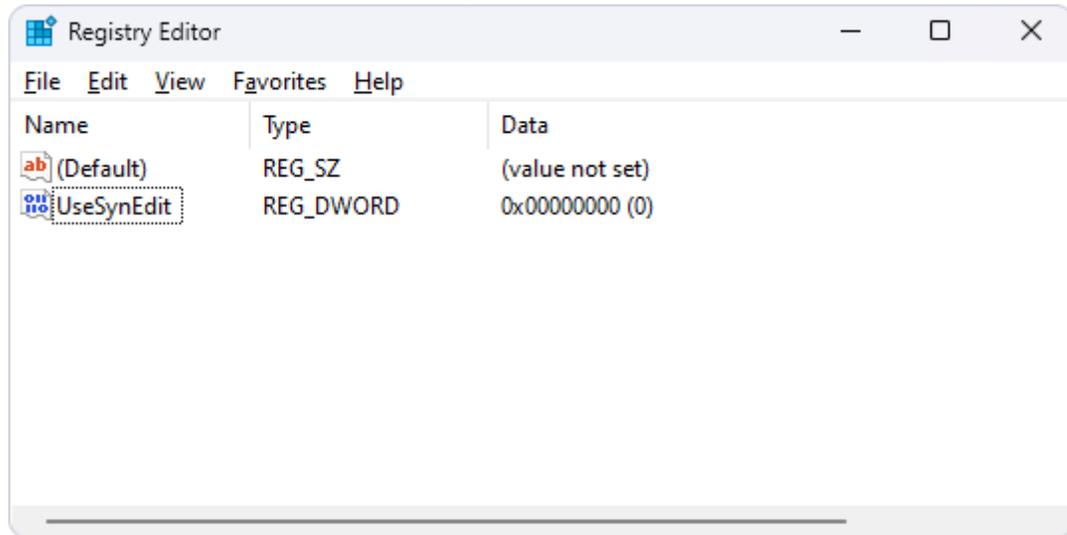
In Lazarus, SynEdit is a system package integrated into the IDE. This means that syntax highlighting for UniDAC component editors is enabled automatically and requires no additional setup.

Disable syntax highlighting in Delphi

If you have SynEdit installed but prefer not to use syntax highlighting in UniDAC component editors, you can disable it in the following way:

1. Open the Registry Editor.

2. Navigate to the **HKEY_CURRENT_USER\Software\Devart\UniDAC\Editors** registry key.
3. Create a **DWORD** value named **UseSynEdit** and set its value to **0**.



Syntax highlighting is now disabled for UniDAC. To enable syntax highlighting again, delete the **UseSynEdit** value or set it to 1.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.20 Compatibility with Previous Versions

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

DBAccess.SQLGeneratorCompatibility:

If the manually assigned [RefreshSQL](#) property contains only "WHERE" clause, UniDAC 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 UniDAC . To restore the old behavior, set the BaseSQLOldBehavior variable to True.

MemDS.SendDataSetChangeEventAfterOpen:

Starting with UniDAC , 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.DoNotRaiseExcetionOnUaFail:

Starting with UniDAC , 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.

Uni.OldTransactionBehaviour:

Since UniDAC version 5.0.1, the DefaultTransaction transaction property was added. All datasets that use the TUniConnection component, use its DefaultTransaction for all operations under data. In earlier UniDAC versions, all the datasets that used TUniConnection, used implicitly created internal transaction. This transaction always remained open, and it was not possible to control it. To restore the old behaviour, set OldTransactionBehaviour to True.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.21 64-bit Development with Embarcadero RAD Studio XE2

RAD Studio XE2 Overview

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

Its main features are the following:

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

Changes in 64-bit Application Development

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

General

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

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

For the same reason, if your design-time packages require that certain DLLs be loaded, you

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

Extended type

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

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](http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Running_the_Platform_Assistant_on_Windows)

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

4.22 C++ Builder Development for Android and iOS

This chapter discusses the basics of developing database applications for iOS and Android in C++ Builder using UniDAC. C++ Builder supports iOS and Android application development since version XE6. You can access a database from iOS or Android in almost the same way you access it from Windows, but you should be aware of some aspects of connecting and deploying files to a mobile device when working with a local database. This article contains connection instructions and sample code for each database supported by UniDAC.

Connecting to a Database in Design-Time

To create an Android application that connects to MySQL, select `File > New > Multi-Device Application - C++ Builder`. Select `Blank Application`, then place the `TUniConnection` and `TMySQLUniProvider` components onto the form. Set the `ProviderName` property of `TUniConnection` to `MySQL` and assign values to the `Username`, `Password`, `Server`, and `Port` properties in the Object Inspector. You can test the database connectivity by setting the `Connected` property to `True`. If the values specified are correct, you will be able to view the list of available databases in the `Database` dropdown.

Compiling the Project

Select `Project > Add to Project...` and add the database provider library for Android 64-bit, which is located in "`C:\Program Files (x86)\Devart\UniDAC for RAD Studio 10.3\Lib\Android64`". For C++ Builder 10.3 Rio, the filename of the MySQL provider is `libmyprovider260.a`. Compile the project.

The table below contains database servers and their corresponding provider libraries for mobile application development in C++ Builder 10.3 Rio using UniDAC.

Database System	Standard Edition	Professional Edition
ASE	libbaseprovi	libtdsprovi

	der260.a	der260.a
SQL Server	libmsprovi der260.a	libtdsprovi der260.a
SQLite	libliteprovi der260.a sqlite3.o	
MySQL	libmyprovi der260.a	
Oracle	liboraprovi der260.a	
PostgreSQL	libpgprovid er260.a	
InterBase ToGo	libibprovid er260.a	
Amazon Redshift	librsprovid er260.a libpgprovid er260.a	
xBase	libdbfprovi der260.a libvquery2 60.a sqlite3.o	

Connecting in Run-Time

Put the needed providers onto the form and add their library files (similar to what you did in design-time). Note that despite having the same name, the provider libraries for Android and iOS are different and located in their respective folders:

"C:\Program Files (x86)\Devart\UniDAC for RAD Studio 10.3\Lib\Android64"

"C:\Program Files (x86)\Devart\UniDAC for RAD Studio 10.3\Lib\iOSDevice64"

Place the `TUniConnection` component onto the form or add the following lines to the header file:

```
#include "DBAccess.hpp"
#include "Uni.hpp"
```

and the following lines to the .cpp file:

```
#pragma link "DBAccess"
#pragma link "Uni"
```

If you are planning to use a local database on a mobile device, add this line to the header file

to get access to the IOUtils namespace.

```
#include <System.IOUtils.hpp>
```

ASE

ASE has no client for Android or iOS, therefore a connection to an ASE server can only be established directly via TCP/IP by setting the `Direct` property to `True`.

```
TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "ASE";
    Connection->Server = "server";
    Connection->Username = "user_name";
    Connection->Password = "password";
    Connection->Database = "database_name";
    Connection->SpecificOptions->Values["Direct"] = "True";
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}
```

SQL Server

SQL Server has no MS SQL Native Client for Android or iOS, therefore a connection to SQL Server can only be established directly via TCP/IP by setting the `Provider` property to `prDirect`.

```
TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "SQL Server";
    Connection->Server = "server";
    Connection->Username = "user_name";
    Connection->Password = "password";
    Connection->Database = "database_name";
    Connection->SpecificOptions->Values["Provider"] = "prDirect";
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}
```

SQLite

If you don't deploy a database with your application, set the `ForceCreateDatabase` property to `True` to create a database file automatically when the user first launches your application.

```

TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "SQLite";
    Connection->SpecificOptions->Values["ForceCreateDatabase"] = "True";
    Connection->Database = System::Sysutils::IncludeTrailingPathDelimiter(
        System::Ioutils::TPath::GetDocumentsPath()) + "db.sqlite3";
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}

```

Oracle

Oracle has no client for Android or iOS, therefore a connection to an Oracle server can only be established directly via TCP/IP by setting the `Direct` property to `True`. To establish a connection to Oracle from Android or iOS, assign your host, port, and service name or system identifier to the `Server` property.

To connect using the service name, the format is as follows:

```

Server = "Host:Port:sn/ServiceName";
Server = "Host:Port:sn=ServiceName"; (deprecated format)

```

To connect using the SID, the format is as follows:

```

Server = "Host:Port:SID";
Server = "Host:Port:sid=SID"; (deprecated format)

```

If the port number is followed by a colon, and the service name prefix (`sn=`) or the SID prefix (`sid=`) is not defined, then by default, the connection will be established using SID. In majority of Oracle servers, the service name is the same as the SID. Consult the Oracle documentation for more information.

```

TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "Oracle";
    Connection->SpecificOptions->Values["Direct"] = "True";
    Connection->Server = "server:1521:orcl";
    Connection->Username = "user_name";
    Connection->Password = "password";
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}

```

MySQL

MySQL has no client for Android or iOS, therefore a connection to a MySQL server can only be established directly via TCP/IP by setting the `Direct` property to `True`.

```
TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "MySQL";
    Connection->SpecificOptions->Values["Direct"] = "True";
    Connection->Server = "server";
    Connection->Port = 3306;
    Connection->Username = "user_name";
    Connection->Password = "password";
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}
```

PostgreSQL

UniDAC supports only a direct connection to PostgreSQL, therefore there's no property that instructs the client on how to connect to the server.

```
TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "PostgreSQL";
    Connection->Server = "server";
    Connection->Port = 5432;
    Connection->Database = "database_name";
    Connection->SpecificOptions->Values["Schema"] = "schema_name";
    Connection->Username = "user_name";
    Connection->Password = "password";
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}
```

InterBase ToGo

You can connect to a local or remote InterBase ToGo database from iOS and Android devices. To connect to a local database, set the path to the database on the device. If you need to establish a connection to a remote server, specify the server address and database name.

Local

```
TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "InterBase";
    Connection->Database = System::Sysutils::IncludeTrailingPathDelimiter(
        System::Ioutils::TPath::GetDocumentsPath()) + "db.gdb";
    Connection->Username = "user_name";
    Connection->Password = "password";
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}
```

Remote

```
TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "InterBase";
    Connection->Server = "server";
    Connection->Database = "C:\db.gdb";
    Connection->Username = "user_name";
    Connection->Password = "password";
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}
```

Amazon Redshift

UniDAC supports only a direct connection to Redshift, therefore there's no property that instructs the client on how to connect to the server.

```
TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "Redshift";
    Connection->Server = "server";
    Connection->Username = "user_name";
    Connection->Password = "password";
    Connection->Database = "database_name";
    Connection->Port = 5439;
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}
```

xBase

xBase databases don't use the client-server model, therefore a connection to an xBase database can only be established directly via TCP/IP by setting the `Direct` property to `True`. To connect to an xBase database, set the path to the database and its format.

```
TUniConnection * Connection;
Connection = new TUniConnection(Form1);
try {
    Connection->ProviderName = "DBF";
    Connection->Database = "folder_name";
    Connection->SpecificOptions->Values["DBFFormat"] = "dfVisualFoxPro";
    Connection->SpecificOptions->Values["Direct"] = "True";
    Connection->Connect();
    ShowMessage("Connected successfully");
}
__finally {
    Connection->Free;
}
```

Deployment

The deployment path is different on Android and iOS. If you want to deploy your application to both platforms, make sure that the deployment paths are specified correctly for both Android and iOS. Despite having the same name, the providers for Android and iOS are different and located in their respective folders. Remember to replace the default value (".") of `Remote Path` with one of the values below.

C++ Builder Function	Deployment Path	Destination on Device
<code>TPath::GetDocumentsPath</code>	<code>.\assets \internal</code>	<code>/data/data/ com.embarca dero.MyProje cts/files</code>
<code>TPath::GetSharedDocumentsPath</code>	<code>.\assets</code>	<code>/mnt/sdcard/ Android/data/ com.embarca dero.MyProje cts/files</code>

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5 Provider-Specific Notes

This section covers provider-specific options and requirements, compatibility, and deployment aspects of data access providers in UniDAC.

Database Providers

- [UniDAC and Adaptive Server Enterprise](#)
- [UniDAC and Advantage Database Server](#)
- [UniDAC and Amazon Redshift](#)
- [UniDAC and DB2](#)
- [UniDAC and DBF](#)
- [UniDAC and InterBase](#)
- [UniDAC and Microsoft Access](#)
- [UniDAC and Microsoft Excel](#)
- [UniDAC and MongoDB](#)
- [UniDAC and MySQL](#)
- [UniDAC and NexusDB](#)
- [UniDAC and PostgreSQL](#)
- [UniDAC and ODBC](#)
- [UniDAC and Oracle](#)
- [UniDAC and SQLite](#)
- [UniDAC and SQL Server](#)

Cloud Providers

- [UniDAC and BigCommerce](#)
- [UniDAC and Google BigQuery](#)
- [UniDAC and Dynamics 365](#)
- [UniDAC and FreshBooks](#)
- [UniDAC and HubSpot](#)

- [UniDAC and Magento](#)
- [UniDAC and Mailchimp](#)
- [UniDAC and NetSuite](#)
- [UniDAC and QuickBooks](#)
- [UniDAC and Salesforce](#)
- [UniDAC and Salesforce MC](#)
- [UniDAC and SugarCRM](#)
- [UniDAC and Zoho CRM](#)
- [Database Specific Aspects of 64-bit Development](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1 Database Providers

5.1.1 UniDAC and Adaptive Server Enterprise

This article provides a brief overview of the SAP Sybase ASE data access provider for UniDAC used to establish a connection to ASE databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [ASE-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

- [Data Type Mapping](#)

Overview

ASE provider is based on the ODBC provider. It uses SAP Sybase ASE ODBC driver to work with database. Main features of SAP Sybase ASE data access provider are:

- High performance
- Easy deployment

The full list of SAP Sybase ASE provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the SAP Sybase ASE provider. Express Edition of UniDAC does not include the SAP Sybase ASE provider.

Compatibility

To learn about ASE database server compatibility, refer to the [Compatibility](#) section.

Requirements

Applications that use the SAP Sybase ASE provider require the following components to be installed on the client computer:

- ODBC (in the current versions of Microsoft Windows, since Windows 2000, ODBC is already included as a standard package);
- Adaptive Server Enterprise client software including ODBC driver.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

ASE-specific options

TUniConnection

Option name	Description
AnsiNull	This option serves primarily for Transact-SQL (Adaptive Server Enterprise) compatibility. AnsiNull affects the results of comparison predicates with NULL constants, and also affects warnings issued for grouped queries over NULL values.
ApplicationName	The name of a client application. The default value is the name of the executable file of your application.
Charset	The character set that will be used to transfer character data between the client and the server.
ClientHostName	The hostname of the client machine.
ColumnWiseBinding	Enables Column-Wise Binding . The default value is False. Note: Row-Wise Binding is enabled by default. However, some ODBC drivers don't support this mode. In this case, set the ColumnWiseBinding option to True.
ConnectionTimeout	The time to wait for a connection to open before raising an exception. The default value is 15.
DetectFieldsOnPrepare	Detects fields when Prepare is executed. The default value is True. Note: This functionality is not supported in some ODBC drivers.
Direct	If set to True, connection is performed directly over TCP/IP, and does not require SAP Sybase ASE software on the client side. Otherwise, provider connects through ODBC.
EncryptPassword	Specifies whether the password will be transmitted in encrypted format. epDisable The default value. Use plain text password. epRequire Use encrypted password. If it is not supported, return an error message. epPrefer Use encrypted password. If it is not supported, use plain text password. Note: If the server is configured to require clients to use an encrypted password, entering a plain text password will cause login to fail.
IPVersion	Use the IPVersion property to specify Internet Protocol Version. Supported values: ivIPBoth

	<p>Specifies that either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) will be used.</p> <p>ivIPv4 (default) Specifies that Internet Protocol Version 4 (IPv4) will be used.</p> <p>ivIPv6 Specifies that Internet Protocol Version 6 (IPv6) will be used.</p> <p>Note: When the IPVersion property is set to ivIPBoth, a connection attempt will be made via IPv6 if it is enabled on the operating system. If the connection attempt fails, a new connection attempt will be made via IPv4.</p>
MultipleConnections	<p>Enables or disables the creation of additional connections to support concurrent sessions, commands and rowset objects.</p>
PrepareMethod	<p>Use the option to specify whether stored procedures are created on the server for calls to SQLPrepare.</p> <p>Supported values:</p> <p>pmNone Stored procedures are created for every call to SQLPrepare, which may decrease performance when processing statements that do not contain parameters.</p> <p>pmPartial (default) Stored procedures are created only if the statement contains parameters. Otherwise, the statement is cached and executed directly at SQLExecute time.</p> <p>pmFull Stored procedures are never created. Any syntax or similar errors are reported at the time of SQLExecute.</p> <p>pmFullatPrepare Stored procedures are never created. Any syntax or similar errors are returned at the time of SQLPrepare instead of SQLExecute.</p>
SelectMethod	<p>Specifies whether cursors are to be used by the driver. smDirect indicates do not use cursors and smCursor indicates use cursors.</p> <p>The default value is <i>smDirect</i></p>
QuotedIdentifier	<p>To avoid conflicts in procedures and queries that contain reserved words, you should use the QuotedIdentifier option. The QuotedIdentifier option tells Adaptive Server to consider any character string enclosed in double quotes as an identifier. If this option is disabled (by default), ASE considers everything inside</p>

	the double quotes as a simple string.
TextSize	The maximum size of binary or text data in bytes that will be sent to or received from Adaptive Server, for example, TextSize=64000 sets this limit to 64K bytes. The default value is 0. Note: This option has no effect in the Direct mode because values are never truncated.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about returned fields and tables they belong to. The default value is True.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.

TUniScript

The TUniDump component has no ASE-specific options.

TUniLoader

The TUniLoader component has no ASE-specific options.

TUniDump

The TUniDump component has no ASE-specific options.

Data Type Mapping

The following table lists the constants for mapping SAP ASE data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
aseChar	Maps char to Delphi data types.
aseNChar	Maps nchar to Delphi data types.
aseUniChar	Maps unichar to Delphi data types.
aseNVarChar	Maps nvarchar to Delphi data types.
aseVarChar	Maps varchar to Delphi data types.
aseUniVarChar	Maps univarchar to Delphi data types.
aseTinyInt	Maps tinyint to Delphi data types.
aseSmallInt	Maps smallint to Delphi data types.
aseUSmallInt	Maps usmallint to Delphi data types.
aseInteger	Maps integer to Delphi data types.
aseUInteger	Maps uninteger to Delphi data types.
aseBigint	Maps bigint to Delphi data types.
aseUBigint	Maps ubigint to Delphi data types.
aseDecimal	Maps decimal to Delphi data types.
aseFloat	Maps float to Delphi data types.
aseDouble	Maps double to Delphi data types.
aseReal	Maps real to Delphi data types.
aseNumeric	Maps numeric to Delphi data types.
aseDate	Maps date to Delphi data types.
aseTime	Maps time to Delphi data types.
aseDateTime	Maps datetime to Delphi data types.
aseBit	Maps bit to Delphi data types.
aseBinary	Maps binary to Delphi data types.
aseVarBinary	Maps varbinary to Delphi data types.
aseImage	Maps image to Delphi data types.
aseText	Maps text to Delphi data types.
aseUniText	Maps unitext to Delphi data types.
aseXml	Maps xml to Delphi data types.
aseSmallmoney	Maps smallmoney to Delphi data types.
aseMoney	Maps money to Delphi data types.
aseSmalldatetime	Maps smalldatetime to Delphi data types.
aseTimestamp	Maps timestamp to Delphi data types.

Reserved.

5.1.2 UniDAC and Advantage Database Server

This article provides a brief overview of the Advantage data access provider for UniDAC used to establish a connection to Advantage from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Advantage-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [Data Type Mapping](#)

Overview

Advantage provider is based on the ODBC provider. It uses Advantage ODBC driver to work with database. Main features of Advantage data access provider are:

- High performance
- Easy deployment

The full list of Advantage provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Advantage provider. Express Edition of UniDAC does not include the Advantage provider.

Compatibility

To learn the supported versions of Advantage Database Server, refer to the [Compatibility](#) section.

Requirements

Applications that use the Advantage provider require the following components to be installed on the client computer:

- ODBC (in the current versions of Microsoft Windows, since Windows 2000, ODBC is already included as a standard package);
- Advantage ODBC driver.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Advantage-specific options

TUniConnection

Option name	Description
DefaultType	<p>Specifies the type of database files to be used.</p> <p>Supported values:</p> <p>dtAdvantage (default) Specifies that proprietary ADT tables with ADI index and ADM memo file formats will be used.</p> <p>dtFoxPro Specifies that FoxPro-compatible DBF tables with CDX index and FPT memo file formats will be used.</p> <p>dtVisualFoxPro Specifies that Visual FoxPro-compatible DBF tables with CDX index and FPT memo file formats will be used.</p> <p>dtClipper</p>

	Specifies that CA-Clipper-compatible DBF tables with NTX index and DBT memo fields will be used.
ColumnWiseBinding	<p>If set to True, the option enables Column-Wise Binding mode.</p> <p>The default value is False.</p> <p>Note: Row-Wise Binding mode is enabled by default. However, some ODBC drivers don't support this mode. In such case, set the ColumnWiseBinding option to True.</p>
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
ServerTypes	<p>Specifies the Advantage server types, to which connections should be attempted. Valid values include ADS, ALS, and AIS. ADL - Remote , ALS - local, and AIS - Internet Servers.</p> <p>These values can be logically OR'ed together with the "," in order to choose multiple server types. If multiple types are specified and multiple server types are available, the order of precedence is ADS first, AIS second, and ALS last.</p>
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about returned fields and tables they belong to. The default value is True.
FetchAll	<p>If True, all records of a query are requested from database server when the dataset is being opened.</p> <p>If False, records are retrieved when a data-aware component or a program requests it. The default value is False.</p>

TUniScript

The TUniDump component has no Advantage-specific options.

TUniLoader

The TUniLoader component has no Advantage-specific options.

TUniDump

The TUniDump component has no Advantage-specific options.

Data Type Mapping

The following table lists the constants for mapping Advantage Database Server data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
adsCharacter	Maps to Delphi data types.
adsNChar	Maps NChar to Delphi data types.
adsVarChar	Maps Varchar to Delphi data types.
adsNVarChar	Maps NVarChar to Delphi data types.
adsShortInteger	Maps Short to Delphi data types.
adsInteger	Maps Integer to Delphi data types.
adsNumeric	Maps Numeric to Delphi data types.
adsDouble	Maps Double to Delphi data types.
adsMoney	Maps Money to Delphi data types.
adsDate	Maps Date to Delphi data types.
adsTime	Maps Time to Delphi data types.
adsTimeStamp	Maps TimeStamp to Delphi data types.
adsLogical	Maps Logical to Delphi data types.
adsBinary	Maps Binary to Delphi data types.
adsVarBinary	Maps VarBinary to Delphi data types.
adsImage	Maps Image to Delphi data types.
adsMemo	Maps Memo to Delphi data types.
adsNMEMO	Maps NMEMO to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.3 UniDAC and Amazon Redshift

This article provides a brief overview of the Amazon Redshift data access provider for UniDAC used to establish a connection to Amazon Redshift from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Requirements](#)
- [Deployment](#)
- [Amazon Redshift-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript, TUniDump, TUniLoader](#)
- [Data Type Mapping](#)

Overview

Main features of Amazon Redshift data access provider are:

- Direct access to Amazon Redshift without additional client libraries or tools.
- High performance
- Easy deployment

The full list of provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Amazon Redshift provider.

Requirements

The Amazon Redshift provider and the PostgreSQL provider are included in one package (pgproviderXX.bpl), therefore, they are installed together.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more

information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

For more information about deployment of UniDAC-based applications, please, refer to the common [Deployment topic](#).

Amazon Redshift-specific options

Though UniDAC is components that provide unified interface to work with different database servers, it also lets you tune behaviour for each server individually. For thin setup of a certain database server, UniDAC provides server-specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a string list. Therefore you can use the following syntax to assign an option value:

```
UniConnection.SpecificOptions.Values['ConnectionTimeout'] := '15';
```

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
ApplicationName	The name of a client application. The default value is the name of the executable file of your application.
Charset	Specifies the character set that will be used to transfer character data between the client and the server.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
HttpPassword	Use the HttpPassword option to specify the password for HTTP authorization.
HttpTrustServerCertificate	This option specifies whether or not the driver should trust the server certificate when connecting to the server. The default value is False—the driver won't trust the server certificate and will verify validity of the server certificate instead. If set to True, the driver will trust the server certificate.
HttpUrl	Use the HttpUrl option to specify the URL of the PHP tunneling script.
HttpUsername	Use the HttpUsername option to specify the username for HTTP authorization.
IPVersion	Use the IPVersion property to specify Internet Protocol Version.

	<p>Supported values:</p> <p><i>ivIPBoth</i> Specifies that either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) will be used.</p> <p><i>ivIPv4 (default)</i> Specifies that Internet Protocol Version 4 (IPv4) will be used.</p> <p><i>ivIPv6</i> Specifies that Internet Protocol Version 6 (IPv6) will be used. Note: When the TIPVersion property is set to <i>ivIPBoth</i>, a connection attempt will be made via IPv6 if it is enabled on the operating system. If the connection attempt fails, a new connection attempt will be made via IPv4.</p>
MessagesCharset	Specifies the character set that will be used to transfer error messages from the server to the client.
MultipleConnections	Enables or disables the creation of an additional internal connection for TUniQuery, when necessary. The default value is True.
ProtocolVersion	<p>Specifies protocol version to be used when several versions are available.</p> <p>Supported values:</p> <p><i>pv20</i> Set ProtocolVersion to <i>pv20</i> to enforce protocol version 2.0.</p> <p><i>pv30 (default)</i> Set ProtocolVersion to <i>pv30</i> to enforce protocol version 3.0.</p> <p><i>pvAuto</i> Set ProtocolVersion to <i>pvAuto</i> to automatically select between protocol versions depending on the specific query for the best possible performance.</p>
ProxyHostname	Use the ProxyHostname option to specify the host name or IP address to connect to the proxy server.
ProxyPassword	Use the ProxyPassword option to specify the password for the proxy server.
ProxyPort	Use the ProxyPort option to specify the port for a TCP/IP connection with the proxy server.
ProxyUsername	Use the ProxyUsername option to specify the username for the proxy server.
Schema	Use the Schema property to set the search path for the connection to the specified schema. The setting offers a convenient way to perform operations on objects in a schema other than that of the current user without having to qualify the objects with the schema name.
SSLCACert	The pathname to the certificate authority file.

SSLCert	The pathname to the certificate file.
SSLChipherList	The list of allowable ciphers to use for SSL encryption.
SSLKey	The pathname to the key file.
SSLMode	This option determines whether or with what priority an SSL connection will be negotiated with the server.
UseHttp	The UseHttp option enables the use of HTTP tunneling to connect to the server. The default value is False.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.
UuidWithBraces	Use the UuidWithBraces option to specify whether the values of UUID fields are returned with braces. The default value is True.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
UnpreparedExecute	If 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.
UseParamTypes	Set this 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.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
AutoDeleteBlob	If True (the default value), the BLOBs are deleted from database automatically when a record that holds these BLOBs' OIDs is deleted from dataset.
CacheBlobs	If True (the default value), then local memory buffer is allocated to hold a copy of the BLOB content.
CommandTimeout	The time to wait for a statement to execute.
CursorWithHold	When this option is False (default), an active transaction is required to open a query in FetchAll=False mode. If there is no active transaction, UniDAC opens additional internal connection and starts transaction on this connection. When this option is True, UniDAC 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.
DeferredBlobRead	If True, all BLOB values are fetched only when they are explicitly requested. Otherwise entire record set with any BLOB values is returned when dataset is opened. Whether BLOB values are cached locally to be reused later is controlled by the CacheLobs option.
ExtendedFieldsInfo	If True, an additional query is performed to get information about returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is True.
KeySequence	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 the database.
OIDAsInt	If True, OID fields are mapped on TIntegerField. If False, values of OID fields are treated as large objects' OID, and these fields are mapped on TBlobField.
SequenceMode	Set the SequenceMode property to specify which method is used internally to generate sequenced field. The following values are allowed for this property: smInsert New record is inserted into the dataset with the first key field populated with a sequenced value. Application may modify this field before posting the record to the database. smPost Database server populates key field with a sequenced value when application posts the record to the database. Any value put into the key field before post will be overwritten.
UnknownAsString	If True, all Amazon Redshift data types that are fetched as text, and don't have limited field size, are mapped on TStringField with default size 8192. If False, such types are mapped on TMemoField. The TEXT data type is always mapped on TMemoField regardless of this option.
UnpreparedExecute	If 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.
UseParamTypes	Set this 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.
--	---

TUniScript, TUniDump, TUniLoader

The TUniScript, TUniDump, TUniLoader components have no Amazon Redshift-specific options.

Data Type Mapping

The following table lists the constants for mapping Amazon Redshift data types to Delphi data types. See the [Data Type Mapping](#) tutorial for more information.

Constant	Description
pgBigInt	Maps bigint to Delphi data types.
pgBigSerial	Maps bigserial to Delphi data types.
pgBit	Maps bit to Delphi data types.
pgBitVarying	Maps bit varying to Delphi data types.
pgBoolean	Maps boolean to Delphi data types.
pgBytea	Maps bytea to Delphi data types.
pgCharacter	Maps character to Delphi data types.
pgCharacterVarying	Maps character varying to Delphi data types.
pgDate	Maps date to Delphi data types.
pgDoublePrecision	Maps double precision to Delphi data types.
pgInteger	Maps integer to Delphi data types.
pgMoney	Maps money to Delphi data types.
pgNumeric	Maps numeric to Delphi data types.
pgReal	Maps real to Delphi data types.
pgSerial	Maps serial to Delphi data types.
pgSmallint	Maps smallint to Delphi data types.
pgText	Maps text to Delphi data types.
pgTime	Maps time to Delphi data types.
pgTimeStamp	Maps timestamp to Delphi data types.
pgTimeStampWithTimeZone	Maps timestamp with time zone to Delphi data types.
pgTimeWithTimeZone	Maps time with time zone to Delphi data types.
pgUUID	Maps uuid to Delphi data types.

Reserved.

5.1.4 UniDAC and DB2

This article provides a brief overview of the DB2 data access provider for UniDAC used to establish a connection to DB2 databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [DB2-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [Data Type Mapping](#)

Overview

DB2 provider is based on the ODBC provider. It uses DB2 ODBC driver to work with a database. Main features of the DB2 data access provider are:

- High performance
- Easy deployment

The full list of the DB2 provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the DB2 provider. Express Edition of UniDAC does not include the DB2 provider.

Compatibility

To learn about DB2 database server compatibility, refer to the [Compatibility](#) section.

Requirements

Applications that use the DB2 provider require the following components to be installed on the client computer:

- ODBC (in the current versions of Microsoft Windows, since Windows 2000, ODBC is already included as a standard package);
- DB2 client software including the ODBC driver.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

DB2-specific options

TUniConnection

Option name	Description
ColumnWiseBinding	If set to True, the option enables Column-Wise Binding mode. The default value is False. Note: Row-Wise Binding mode is enabled by default. However, some ODBC drivers don't support this mode. In such case, set the ColumnWiseBinding option to True.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
FunctionPath	Use the FunctionPath property to change the current function path of the connection to the specified value. You can specify several names separated by comma. This option can be used to call stored procedures from a schema other than that of the current user without having to qualify the objects with the schema name.
Schema	Use the Schema property to change the current schema of the connection to the specified schema. This setting offers a

	convenient way to perform operations on objects in a schema other than that of the current user without having to qualify the objects with the schema name.
Security	Gets or sets the security type being used. If set to <code>smSSL</code> , SSL is enabled. The default value is <code>smDisable</code> .
SSLClientKeyStash	Gets or sets the name of the SSL stash (.sth) file used for an SSL connection. The option value is the name of the SSL stash file used for an SSL connection. Note: When Security is set to SSL, you must specify the <code>SSLClientKeystoredb</code> option along with this option. The SSL stash file stores an encrypted password for the key database file.
SSLClientkeystoredb	Gets or sets the name of the SSL key database file used for an SSL connection. The option value is the name of the SSL key database file used for an SSL connection. Note: When Security is set to SSL, you must specify the <code>SSLClientKeystash</code> option along with this option.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to <code>True</code> , all character data is stored as <code>WideStrings</code> , and <code>TStringField</code> is replaced by <code>TWideStringField</code> .

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If <code>True</code> , an additional query is performed to get information about returned fields and tables they belong to. The default value is <code>True</code> .
KeySequence	Use the <code>KeySequence</code> property to specify the name of the sequence that will be used to fill in a key field after a new record is inserted or posted to the database.

FetchAll	<p>If True, all records of a query are requested from database server when the dataset is being opened.</p> <p>If False, records are retrieved when a data-aware component or a program requests it. The default value is False.</p>
SequenceMode	<p>Set the SequenceMode property to specify which method is used internally to generate sequenced field.</p> <p>The following values are allowed for this property:</p> <p>smInsert New record is inserted into the dataset with the first key field populated with a sequenced value. Application may modify this field before posting the record to the database.</p> <p>smPost Database server populates the key field with a sequenced value when application posts the record to the database. Any value put into the key field before post will be overwritten.</p>

TUniScript

The TUniScript component has no DB2-specific options.

TUniLoader

The TUniLoader component has no DB2-specific options.

TUniDump

The TUniDump component has no DB2-specific options.

Data Type Mapping

The following table lists the constants for mapping DB2 data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
db2Char	Maps CHARACTER to Delphi data types.
db2VarChar	Maps VARCHAR to Delphi data types.
db2SmallInt	Maps SMALLINT to Delphi data types.
db2Integer	Maps INTEGER to Delphi data types.
db2BigInt	Maps BIGINT to Delphi data types.

db2Decimal	Maps DECIMAL to Delphi data types.
db2Float	Maps FLOAT to Delphi data types.
db2Double	Maps DOUBLE to Delphi data types.
db2Real	Maps REAL to Delphi data types.
db2Numeric	Maps NUMERIC to Delphi data types.
db2Date	Maps DATE to Delphi data types.
db2Time	Maps TIME to Delphi data types.
db2TimeStamp	Maps TIMESTAMP to Delphi data types.
db2Binary	Maps BINARY to Delphi data types.
db2VarBinary	Maps VARBINARY to Delphi data types.
db2Blob	Maps BLOB to Delphi data types.
db2Clob	Maps CLOB to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.5 UniDAC and DBF

This article provides a brief overview of the DBF data access provider for UniDAC used to establish a connection to DBF databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Requirements](#)
- [Deployment](#)
- [DBF-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [Data Type Mapping](#)

Overview

Main features of the DBF data access provider are:

- Direct access to the database without using Microsoft dBase ODBC driver
- High performance
- Easy deployment

The full list of the DBF provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the DBF provider. Express Edition of UniDAC does not include the DBF provider.

Compatibility

To learn the DBF formats supported by the provider, refer to the [Compatibility](#) section.

Requirements

If your application is working in the Direct mode, it is not required to install any additional software on the client. For application that has Direct mode disabled, it is required to install the following components on the client computer:

- ODBC (in the current versions of Microsoft Windows, since Windows 2000, ODBC is already included as a standard package);
- Microsoft dBase ODBC driver

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

DBF-specific options

TUniConnection

Option name	Description
AllFieldsAsNullable	Used to open DBF tables with incorrect data (tables that have

	records with null values in non-nullable columns). The default value is False.
CodePage	Specifies a code page when working with a database. Available values: dpDefault, dpUnitedStatesOEM, dpGreekDOS, dpWesternEuropeanDOS, dpTurkishDOS, dpCentralEuropeanDOS, dpPortugueseDOS, dpIcelandicDOS, dpFrenchCanadianDOS, dpNordicDOS, dpCyrillicDOS, dpThai, dpJapanese, dpChineseSimplified, dpChineseTraditional, dpKorean, dpCentralEuropeanANSI, dpCyrillicANSI, dpWesternEuropeanANSI, dpGreekANSI, dpTurkishANSI, dpHebrewANSI, dpArabicANSI, dpBalticANSI. Default value is dpDefault.
CollatingSequence	Specifies the collation sequence. Available values: ASCII and International. The default value is ASCII.
ColumnWiseBinding	If set to True, the option enables Column-Wise Binding mode. The default value is False. Note: Row-Wise Binding mode is enabled by default. However, some ODBC drivers don't support this mode. In such case, set the ColumnWiseBinding option to True.
Connect Mode	Used to specify how connections access DBF table files. Exclusive Only one connection can access the table file at a time. The active connection holds read/write lock on the file to prevent other connections from reading or writing to it. Shared The default value. Multiple concurrent connections can read/write to the same table. The active connection holds read/write lock on the table file, but releases the lock once it has finished reading or writing the data. Unsafe Multiple concurrent connections can read and write to the table file. This mode should be used with caution because it allows multiple connections to modify the file simultaneously. Since DBF databases do not support transactions, an attempt to change the same file simultaneously by multiple connections can cause data corruption in the table file.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
DBFFormat	The default database format that will be used when creating new tables and working with indexes. Available values: dfAuto,

	dfdBaseIII, dfdBaseIV, dfdBaseV, dfdBaseVII, dfFoxPro2, dfVisualFoxPro, dfHiPerSix, dfCodebase and dfClipper. Default value is dfAuto. When using dfAuto, the format is detected by .DBF file header. For any other values, .DBF file header will be ignored. The format from the DBFFormat value will be forced used for all .DBF files in the folder.
Direct	If set to True, connection to the database is performed directly, and does not require any additional software on the client side. Otherwise, the provider connects using Microsoft dBase ODBC driver. Default value is False.
IdentifierCase	The IdentifierCase property allows you to set the case for field names. Supported values: icOriginal Field names are returned without changing the case. icLower Field names are returned in the lowercase. icUpper Field names are returned in the uppercase.
IgnoreBrokenTables	If set to True, corrupted tables in the directory will be ignored, and an exception won't be raised. The default value is False.
IgnoreDataErrors	If set to True, corrupted data errors will be ignored when opening a DBF table and an exception will not be raised. The default value is False.
IgnoreIndexErrors	If set to True, errors in database indexes will be ignored when opening a DBF table, and an exception won't be raised. The default value is False.
IgnoreMetadataErrors	If set to True, metadata errors will be ignored when opening a DBF table and an exception will not be raised. The default value is False.
IndexOnReading	Specifies a mechanism of indexes when fetching tables data. Available values: ikNative and ikLocal. When set to ikNative, UniDAC will use standard DBF indexes. We recommend using it when executing SELECT SQL queries for one table with the WHERE clause. When set to ikLocal, UniDAC will use its internal data indexing mechanism. We recommend using it when the SELECT SQL query is executed for several tables (for example, JOIN) with the WHERE clause.
UseFileCodepage	Enables or disables the use of the file encoding when reading or writing textual data. When UseFileCodepage is False, the CodePage option has no effect, and the local encoding is used.

	The default value is True.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the database. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField. The default value is False.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about returned fields and tables they belong to. The default value is True.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.

TUniScript

The TUniDump component has no DBF-specific options.

TUniLoader

The TUniLoader component has no DBF-specific options.

TUniDump

The TUniDump component has no DBF-specific options.

Data Type Mapping

The following table lists the constants for mapping DBF data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
dbfChar	Maps CHAR to Delphi data types.

dbfVarChar	Maps VARCHAR to Delphi data types.
dbfMemo	Maps MEMO to Delphi data types.
dbfAutoincrement	Maps AUTOINCREMENT to Delphi data types.
dbfLogical	Maps LOGICAL to Delphi data types.
dbfInteger	Maps INTEGER to Delphi data types.
dbfFloat	Maps FLOAT to Delphi data types.
dbfDouble	Maps DOUBLE to Delphi data types.
dbfCurrency	Maps CURRENCY to Delphi data types.
dbfNumeric	Maps NUMERIC to Delphi data types.
dbfDate	Maps DATE to Delphi data types.
dbfTime	Maps TIME to Delphi data types.
dbfVarBinary	Maps VARBINARY to Delphi data types.
dbfBlob	Maps BLOB to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.6 UniDAC and InterBase/Firebird

5.1.6.1 InterBase/Firebird Provider

This article provides a brief overview of the InterBase data access provider for UniDAC used to establish a connection to InterBase/Firebird from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [InterBase-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

- [InterBase-specific notes](#)
 - [Parallel transactions management](#)
- [Data Type Mapping](#)

Overview

InterBase data access provider is based on the InterBase Data Access Components ([IBDAC](#)) library, which is one of the best known Delphi data access solutions for InterBase and Firebird. The main features of InterBase data access provider are:

- High performance
- Easy deployment
- Comprehensive support for the latest versions of InterBase/Firebird server

The full list of InterBase provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the InterBase provider. For Express Edition of UniDAC, the InterBase provider can be installed with IBDAC.

Compatibility

To learn the supported versions of InterBase and Firebird, refer to the [Compatibility](#) section.

Requirements

Applications that use the InterBase provider require InterBase/Firebird client software only. The InterBase provider dynamically loads InterBase client DLL (GDS32.DLL or FBClient.dll for Firebird) available on user systems. To locate DLL you can set the ClientLibrary specific option of TUniConnection with the path to the client library. By default the InterBase provider searches a client library in directories specified in the PATH environment variable.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

InterBase-specific options

Though UniDAC is components that provide unified interface to work with different database servers, it also lets you tune behaviour for each server individually. For thin setup of a certain database server, UniDAC provides server-specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a string list. Therefore you can use the following syntax to assign an option value:

```
TUniConnection.SpecificOptions.Values['CharLength'] := '1';
```

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
CharLength	Specifies the size in bytes of a single character. Set this option with the number in range [0..6] to reflect InterBase support for the national languages. Setting CharLength to zero will instruct TUniConnection to interrogate InterBase server for the actual character length. The default value is 1.
Charset	Sets character set that IBDAC uses to read and write character data.
ClientLibrary	Use the ClientLibrary option to set or get the client library location.
EnableMemos	If set to True, TMemoField and TWideMemoField will be created for BLOB subtype 1 fields. The default value is False.
ForceUnloadClientLibrary	Use the option to force unloading of the client library after the connection is closed. The default value is False.
IPVersion	Use the IPVersion property to specify Internet Protocol Version. Supported values: ivIPBoth (default) Specifies that either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) will be used. ivIPv4 Specifies that Internet Protocol Version 4 (IPv4) will be used. ivIPv6 Specifies that Internet Protocol Version 6 (IPv6) will be used.

	<p>Note: Internet Protocol Version support has been added in Firebird 3. To use the IPVersion option, your client library version must be version 3 or higher.</p> <p>When the TIPVersion property is set to ivIPBoth, a connection attempt will be made via IPv6 if it is enabled on the operating system. If the connection attempt fails, a new connection attempt will be made via IPv4.</p>
NoDBTriggers	Use the option to enable or disable all database triggers. By default, all triggers are enabled.
Params	The option allows specifying custom parameters of the transaction. Refer to the InterBase API guide for more information on the parameters. Custom parameters will be used only when the TUniTransaction.IsolationLevel property is set to ilCustom. Multiple parameters can be separated either with the CRLF or with the ";" character.
Protocol	Network protocol of connection with InterBase server. The default value is TCP.
Role	InterBase connection role.
SQLDialect	Use SQLDialect to set or return SQL Dialect used by InterBase client. The SQLDialect property cannot be set to a value greater than the database SQL dialect when the connection is active. If the connection is inactive, the SQLDialect option will be downgraded to match the database SQL dialect.
SSLClientCertFile	The name and location of the client certificate file. The file must be in the PEM format and contain both the client certificate and the private key.
SSLClientPassPhrase	The private key passphrase. You can use either this option or the SSLClientPassPhraseFile option.
SSLClientPassPhraseFile	The name and location of the text file containing the client private key passphrase. You can use either this option or the SSLClientPassPhrase option.
SSLServerPublicFile	The name and location of the CA certificate file in the PEM format.
SSLServerPublicPath	The location of the directory with the CA certificate files in the PEM format. Each file in the directory must contain only a single CA certificate and the files must be named by the hash of the subject name and extension of ".0". It is recommended that you use SSLServerPublicFile instead. If you specify both, SSLServerPublicFile will be used.
TrustedAuthentication	<p>Windows "Trusted User" security can be applied for authenticating Firebird users on a Windows host.</p> <p>When the option is set to True, the Firebird security database is ignored during establishing a connection, and only Windows authentication is used.</p> <p>The default value is False</p>

	More detailed information about this authentication mode is available at http://firebirdsql.org/rlsnotesh/rlsnotes210.html#mfb210-wintrusted .
UseSSL	Enables or disables SSL connections. The default value is False.
UseUnicode	Enables or disables Unicode support. Affects on the character data fetched from the server. When set to True all character data is stored as WideStrings, and TStringField is replaced with TWideStringField.
SimpleNumericMap	Used to create ftBCD fields. When it is set to "False" and EnableBCD to "True", fields like DECIMAL(14, 4) are mapped as ftBCD. The option default value is "True".
WireCompression	Enables or disables compression of data over the wire at global or individual database level. Use Params to pass this connection parameter, for example WireCompression=True. Disabled by default. Note: You should place zlib1.dll in the same location as fbclient.dll.

TUniSQL

Option name	Description
AutoCommit	Used to automatically commit each update, insert or delete statement by database server. When using the option it should be kept in mind that the AutoCommit property of TUniConnection has higher precedence over the same properties in components. When the AutoCommit property of a dataset is True and TUniConnection.AutoCommit is True, each update, insert or delete statement is automatically committed by database server. When TUniConnection.AutoCommit is False, automatic commit does not occur, regardless of the value of the AutoCommit option of the dataset.
DescribeParams	Specifies whether to query the Name, ParamType, DataType, Size, and TableName properties from the server when preparing a query. The default value is False.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
AutoCommit	Used to automatically commit each update, insert or delete statement by database server. When using the option it should be kept in mind that the AutoCommit property of TUniConnection has higher precedence over the same properties in components. When the AutoCommit property of a dataset is True and

	<p>TUniConnection.AutoCommit is True, each update, insert or delete statement is automatically committed by database server. When TUniConnection.AutoCommit is False, automatic commit does not occur, regardless of the value of the AutoCommit option of the dataset.</p>
AutoClose	<p>The cursor will be closed after fetching all rows. Allows to reduce the number of opened cursors on the server.</p>
BooleanDomainFields	<p>If the BooleanDomainFields property is set to True, TBooleanField objects are created for fields that have domain of the integer data type, and the domain name contains 'BOOLEAN'. The default value is True. Note: This option has no effect when SetDomainNames is set to False.</p>
CacheArrays	<p>If True, local memory buffer is allocated for a copy of the array. The default value is True. This option has no effect when DeferredArrayRead is set to False because all BLOBs are fetched to the dataset in that case.</p>
CacheBlobs	<p>If True (the default value), local memory buffer is allocated to hold a copy of the BLOB content. Note: The CacheBlobs option controls the way streamed BLOB objects are handled. If False, application can access streamed BLOB values on the server without caching BLOBs on the client - only the requested portions of data are fetched. Setting CacheBlobs to False may reduce network traffic since only the required data is fetched, and reduce memory consumption on the client because the returned record sets do not hold contents of BLOB fields. This feature is only available for streamed BLOBs when StreamedBlobs is set to True. This option has no effect if DeferredBlobRead is set to False because all BLOB values are fetched to the dataset in that case.</p>
ComplexArrayFields	<p>If the ComplexArrayFields property is set to False, any array field is stored as a single TIBCArrayField object. If the option and ObjectView are set to True, array items are stored hierarchically. If the option is set to True, but ObjectView is False, all array items are stored as sibling fields.</p>
DeferredArrayRead	<p>If True, all InterBase array values are fetched only when they are explicitly requested. Otherwise the entire record set with any array values is returned when dataset is opened. Whether array values are cached locally to be reused later or not is controlled by the CacheArrays option.</p>
DeferredBlobRead	<p>If True, all InterBase BLOB values are fetched only when they are explicitly requested. Otherwise the entire record set with any BLOB values is returned when dataset is opened. Whether BLOB values are cached locally to be reused later or not is controlled by</p>

	the CacheBlobs option.
DescribeParams	Specifies whether to query the Name, ParamType, DataType, Size, and TableName properties from the server when preparing a query. The default value is False.
ExtendedFieldsInfo	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. The default value is True.
FetchAll	If True, all records of the query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If True, then all non-BLOB fields are treated as being of string data type.
GeneratorMode	Set the GeneratorMode property to specify which method is used internally to generate sequenced field. The following values are allowed for this property: gmInsert New record is inserted into the dataset with the first key field populated with a sequenced value. Application may modify this field before posting the record to the database. gmPost Database server populates key field with a sequenced value when application posts the record to the database. Any value put into key field before post will be overwritten.
GeneratorStep	Use the GeneratorStep option to set the increment for increasing or decreasing current generator value when using automatic key field value generation feature. The default value is 1.
KeyGenerator	Use the KeyGenerator option to specify the name of a generator that will be used to fill in a key field after a new record is inserted or posted to the database. KeyGenerator is used only if the KeyFields property is assigned.
QueryRowsAffected	Use the option to increase the performance of update operations. The default value is True.
SetDomainNames	Use the option to retrieve the DOMAIN name for a field. The default value is False.
StreamedBlobs	If True, all BLOBs are handled and saved as streamed BLOBs. Otherwise, BLOBs are handled and saved as segmented BLOBs. Setting this option to True allows you to benefit from the CacheBlobs option.

TUniScript

Option name	Description
AutoDDL	Use the AutoDDL property to determine whether DDL statements must be executed in a separate transaction.

TUniTransaction

Option name	Description
IsolationLevel	<p>ilCustom The parameters of the transaction are set manually in the Params property.</p> <p>ilSnapshot ilRepeatableRead The default isolation level. Provides a stable, committed view of the database at the time the transaction starts. Other simultaneous transactions can UPDATE and INSERT rows, but this transaction cannot see these changes. For updated rows, this transaction sees versions of these rows as they existed at the start of the transaction. If this transaction attempts to update or delete rows changed by another transaction, an update conflict is reported.</p> <p>ilIsolated Provides a transaction read-only access to the tables it uses. Other simultaneous transactions may be able to select rows from these tables, but they can not insert, update, and delete rows from these tables.</p> <p>ilReadCommitted Enables the transaction to see all committed data in the database and to update rows updated and committed by other simultaneous transactions without causing lost update problems.</p> <p>ilReadUnCommitted Not supported.</p>
Params	<p>The option allows to specify custom parameters of the transaction. Refer to InterBase API Guide for more information on this parameters. Custom parameters will be used only when the TUniTransaction.IsolationLevel property is set to ilCustom. Multiple parameters can be separated either with the CRLF or with the ";" character.</p>

TUniLoader

Option name	Description
AutoCommit	Used to automatically commit each update, insert or delete statement by database server. When using the option it should be kept in mind that the AutoCommit property of TUniConnection has higher precedence over the same properties in components. When the AutoCommit property of a dataset is True and TUniConnection.AutoCommit is True, each update, insert or delete statement is automatically committed by database server. When TUniConnection.AutoCommit is False, automatic commit does not occur, regardless of the value of the AutoCommit option of the dataset.
InsertMode	Use the InsertMode option to specify the type of statement used for loading data to InterBase database. If the value is imInsert (default value), the INSERT INTO statement will be used. If set to imUpdateOrInsert, the UPDATE OR INSERT INTO statement will be used.
QuoteNames	Use the QuoteNames option to quote all database object names in automatically generated SQL statements, such as UPDATE statements. The default value is False.
RowsPerBatch	Use the RowsPerBatch option to specify the number of records that are sent to the server in a single operation. The default value is 50.

TUniDump

The TUniDump component has no InterBase-specific options.

InterBase-specific notes

This chapter describes several special cases of using InterBase data provider.

Parallel transactions management

InterBase and Firebird database servers support multiple parallel transactions within one connection. You can use this feature with UniDAC and InterBase provider. You should link the TUniTransaction component to a component you want to interact with the sever within a separate transaction. To link a TUniTransaction object to a component, for example to TUniQuery, assign the TUniTranaction object to the TUniQuery.Transaction property:

```
UniQuery1.Transaction := UniTransaction1;
```

The Transaction property persists in the following components: [TUniQuery](#) , [TUniTable](#) , [TUniStoredProc](#) , [TUniSQL](#) , [TUniScript](#) , [TUniMetaData](#) .

Data Type Mapping

The following table lists the constants for mapping InterBase/Firebird data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
ibcArray	Maps ARRAY to Delphi data types.
ibcBigint	Maps BIGINT to Delphi data types.
ibcBlob	Maps BLOB to Delphi data types.
ibcBoolean	Maps BOOLEAN to Delphi data types.
ibcChar	Maps CHAR to Delphi data types.
ibcCharBin	Maps BINARY to Delphi data types.
ibcDate	Maps DATE to Delphi data types.
ibcDecimal	Maps DECIMAL to Delphi data types.
ibcDouble	Maps DOUBLE PRECISION to Delphi data types.
ibcFloat	Maps FLOAT to Delphi data types.
ibcInteger	Maps INTEGER to Delphi data types.
ibcNumeric	Maps NUMERIC to Delphi data types.
ibcSmallint	Maps SMALLINT to Delphi data types.
ibcText	Maps TEXT to Delphi data types.
ibcTime	Maps TIME to Delphi data types.
ibcTimestamp	Maps TIMESTAMP to Delphi data types.
ibcVarchar	Maps VARCHAR to Delphi data types.
ibcVarcharBin	Maps VARCHAR BINARY to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.6.2 OTW Network Encryption

Encrypting Network Using Over-the-Wire (OTW)

The InterBase provider supports the Over-the-Wire (OTW) encryption feature of InterBase to encrypt data during the transmission process. InterBase OTW encryption uses SSL v3 and TLS v1 security protocols and supports AES and DES encryptions. Before setting up OTW encryption on the server and client side, you must obtain the necessary security certificates

from a certificate authority (CA). Both the client and server must have the X.509 files in the PEM format installed to use OTW encryption. After configuring the OTW parameters on the server, set up the client side in your UniDAC-based application. The OTW encryption parameters can be set up at runtime as follows:

Delphi

```
var
  UniConnection1: TUniConnection;
begin
  UniConnection1 := TUniConnection.Create(nil);
  try
    UniConnection1.ProviderName := 'InterBase';
    UniConnection1.Server := '127.0.0.1';
    UniConnection1.Database := 'database';
    UniConnection1.Username := 'username';
    UniConnection1.Password := 'password';
    UniConnection1.Port := 3050;
    UniConnection1.SpecificOptions.Values['ClientLibrary'] := 'gds32.dll';
    UniConnection1.LoginPrompt := False;
    // OTW encryption properties
    UniConnection1.SpecificOptions.Values['ClientCertFile'] := 'clientcert.p
    UniConnection1.SpecificOptions.Values['ClientPassPhrase'] := 'passphrase
    UniConnection1.SpecificOptions.Values['ServerPublicFile'] := 'cacert.pem
    UniConnection1.SpecificOptions.Values['UseSSL'] := 'True';
    UniConnection1.Open;
  finally
    UniConnection1.Free;
  end;
end;
```

C++ Builder

```
TUniConnection* UniConnection = new TUniConnection(NULL);
try {
  UniConnection->ProviderName = "InterBase";
  UniConnection1->Server = "127.0.0.1";
  UniConnection1->Database = "database";
  UniConnection1->Username = "username";
  UniConnection1->Password = "password";
  UniConnection1->Port = 3050;
  UniConnection1->SpecificOptions->Values["ClientLibrary"] = "gds32.dll";
  UniConnection1->LoginPrompt = False;
  // OTW encryption properties
  UniConnection1->SpecificOptions->Values["ClientCertFile"] = "clientcert.
  UniConnection1->SpecificOptions->Values["ClientPassPhrase"] = "passphas
  UniConnection1->SpecificOptions->Values["ServerPublicFile"] = "cacert.pe
  UniConnection1->SpecificOptions->Values["UseSSL"] = "True";
  UniConnection1->Open;
}
finally {
  UniConnection1->Free();
}
```

See Also

- [Encrypting Network Communication](#)
- [InterBase Provider Options](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.7 UniDAC and Microsoft Access

This article provides a brief overview of the Microsoft Access data access provider for UniDAC used to establish a connection to Access databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Access-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [Connecting in Direct mode](#)
- [Data Type Mapping](#)

Overview

Access provider is based on the ODBC provider. It uses Microsoft Access ODBC driver to work with a database. Main features of the Access data access provider are:

- High performance

- Easy deployment

The full list of the Access provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Access provider. Express Edition of UniDAC does not include the Access provider.

Compatibility

To learn the supported versions of Microsoft Access, refer to the [Compatibility](#) section.

Requirements

Applications that use the Access provider require Microsoft Data Access Components (MDAC) to be installed on the client computer. In the current versions of Microsoft Windows, since Windows 2000, MDAC is already included as a standard package.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Access-specific options

TUniConnection

Option name	Description
Charset	Specifies the encoding for storing textual data in the database. The option is used for working with older Access formats (e.g., Access 97) that do not support Unicode.
ColumnWiseBinding	If set to True, the option enables Column-Wise Binding mode. The default value is False. Note: Row-Wise Binding mode is enabled by default. However, some ODBC drivers don't support this mode. In such case, set the ColumnWiseBinding option to True.

ConnectionTimeout	The time to wait for a connection to open before raising an exception.
Direct	<p>Allows accessing Microsoft Access databases directly or through the Microsoft Access ODBC Driver.</p> <p>The following values are supported:</p> <p>False Establishes the connection to Microsoft Access databases directly via the ODBC Driver.</p> <p>True Allows accessing Microsoft Access databases without using the ODBC Driver.</p> <p>The default value is False.</p>
DriverVersion	<p>Use the DriverVersion property to specify the version of Microsoft Access Driver (*.mdb, *.acldb).</p> <p>The following values are supported:</p> <p>dvAuto The code first tests for the presence of *.acldb driver - if it is not found, *.mdb will be used.</p> <p>dvAcldb Specifies that *.acldb driver will be used.</p> <p>dvMdb Specifies that *.mdb driver will be used.</p> <p>The default value is dvAuto.</p>
ExclusiveLock	If True, a database will be opened in the Exclusive mode and can be accessed by only one user at a time. Performance is enhanced when running in the Exclusive mode.
ExtendedAnsiSQL	<p>If True, an extended SQL support is enabled.</p> <p>Two new data types are available in Jet 4.0 databases when the ExtendedAnsiSQL flag is turned on: SQL_DECIMAL and SQL_NUMERIC. The default precision and scale are 18 and 0, respectively. Data accessed via ODBC that is typed as SQL_DECIMAL or SQL_NUMERIC will be mapped to Microsoft Jet Decimal instead of Currency.</p> <p>When the ExtendedAnsiSQL flag is turned off, you cannot create tables with decimal or numeric types, and these types will not appear in SQLGetTypeInfo(). However, if the table contains the</p>

	new data types, they can be used with the correct data types.
ForceCreateDatabase	Is used to force TLiteConnection to create a new database before opening a connection, if the database does not exist.
ReadOnly	<p>Allows implementing a read-only multi-user mode to enable several users to read data from a Microsoft Access database simultaneously.</p> <p>The following values are supported:</p> <p>False Keeps a single-user mode.</p> <p>True Allows two or more users to access the database simultaneously, however only in the ReadOnly mode. The default value is False.</p>
SystemDatabase	The full path to the Microsoft Access system database to be used with the Microsoft Access database you want to access.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about returned fields and tables they belong to. The default value is True.
FetchAll	<p>If True, all records of a query are requested from database server when the dataset is being opened.</p> <p>If False, records are retrieved when a data-aware component or a program requests it. The default value is False.</p>

TUniScript

The TUniScript component has no Access-specific options.

TUniLoader

The TUniLoader component has no Access-specific options.

TUniDump

The TUniDump component has no Access-specific options.

Connecting in Direct mode

Direct connection enables users to connect to Microsoft Access databases directly, without the need for Microsoft Access to be installed on their machine.

Data Type Mapping

The following table lists the constants for mapping Access data types to Delphi data types.

See [Data Type Mapping](#) for more information.

Constant	Description
accText	Maps Short Text to Delphi data types.
accLongText	Maps Long Text to Delphi data types.
accByte	Maps Byte to Delphi data types.
accInteger	Maps Integer to Delphi data types.
accLong	Maps Long to Delphi data types.
accSingle	Maps Single to Delphi data types.
accDouble	Maps Double to Delphi data types.
accNumeric	Maps Numeric to Delphi data types.
accDateTime	Maps Date/Time to Delphi data types.
accBit	Maps Bit to Delphi data types.
accBinary	Maps Binary to Delphi data types.
accVarBinary	Maps VarBinary to Delphi data types.
accLongBinaryLong Binary	Maps to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.8 UniDAC and Microsoft Excel

This article provides a brief overview of the Microsoft Excel data access provider for UniDAC used to establish a connection to Excel workbooks from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Excel-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [Connecting in Direct mode](#)
- [Data Type Mapping](#)

Overview

Excel provider is based on the ODBC provider. It uses Microsoft Excel ODBC driver to work with a workbook. Main features of the Excel data access provider are as follows:

- High performance
- Easy deployment

The full list of the Excel provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Excel provider. Express Edition of UniDAC does not include the Excel provider.

Compatibility

To learn the supported versions of Microsoft Excel, refer to the [Compatibility](#) section.

Requirements

Applications that use the Excel provider require Microsoft Data Access Components (MDAC) to be installed on the client computer. In the current versions of Microsoft Windows, since Windows 2000, MDAC is already included as a standard package.

Deployment

When an application was built without runtime packages (Link with runtime packages is set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note, that UniDAC Trial requires deployment of additional BPL files regardless of the Link with the runtime packages.

Excel-specific options

TUniConnection

Option name	Description
Database	The full path to the workbook to connect to.
Default Cell Type	Specifies what string data types are used by the provider for literal columns. <ul style="list-style-type: none"> cdString - All literal columns will be returned as string. cdMemo - All literal columns will be returned as memo. The default value is cdString.
Describe Amount	Specifies the number of sample worksheet rows used to create a list of columns in the corresponding table. The default value is 25.
Exclusive Lock	If set to True, this parameter enforces a single-user access mode for the workbook. Even with ReadOnly set to True, only one user is able to access the workbook at a time. The default value is False.
Header Row	Indicates whether the data in the very first worksheet row should be treated as column names. If set to True, then each column value represents the corresponding table column name. The default value is False.
Hide RowId	Specifies whether the driver should hide the ROWID pseudo-column. The default value is True.
Include Empty Rows	Indicates, whether to include empty lines in the worksheet or skip

	them while reading data. This option is enabled by default.
Read Only	By default, Microsoft Excel is a single-user system. The driver implements a read-only multi-user mode to enable several users to read data from the workbook simultaneously. The possible values are True and False. The default value is False. By setting ReadOnly to True, two or more users are able to access the workbook simultaneously, however only in the read-only mode.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about returned fields and tables they belong to. The default value is True.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.

TUniScript

The TUniDump component has no Excel-specific options.

TUniLoader

The TUniLoader component has no Excel-specific options.

TUniDump

The TUniDump component has no Excel-specific options.

Connecting in Direct mode

Direct connection enables users to connect to Microsoft Excel workbooks directly, without the need for Microsoft Excel to be installed on their machine.

Data Type Mapping

The following table lists the constants for mapping Excel data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
excelBoolean	Maps boolean to Delphi data types.
excelInteger	Maps integer to Delphi data types.
excelDouble	Maps float to Delphi data types.
excelDate	Maps date to Delphi data types.
excelTime	Maps time to Delphi data types.
excelDateTime	Maps datetime to Delphi data types.
string	SQL_VARCHAR

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.9 UniDAC and MongoDB

This article provides a brief overview of the MongoDB data access provider for UniDAC used to establish a connection to MongoDB from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [MongoDB-specific options](#)
 - [TUniConnection](#)
 - [TUniQuery, TUniTable](#)
 - [TUniSQL](#)
 - [TUniStoredProc, TUniScript, TUniDump, TUniLoader, TUniTransaction](#)
- [MongoDB-specific notes](#)
 - [Data types](#)
 - [Query and update operations](#)

- [Accessing a document using the TMongoDocument class](#)
- [Obtaining a document](#)
- [Accessing a document as JSON](#)
- [Accessing the document fields](#)
- [Modifying a document using the "fluent" interface](#)
- [Data Type Mapping](#)

Overview

The main features of MongoDB data access provider are:

- High performance
- Easy deployment
- Full support for the latest versions of the MongoDB server

The full list of MongoDB provider features can be found in [Features](#) page.

Both [Professional and Standard](#) editions of UniDAC include the MongoDB provider. Express Edition of UniDAC does not include the MongoDB provider.

Compatibility

To learn the supported MongoDB versions and clients, refer to the [Compatibility](#) section.

Requirements

Applications that use the MongoDB provider require *libmongoc* and *libbson* client libraries. The MongoDB provider dynamically loads client libraries (for example, libmongoc-1.0.dll and libbson-1.0.dll on Windows) available on user system. To locate DLLs you can set ClientLibrary and BSONLibrary specific options of TUniConnection respectively with paths to client libraries. By default, the MongoDB provider searches for client libraries in the directories specified in the PATH environment variable.

In addition to the standard client libraries, you can use the ones distributed with UniDAC. 32-bit libraries are located in the 'Bin\Win32\' subfolder relative to the folder where UniDAC was installed. 64-bit ones in the 'Bin\Win64\' subfolder. For example:

```
UniConnection1.SpecificOptions.Values['MongoDB.BSONLibrary'] := 'C:\Progra
```

```
UniConnection1.SpecificOptions.Values['MongoDB.ClientLibrary'] := 'C:\Prog
UniConnection1.Connect;
```

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

MongoDB-specific options

Though UniDAC is a library of components that provide unified interface to work with different database servers, it also lets you tune behaviour for each server individually. For thin setup of a certain database server, UniDAC provides server-specific options. These options can be applied to such components as TUniConnection, TUniQuery and TUniTable via their SpecificOptions property. SpecificOptions is a string list. Therefore you can use the following syntax to assign an option value:

```
TUniConnection.SpecificOptions.Values['UseUnicode'] := 'True';
```

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
AdditionalServers	Specifies additional servers to connect to, separated by commas. Each server has to be specified in the <i>host[:port]</i> format as it is described in the official MongoDB documentation .
BSONLibrary	Use the BSONLibrary option to set or get the <i>libbson</i> client library location.
ClientLibrary	Use the ClientLibrary option to set or get the <i>libmongoc</i> client library location.
ConnectionFormat	The values assigned to this property indicate how the Data Source parameter will be treated. The following values are allowed for this property: cfStandard When assigned, the Data Source parameter is treated as a connection URI used for connecting to a self-hosted MongoDB standalone deployment, replica set, or sharded cluster.

	<p>cfDnsSeedList</p> <p>When assigned, the Data Source parameter is treated as a support of a DNS-constructed seed list. The default value is cfStandard.</p>
ConnectionOptions	<p>Connection specific options. See official MongoDB documentation for a full description of these options.</p>
LowerCaseObjectID	<p>Use the option to return ObjectId values in lower case. The default value is False.</p>
SQLEngine	<p>If set to True, the driver will use the SQL language to access data in a MongoDB database, otherwise it will use the standard Mongo query language. The default value is False.</p>
UseUnicode	<p>Enables or disables Unicode support. Affects on the character data fetched from the server. When set to True all character data is stored as WideString, and TStringField is replaced with TWideStringField.</p>

TUniQuery, TUniTable

Option name	Description
AllowAddField	<p>If True, then when editing an existing document, it allows to add new fields to the document. If False, an attempt to add a new field to the document will raise an exceptin. For newly created documents adding new fields is always allowed. The default value is True.</p>
AllowChangeType	<p>If True, when editing an existing document, it allows to assign a value of another type to the existing document field. If False, an attempt to assign a value of another type will raise an exceptin. For newly created documents changing field type is always allowed. The default value is True.</p>
ComplexAsString	<p>If True, then complex fields of a document (which are of <i>object</i>, <i>array</i>, <i>timestamp</i>, <i>binary</i>, <i>regular expression</i> or <i>JavaScript</i> type) are mapped as TStringField and their content is displayed in the Extended JSON format. If False, such fields are mapped as TADTField with its child fields. The default value is False.</p>
DescribeAmount	<p>Specifies the number of sample documents used to create a list of fields in the dataset when <i>DescribeMethod</i> is set to dmGrid. The default value is 25.</p>
DescribeMethod	<p>Defines a way of creating dataset fields.</p> <p>The following values are allowed for this property:</p> <p>dmGrid</p> <p>The field list is generated based on a sample of <i>DescribeAmont</i> documents. The list includes all unique fields from all documents in the sample.</p>

	<code>dmObject</code> The dataset has a single field of the <code>ftADT</code> type, which provides access to the entire document. The default value is <code>dmGrid</code> .
<code>FetchAll</code>	If <code>True</code> , all records of the query are requested from database server when the dataset is being opened. If <code>False</code> , records are retrieved when a data-aware component or a program requests it. The default value is <code>False</code> .

Note: Since parametrized commands are not supported in MongoDB, the MongoDB provider does not support parameters. Also, update SQL-s are not supported too.

TUniSQL

The TUniSQL component has no MongoDB-specific options.

TUniStoredProc, TUniScript, TUniDump, TUniLoader, TUniTransaction

TUniStoredProc, TUniScript, TUniDump, TUniLoader and TUniTransaction components are not supported for the MongoDB provider.

MongoDB-specific notes

This chapter describes several special cases of using the MongoDB provider.

Data types

The MongoDB provider supports the following MongoDB data types:

- String
- 32-bit integer
- 64-bit integer
- Double
- Boolean
- Date
- ObjectId
- Object
- Array

- Timestamp
- Binary
- Regular Expression
- JavaScript
- JavaScript (with scope)
- Null
- Min key
- Max key

By default, document fields of these types are mapped in a dataset as follows:

- *String*, *integer*, *double*, *boolean* and *date* data types are simple types and in a dataset they are mapped to `ftString`, `ftInteger`, `ftLargeint`, `ftBoolean` and `ftDate` fields respectively.
- *Object*, *array*, *timestamp*, *binary*, *regular expression* and *JavaScript* types are complex types and they are mapped either to `ftString` or `ftADT` fields, depending on the [ComplexAsString](#) option value.
- *ObjectId* type is mapped as `ftString` and is displayed as 24-character hexadecimal string.
- *Null* type is mapped as `ftString` and is displayed as 'null'.
- *Min key* and *max key* data types are mapped to `ftString` and are displayed in the [Extended JSON](#) format.

Query and update operations

Since MongoDB is a No-SQL database, the MongoDB provider does not support regular SQL to manage documents. Instead, it supports native [MongoDB command syntax](#) to perform CRUD operations:

- Use the [find](#) command to query documents from a collection, for example:

```
UniQuery1.SQL.Text := '{"find":"restaurants", "filter":{"cuisine":"italian"}}';
UniQuery1.Open;
```

- Use the [insert](#) command to insert documents into a collection, for example:

```
UniQuery1.SQL.Text := '{"insert":"restaurants", "documents":[{"_id":1, "name":...}]}';
UniQuery1.Execute;
```

- Use the [update](#) command to update documents, for example:

```
UniQuery1.SQL.Text := '{"update":"restaurants", "updates":[{"q":{"name":"Vo  
UniQuery1.Execute;
```

- Use the [delete](#) command to delete documents from a collection, for example:

```
UniQuery1.SQL.Text := '{"delete":"restaurants", "deletes":[{"q":{"name":"Vo  
UniQuery1.Execute;
```

Accessing a document using the TMongoDocument class

To access and modify a document in the code, you can use a special TMongoDocument class that has a set of properties and methods for working with the document structure. The data set always contains at least one field of the ftADT type, named in the following format: **<database>_<collection>**, where **<database>** is the database name and **<collection>** is the name of the collection. This field provides access to the entire document using the TMongoDocument class.

Obtaining a document

To obtain an existing document instance, use the TUniQuery.GetObject method:

```
uses  
...  
  MongoObjectsUni;  
...  
var  
  Document: TMongoDocument;  
begin  
  UniQuery1.Edit;  
  Document := UniQuery1.GetObject('test_restaurants') as TMongoDocument;  
...  

```

Or, for a newly created document:

```
uses  
...  
  MongoObjectsUni;  
...  
var  
  Document: TMongoDocument;  
begin  
  UniQuery1.Append;  
  Document := UniQuery1.GetObject('test_restaurants') as TMongoDocument;  
...  

```

Accessing a document as JSON

To access or change the entire document in the JSON format, use the following properties and methods:

- The *AsString* property allows getting or setting the contents of a document as a JSON string, for example:

```
ShowMessage(Document.AsString);  
Document.AsString := '{"_id":1, "name":"volare", "cuisine":"italian"}';
```

- The *LoadFromFile* and *SaveToFile* methods allow loading or saving the contents of a document in a text file.
- The *LoadFromStream* and *SaveToStream* methods allow loading or saving the contents of a document in a stream.

Accessing the document fields

To iterate through the document fields use *FieldCount* and *Fields* property. To access the field value use its *Name* property. To access the field value use its *Value* property. For fields of complex [data types](#) the return value contains the JSON representation of the field.

Example:

```
for i := 0 to Document.FieldCount - 1 do  
  ShowMessage(Document.Fields[i].Name + ': ' + Document.Fields[i].Value);
```

Also, you can access the particular field of the document via its name using the *FieldByName* property, for example:

```
ShowMessage(Document.FieldByName['name'].Value);
```

or

```
ShowMessage(Document['name'].Value);
```

Modifying a document using the "fluent" interface

The *TMongoDocument* class provides a set of *SetXX* methods which allow you to easily change its structure. Methods can be combined one by one into a chain, thus making it easier to write code.

- *SetString*(const Name: string; const Value: string)
- *SetInteger*(const Name: string; const Value: integer)
- *SetInt64*(const Name: string; const Value: Int64)

- SetDouble(const Name: string; const Value: double)
- SetBoolean(const Name: string; const Value: boolean)
- SetDateTime(const Name: string; const Value: TDateTime)
- SetOid(const Name: string; const Value: TJSONOid)

These methods add a simple field named *Name* with the specified *Value* to the document, or change its value if the field exists. When the existing field has the different type, then if the [AllowChangeType](#) property of the dataset is set to True, the field type will also be changed.

Example:

```
Document
.SetString('name', 'Trattoria');
```

- SetTimestamp(const Name: string; const Timestamp: integer; Increment: Cardinal)
- SetBinary(const Name: string; const Binary: TBytes; const SubType: integer)
- SetJavaCode(const Name: string; const Code: string)
- SetJavaScopeCode(const Name: string; const Code: string; const Scope: array of Variant)
- SetRegex(const Name: string; const Pattern, Options: string)

These methods add corresponding complex fields to the document.

Note: For the *SetJavaScopeCode* method, the *Scope* argument is an array of pairs of identifiers and values, representing the scope.

- SetNull(const Name: string)
- SetMinKey(const Name: string)
- SetMaxKey(const Name: string)

Since *Null*, *MinKey* and *MaxKey* are constant types, the methods do not contain the *Value* argument.

- `SetObject(const Name: string)`
- `SetArray(const Name: string)`
- `SetEnd`

These methods are intended to add fields of *Object* and *Array* types to the document. After using *SetObject* or *SetArray* methods, all the following *SetXX* methods add fields to the object or array, not to the document. So, you should use the *SetEnd* method to return to the document level. Example:

```
Document
  .SetObject('address')
  .SetString('city', 'Chicago')
  .SetString('street', 'Dearborn')
  .SetInteger('building', 10)
  .SetEnd
  .SetString('cuisine', 'italian');
```

- `Unset(const Name: string)`

Removes a field with the specified name from the document.

Data Type Mapping

The following table lists the constants for mapping MongoDB data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
<code>mongoString</code>	Maps <code>String</code> to Delphi data types.
<code>mongoNumber</code>	Maps <code>Number</code> to Delphi data types.
<code>mongoBoolean</code>	Maps <code>Boolean</code> to Delphi data types.
<code>mongoObject</code>	Maps <code>Object</code> to Delphi data types.
<code>mongoArray</code>	Maps <code>Array</code> to Delphi data types.
<code>mongoNull</code>	Maps <code>Null</code> to Delphi data types.
<code>mongoObjectId</code>	Maps <code>ObjectId</code> to Delphi data types.

mongoInt32	Maps 32-bit integer to Delphi data types.
mongoInt64	Maps 64-bit integer to Delphi data types.
mongoDouble	Maps Double to Delphi data types.
mongoDateTime	Maps DateTime to Delphi data types.
mongoTimestamp	Maps Timestamp to Delphi data types.
mongoUndefined	Maps Undefined to Delphi data types.
mongoBinary	Maps Binary data to Delphi data types.
mongoRegex	Maps Regular Expression to Delphi data types.
mongoJavaspan	Maps Javascript span to Delphi data types.
mongoJavaScopespan	Maps JavaScript span with scope to Delphi data types.
mongoMinKey	Maps Min key to Delphi data types.
mongoMaxKey	Maps Max key to Delphi data types.
mongoDBPointer	Maps DBPointer to Delphi data types.
mongoDecimal128	Maps Decimal128 to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.10 UniDAC and MySQL

5.1.10.1 MySQL Provider

This article provides a brief overview of the MySQL data access provider for UniDAC used to establish a connection to MySQL databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [MySQL-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)

- [TUniDump](#)
- [Data Type Mapping](#)

Overview

MySQL data access provider is based on the MySQL Data Access Components ([MyDAC](#)) library, which provides direct access to MySQL database servers from Delphi, C++Builder and Lazarus (FPC). The main features of MySQL data access provider are:

- Direct access to server data without using client library. Does not require installation of the client library or other data provider layers (such as BDE and ODBC)
- High performance
- Easy deployment
- Full support for the latest versions of the MySQL server

The full list of MySQL provider features can be found in [Features](#).

Both [Professional and Standard Editions](#) of UniDAC include the MySQL provider. For Express Edition of UniDAC, the MySQL provider can be installed with MyDAC.

Compatibility

To learn about MySQL database server compatibility, refer to the [Compatibility](#) section.

Requirements

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

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

If you are working with Embedded server, you should have access to Embedded MySQL server library (libmysqld.dll).

	<p>validity of the server certificate instead. If set to True, the driver will trust the server certificate.</p>
Interactive	<p>Determines the inactivity timeout before the server breaks the connection. If true, the server breaks the connection after number of seconds specified in interactive_timeout sever variable, otherwise wait_timeout is used. The default value is false. The interactive_timeout and wait_timeout variables can be set in my.ini file.</p>
IPVersion	<p>Use the IPVersion property to specify Internet Protocol Version.</p> <p>Supported values:</p> <p>ivIPBoth Specifies that either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) will be used.</p> <p>ivIPv4 (default) Specifies that Internet Protocol Version 4 (IPv4) will be used.</p> <p>ivIPv6 Specifies that Internet Protocol Version 6 (IPv6) will be used.</p> <p>Note: When the TIPVersion property is set to ivIPBoth, a connection attempt will be made via IPv6 if it is enabled on the operating system. If the connection attempt fails, a new connection attempt will be made via IPv4.</p>
NullForZeroDelphiDate	<p>Use the NullForZeroDelphiDate property to hide the '30-12-1899' dates. If NullForZeroDelphiDate is set to True, the values of all datetime fields will be changed to Null. If the property is set to False, the '30-12-1899' value will be used as an ordinary date. The default value is false.</p>
OptimizedBigint	<p>Setting this option converts all fields with field length less than 11 of TLargeIntField type into TIntegerField. This allows to process fields that are results of numeric function or cast values as usual Integer fields. The default value is False.</p>
Protocol	<p>Specifies which protocol to use when connecting to the server:</p> <p>mpDefault Similar to mpTCP, except the cases when you connect to a local server and the OS supports sockets (Unix) or named pipes (Windows), they are used instead of TCP/IP to connect to the server.</p> <p>mpTCP Use TCP/IP to connect to the server.</p> <p>mpSocket Uses sockets to connect to the server. Can be used with Direct</p>

	<p>set to False and libmysql.dll 4.1.</p> <p>mpPipe Use NamedPipes to connect to the server.</p> <p>mpMemory To connect to the server using SharedMem. Can be used with Direct set to False and libmysql.dll 4.1.</p> <p>mpSSL Use protected SSL connection with the server.</p> <p>mpHttp Uses HTTP Network Tunneling to connect to the server.</p>
HttpUrl	Holds the url of the tunneling PHP script.
HttpUsername	Holds the user name for HTTP authorization.
HttpPassword	Holds the password for HTTP authorization.
ProxyHostname	Holds the host name or IP address to connect to proxy server.
ProxyPort	Used to specify the port number for TCP/IP connection with proxy server.
ProxyUsername	Holds the proxy server account name.
ProxyPassword	Holds the password for the proxy server account.
SSLCACert	CACert is the pathname to the certificate authority file.
SSLCert	Cert is the pathname to the certificate file.
SSLChipherList	ChipherList is a list of allowable ciphers to use for SSL encryption.
SSLKey	Key is the pathname to the key file.
UseUnicode	<p>Informs server that all data between client and server sides will be passed in UTF-8 coding. Setting this option converts all fields of TStringField type into TWideStringField that allows to work correctly with symbols of almost all languages simultaneously. On the other hand, it causes a delay in working. The default value is False.</p>

TUniSQL

Option name	Description
CommandTimeout	<p>Specifies the amount of time that is expired before an attempt to execute a command is considered unsuccessful. Measured in seconds.</p> <p>If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect. The default value is 0 (infinite).</p>

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
-------------	-------------

BinaryAsString	Specifies the method of representation of BINARY and VARBINARY types. If set to True, binary field data will be retrieved as a string and handled by the TStringField class. The default value is True.
CheckRowVersion	Determines whether the dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data. If CheckRowVersion is True and DataSet has timestamp field when only this field is added into WHERE clause of the generated SQL statement. If CheckRowVersion is True, but there is no TIMESTAMP field, then to WHERE clause all non-BLOB fields will be added. The default value is False.
CommandTimeout	Specifies the amount of time that is expired before an attempt to execute a command is considered unsuccessful. Measured in seconds. If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect. The default value is 0 (infinite).
CreateConnection	Specifies whether an additional connection to a server should be established to execute an additional query in the FetchAll=False mode. If a DataSet is opened in FetchAll=False, the current connection is locked until all records have been fetched. If this option is set to True, an additional connection is created to prevent locking of the current connection when fetching data. In case you have the FetchAll or UniDirectional option enabled, and you don't want an additional connection to be created for each Open/ExecSQL (for example, when dealing with TEMPORARY TABLES or SESSION VARIABLES), set CreateConnection to False.
EnableBoolean	Specifies the method of representation of TINYINT(1) fields. If set to True, these fields will be represented as TBooleanField; otherwise, as TSmallintField. The default value is True.
FetchAll	When set to True, all records of the query are requested from the database server when dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important. When the FetchAll property is False, the first call to Locate and LocateEx methods may take a lot of time to retrieve additional records to the client side.
FieldsAsString	All non-BLOB fields are stored as string (native MySQL format). The default value is False.
NullForZeroDate	For datetime fields with invalid values, for example '2002-12-32', MySQL returns on fetch '0000-00-00' value. According to NullForZeroDate option this value will be represented as Null or

	'0001-01-01' ('0100-01-01' for CLR). The default value is True.
--	---

TUniScript

The TUniScript component has no MySQL-specific options.

TUniLoader

Option name	Description
LockTable	Locks tables while inserting data.
Delayed	Uses INSERT DELAYED syntax.
RowsPerQuery	Use the RowsPerQuery property to get or set the number of rows that will be send to the server for one time. The default value is 0. In this case rows will be grouped by 16Kb (the default value of net_buffer_length).
DuplicateKeys	Use the DuplicateKeys property to specify in what way conflicts with duplicated key values will be resolved.
QuoteNames	Use the QuoteNames option to quote all database object names in automatically generated SQL statements, such as UPDATE statements. The default value is False.

TUniDump

Option name	Description
AddLock	Use the AddLock property to execute LOCK TABLE before data insertion. Used only with doData in P:Devart.MyDac.TMyDump.Objects.
BackupData	Use the option to backup the data in a table. The default value is True.
BackupStoredProcs	Use the enable backup of stored procedures. The default value is False.
BackupTables	Use the option to enable backup of the table structure. The default value is False.
BackupTriggers	Use the option to enable backup of triggers. The default value is False.
BackupViews	Use the option to enable backup of views. The default value is False.
CommitBatchSize	Use the CommitBatchSize option to add COMMIT statement to script after the specified number of strings when dumping table data. The option is useful for recovering large amounts of data. The default value is 0.
DisableKeys	Add /*!40000 ALTER TABLE ... DISABLE KEYS */ before inserting data. Used only with doData in

	P:Devart.MyDac.TMyDump.Objects.
InsertType	<p>Specifies how rows will be inserted into a table.</p> <p>Supported values:</p> <p>itInsert (default) New rows will be inserted into an existing table. If a duplicate entry is encountered, an exception will be raised.</p> <p>itInsertIgnore The insert operation will fail silently for rows containing an unmatched value, but inserts rows that are matched, without raising an exception.</p> <p>itReplaceInto If an old row in a table has the same value as a new row, the old row will be deleted before the new row is inserted.</p>
HexBlob	If the HexBlob property is True, the BLOB values are presented in hexadecimal notation.
UseExtSyntax	Set the UseExtSyntax property to use extended syntax of INSERT on data insertion. Used only with doData in P:Devart.MyDac.TMyDump.Objects.
UseDelayedIns	Set the UseDelayedIns property to use INSERT DELAYED. Used only with doData in P:Devart.MyDac.TMyDump.Objects.

Data Type Mapping

The following table lists the constants for mapping MySQL data types to Delphi data types.

See [Data Type Mapping](#) for more information.

Constant	Description
myBigint	Maps BIGINT to Delphi data types.
myBigintUnsigned	Maps BIGINT UNSIGNED to Delphi data types.
myBinary	Maps BINARY to Delphi data types.
myBit	Maps BIT to Delphi data types.
myBlob	Maps BLOB to Delphi data types.
myChar	Maps CHAR to Delphi data types.
myDate	Maps DATE to Delphi data types.
myDatetime	Maps DATETIME to Delphi data types.
myDecimal	Maps DECIMAL to Delphi data types.
myDouble	Maps DOUBLE to Delphi data types.
myEnum	Maps ENUM to Delphi data types.
myFloat	Maps FLOAT to Delphi data types.

myInt	Maps INTEGER to Delphi data types.
myIntUnsigned	Maps INTEGER UNSIGNED to Delphi data types.
myJSON	Maps JSON to Delphi data types.
myLongBlob	Maps LONGBLOB to Delphi data types.
myLongText	Maps LONGTEXT to Delphi data types.
myMedium	Maps MEDIUMINT to Delphi data types.
myMediumBlob	Maps MEDIUMBLOB to Delphi data types.
myMediumText	Maps MEDIUMTEXT to Delphi data types.
myMediumUnsigned	Maps MEDIUMINT UNSIGNED to Delphi data types.
myNull	Maps NULL to Delphi data types.
mySet	Maps SET to Delphi data types.
mySmall	Maps SMALLINT to Delphi data types.
mySmallUnsigned	Maps SMALLINT UNSIGNED to Delphi data types.
myText	Maps TEXT to Delphi data types.
myTime	Maps TIME to Delphi data types.
myTimestamp	Maps TIMESTAMP to Delphi data types.
myTiny	Maps TINY to Delphi data types.
myTinyBlob	Maps TINYBLOB to Delphi data types.
myTinyText	Maps TINYTEXT to Delphi data types.
myTinyUnsigned	Maps TINYINT UNSIGNED to Delphi data types.
myUnsignedTypes	Maps UNSIGNED TYPES to Delphi data types.
myVarbinary	Maps VARBINARY to Delphi data types.
myVarchar	Maps VARCHAR to Delphi data types.
myYear	Maps YEAR to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.10.2 Connecting via SSL

Security is critical when exchanging messages between the server and client. Data protection can be achieved through several methods, including establishing an SSL-encrypted connection from a Delphi application to a remote MySQL server. MySQL supports data transfer over TCP/IP, both with or without SSL encryption.

You can establish a secure connection to a MySQL server using [SecureBridge](#), a solution offered by Devart. It allows you to embed an SSL client into a Delphi or C++ Builder application.

This page demonstrates how to connect to MySQL with UniDAC using SSL as the encryption

method.

- [Connect Using SecureBridge](#)
- [Connect Using the OpenSSL library](#)

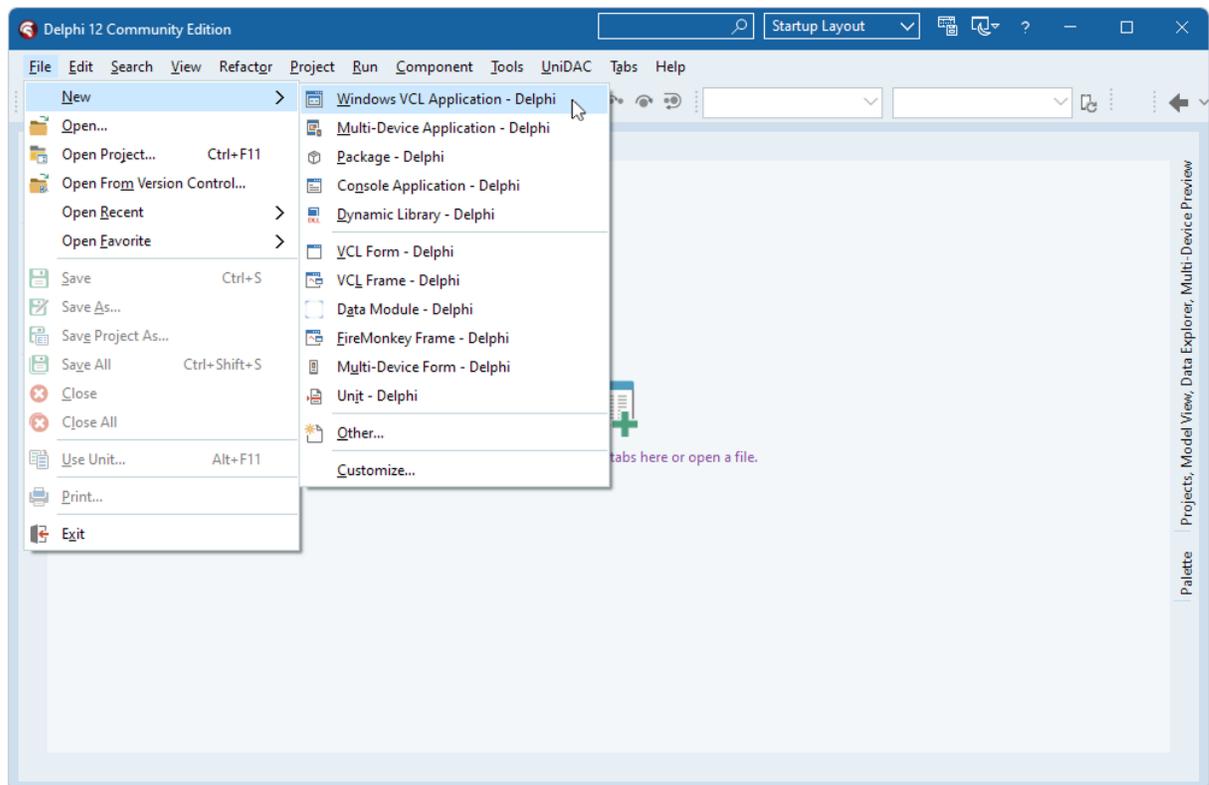
Connect Using SecureBridge

To connect to MySQL using UniDAC and SecureBridge components, no external libraries are required. SecureBridge provides the necessary SSL functionality directly within the components.

Prerequisites:

- Install SecureBridge.
- Install the TCRSSLIHandler component in RAD Studio to bind UniDAC with SecureBridge. For instructions, see `Readme.html`—by default, it is located in `C:\Users\\Documents\Devart\UniDAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge`.

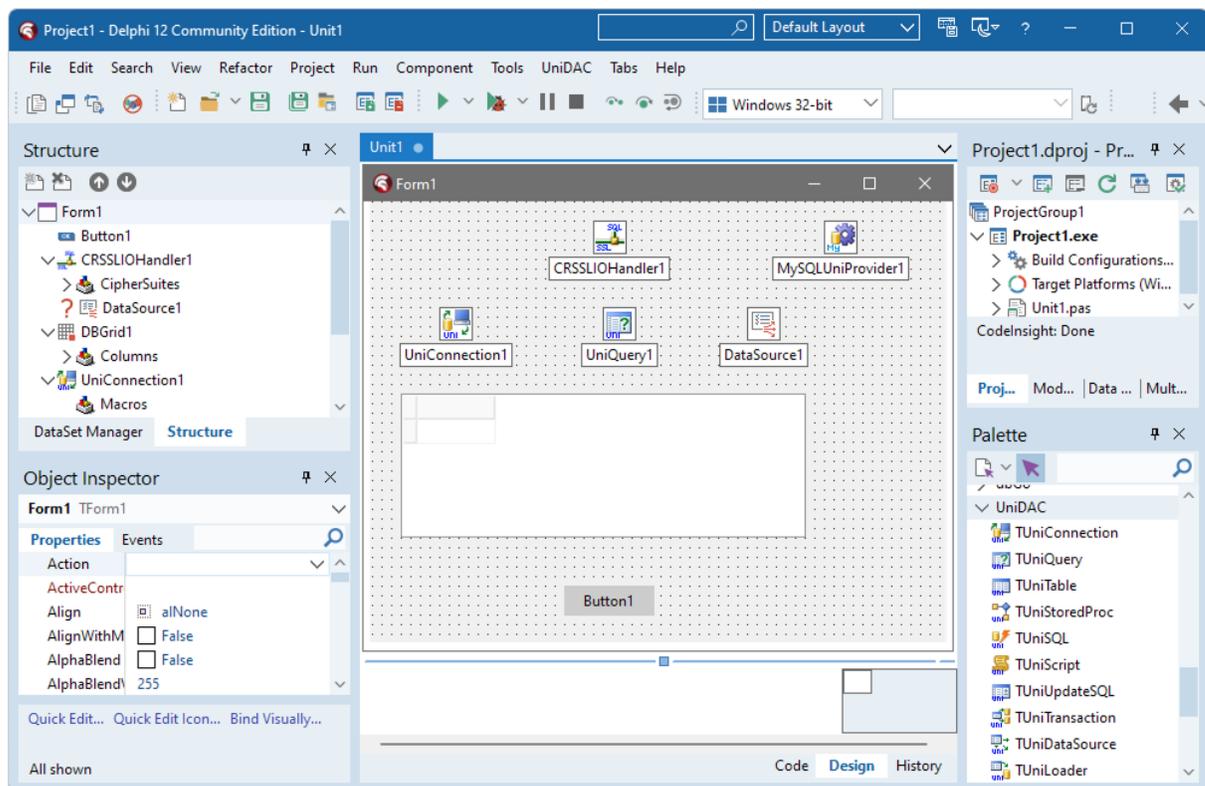
1. In RAD Studio, select **File > New > Windows VCL Application - Delphi**.



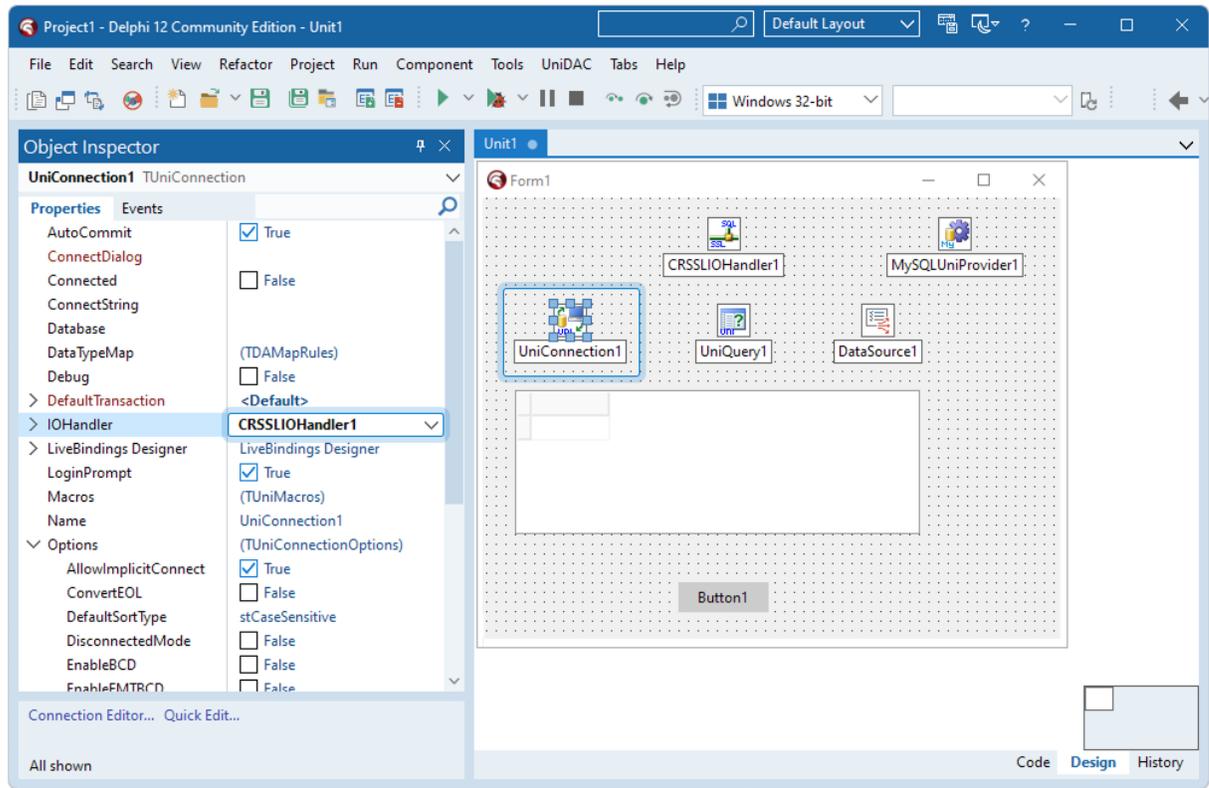
2. Place the following components from **Palette** on the form:

- **TCRSSLIOHandler**
- **TMySQLUniProvider**
- **TUniConnection**
- **TUniQuery**
- **TDataSource**
- **TDBGrid**
- **TButton**

These components are required to establish an SSL connection to a MySQL server using UniDAC, execute a query, and display the retrieved data in a grid.

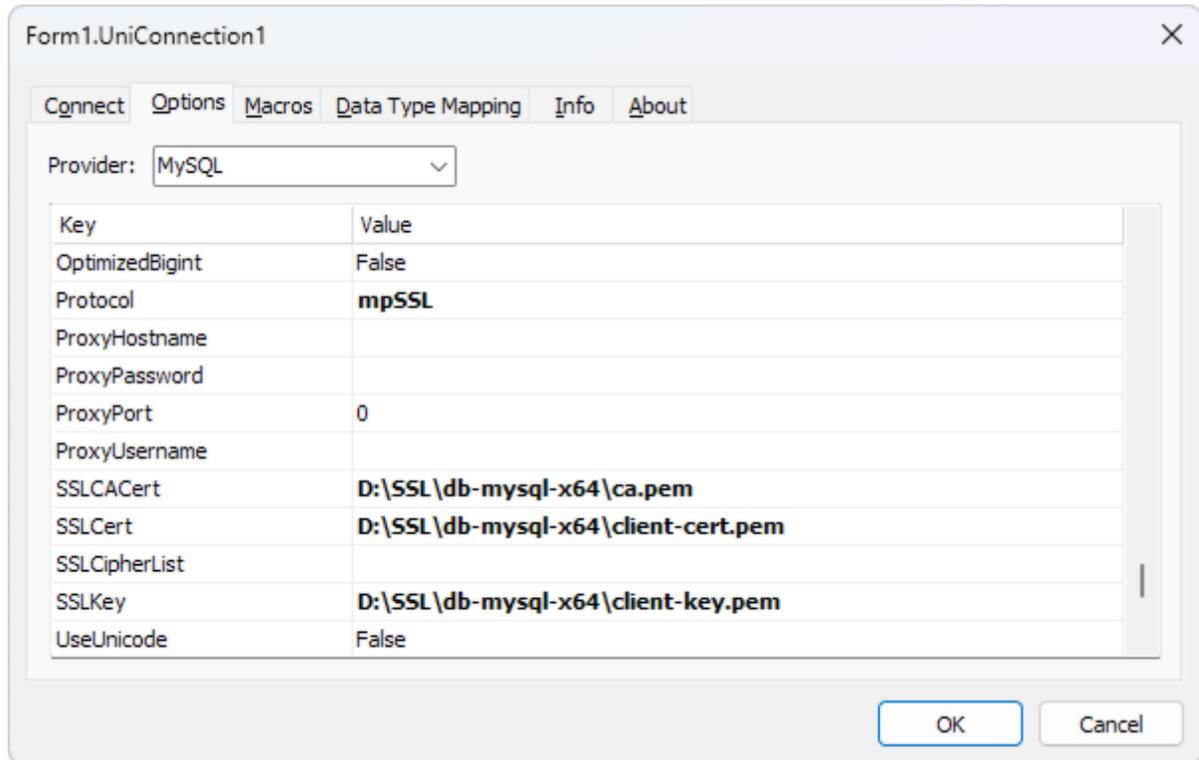


3. Select the **TUniConnection** component and, in **Object Inspector**, set the **IOHandler** property to the instance of **TCRSSLIOHandler** (**CRSSLIOHandler1**).



4. Double-click the **TUniConnection** component and, on the **Options** tab, fill out the fields:

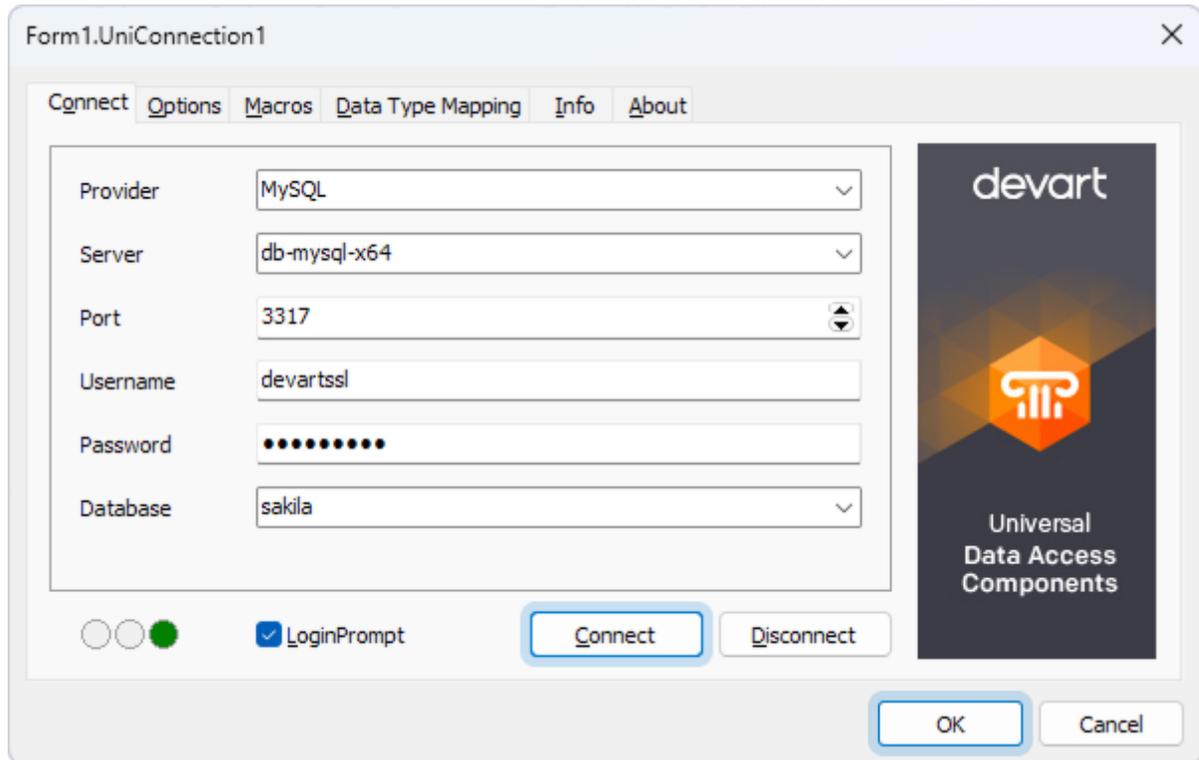
- **Provider** – Select **MySQL**.
- **Protocol** – Select **mpSSL**.
- **SSLCACert** – Specify the full path of the Certificate Authority certificate file.
- **SSLCert** – Specify the full path of the client certificate file.
- **SSLKey** – Specify the full path of the client private key file.



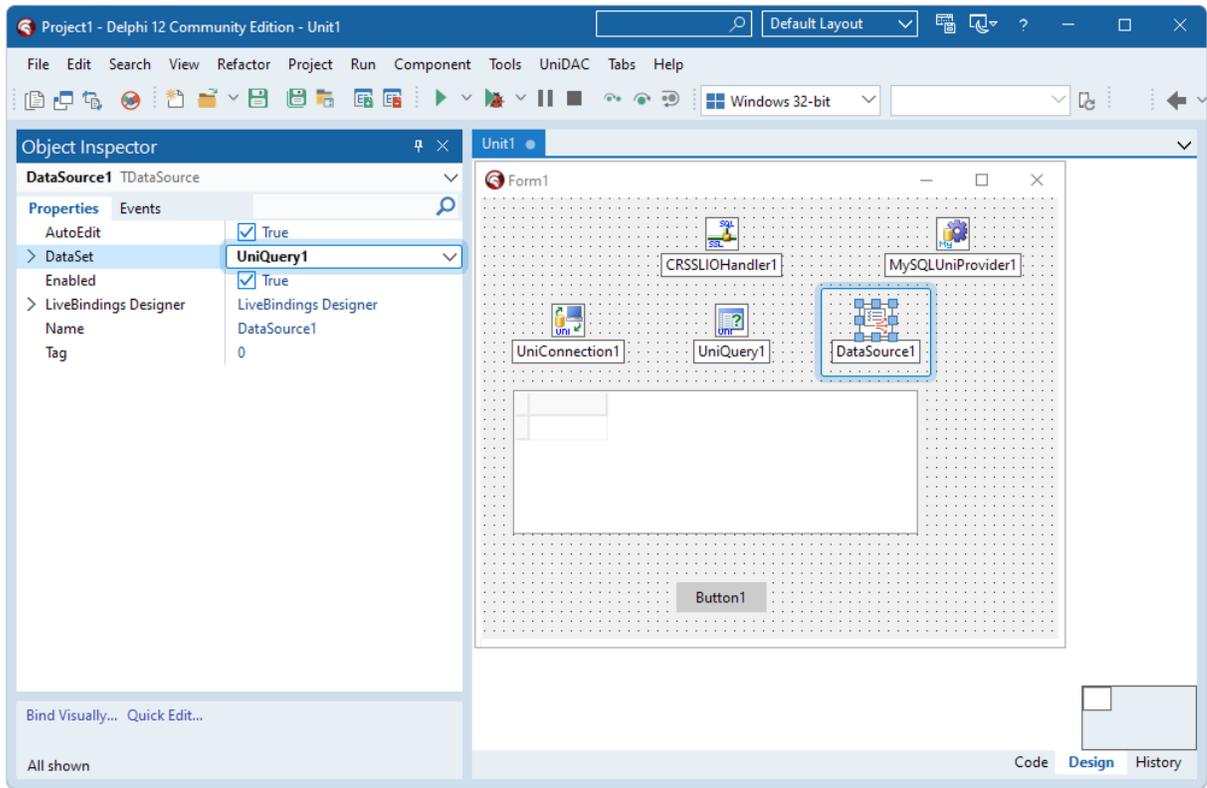
5. On the **Connect** tab, fill out the fields:

- **Provider** – Select **MySQL**.
- **Server** – Enter the host name or IP address of the MySQL server.
- **Port** – Specify the MySQL server port.
- **Username** – Enter the username for the account on the MySQL server.
- **Password** – Enter the password for the account on the MySQL server.
- **Database** – Select the database.

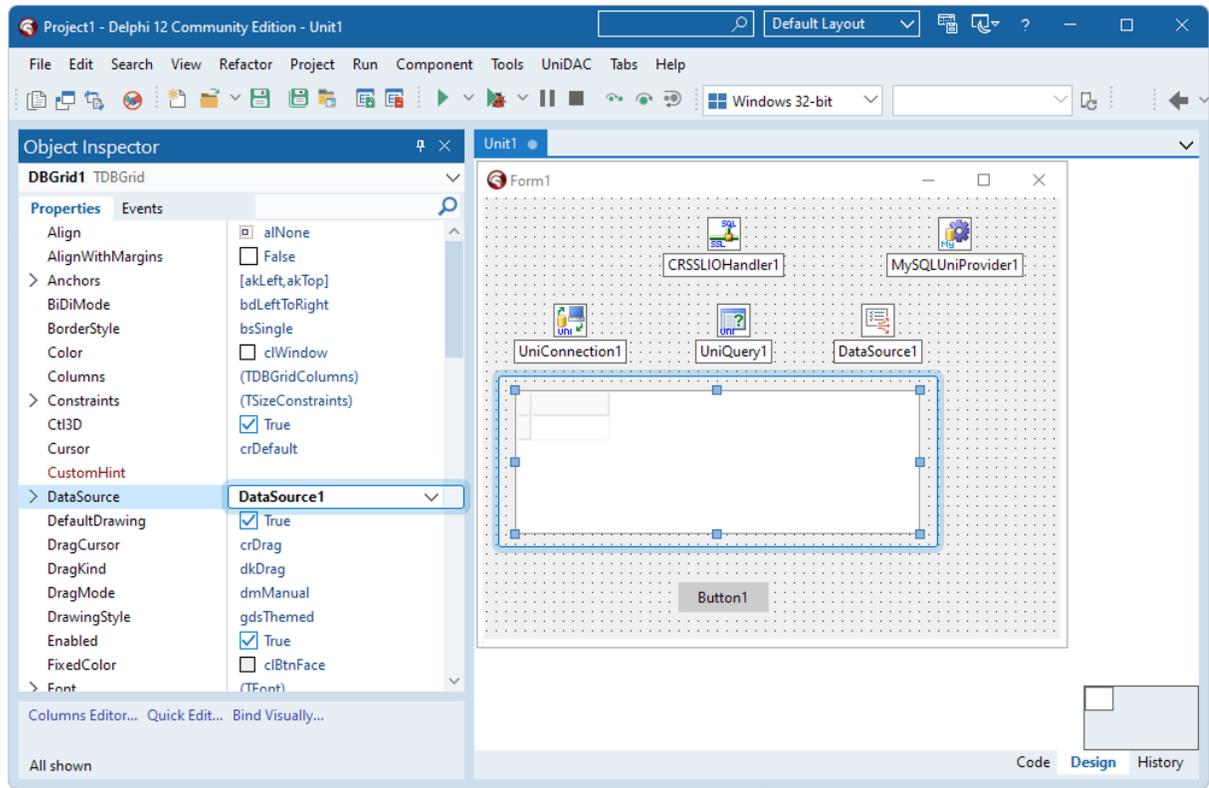
6. Click **Connect** to test the connection to the MySQL server, then click **OK** to close the dialog.



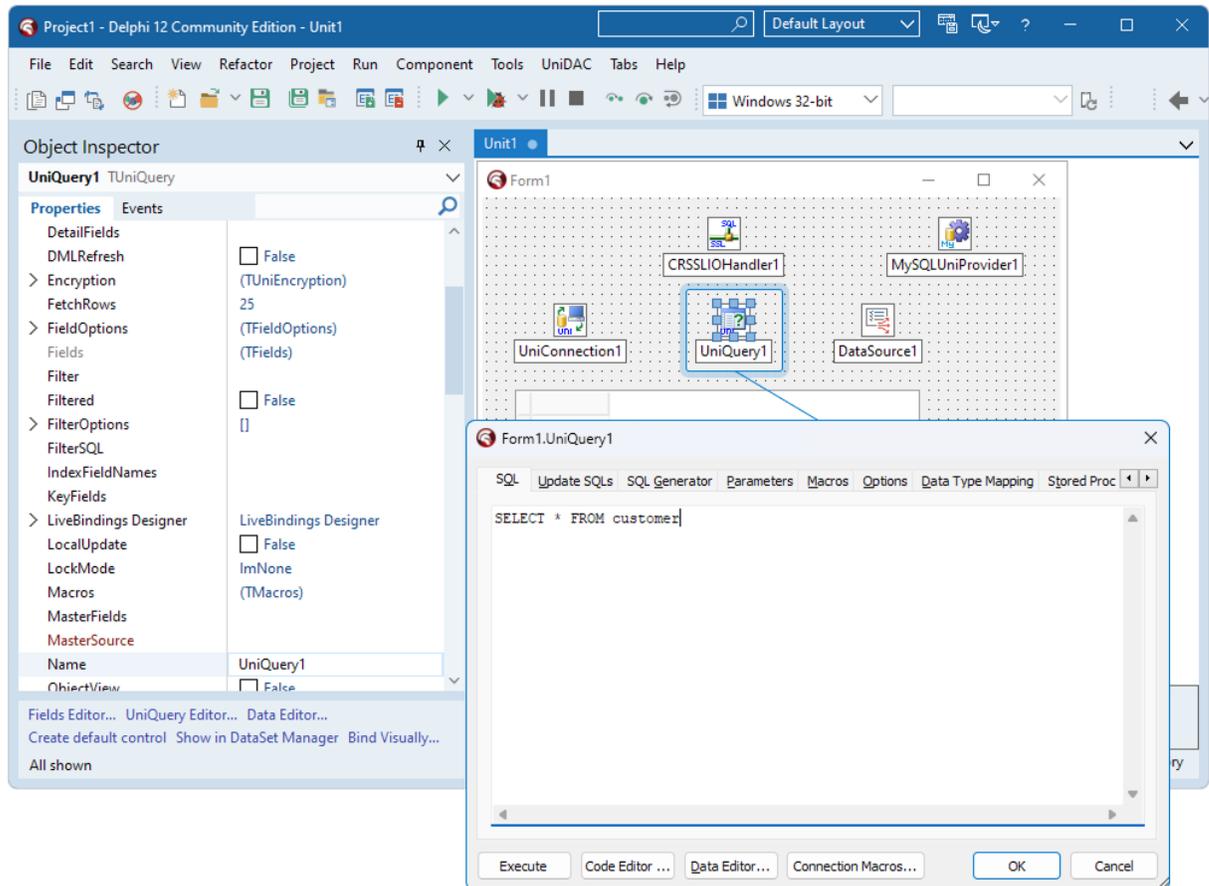
7. Select the **TDataSource** component and set the **DataSet** property to the instance of **TUniQuery (UniQuery1)**.



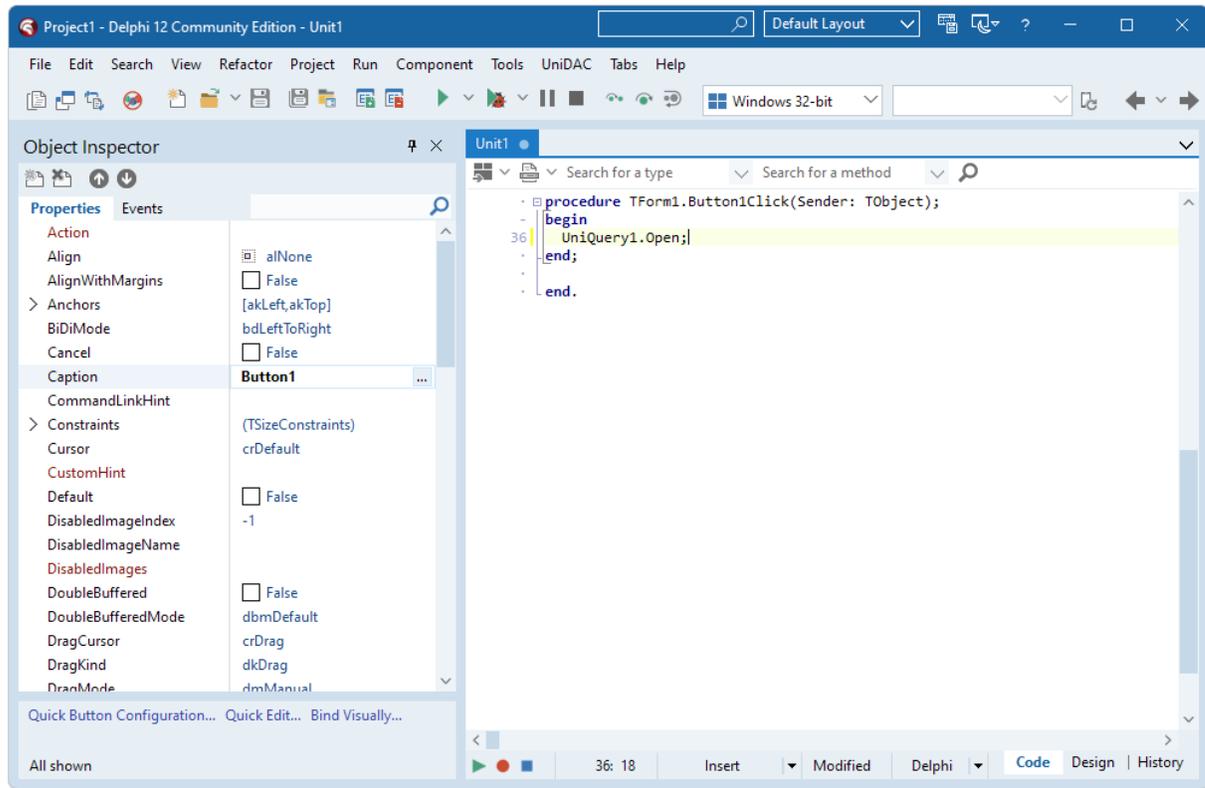
8. Select the **TDBGrid** component and set the **DataSource** property to the instance of **TDataSource (DataSource1)**.



9. Double-click the **TUniQuery** component, enter a SQL query to be run against the MySQL database, and then click **OK**.

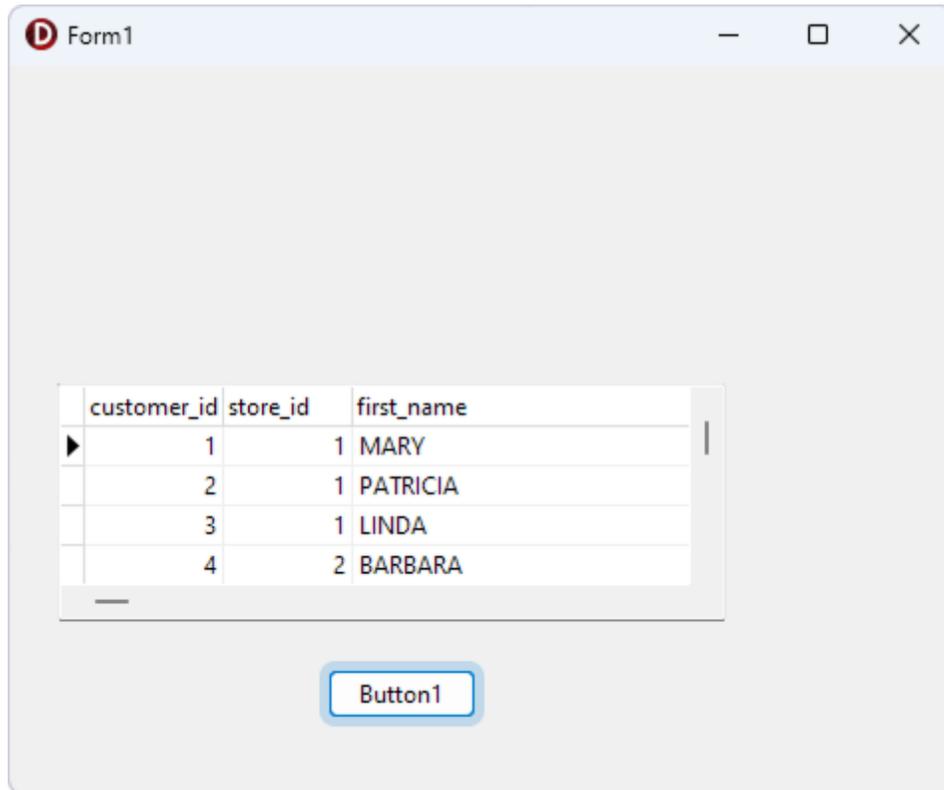


10. Double-click the **TButton** component and add code to call the **Open** method on the **UniQuery1** object to activate the dataset when the button is clicked.



11. Press **F9** to compile and run the application.

12. In the form that appears, click **Button1** to run the query. Data appears in the grid.



Connect Using the OpenSSL library

Another way to embed SSL client functionality into your Delphi app that uses UniDAC components to access MySQL is by using the OpenSSL library. The OpenSSL library implements the SSL protocol and enables servers to communicate with their clients securely.

Important: The authenticity of the CA certificate is not verified when you use the OpenSSL library. UniDAC supports only the basic OpenSSL functionality, which does not include CA certificate verification. To configure advanced SSL features such as CA certificate verification, use [SecureBridge components](#) instead of OpenSSL.

To establish an SSL connection, set the following options:

- `SSLCACert` – The full path of the Certificate Authority certificate file.
- `SSLCert` – The full path of the client certificate file.
- `SSLKey` – The full path of the client private key file.
- `SSLCipherList` – A list of allowed ciphers to use for SSL encryption.

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.10.3 Connecting via SSH

SSH is a protocol that allows users to securely log in 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.

You can use SSH to secure the network connection between a Delphi application and a MySQL server. An SSH connection enables you to run shell commands as if you were physically operating the remote machine.

This page demonstrates how to use UniDAC to connect to a MySQL server securely via an SSH tunnel.

To follow the steps on this page, you need an SSH client and an SSH server. You can create them using [SecureBridge](#), a solution provided by Devart. It enables you to embed an SSH client into your Delphi application and, if needed, configure an SSH server. For more information, see the SecureBridge [tutorial on configuring an SSH server](#). You can also build the SSHServer [demo project](#), which is distributed with SecureBridge.

Your MySQL server must be configured to accept connections from the SSH tunnel.

- [Connect Using SecureBridge](#)
- [Connect Using OpenSSH or Any Other Third-Party SSH Tunneling Tools](#)

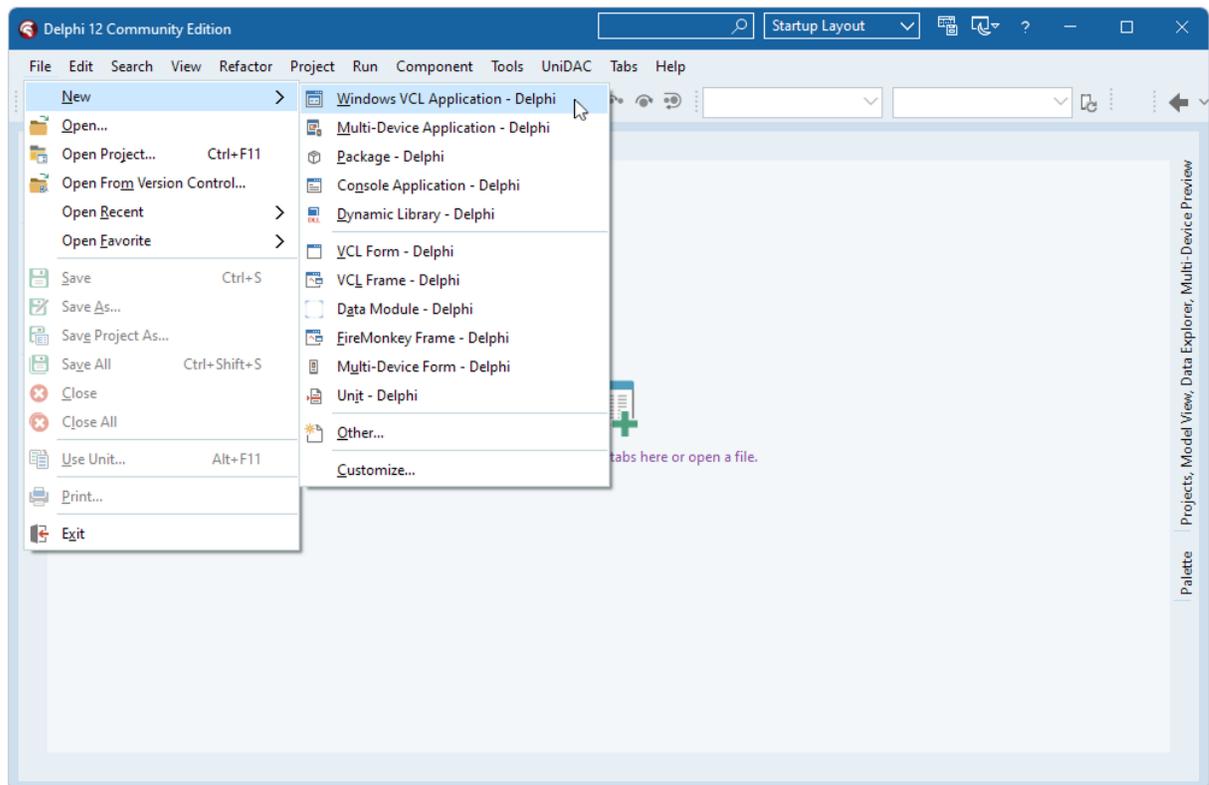
Connect Using SecureBridge

You can establish a secure SSH tunnel using SecureBridge without the need for third-party applications. SecureBridge provides all the necessary components to create an SSH client within your Delphi application.

Prerequisites:

- Install SecureBridge.
- Install the TCRSSHIOHandler component in RAD Studio to bind UniDAC with SecureBridge. For instructions, see `Readme.html`—by default, it is located in `C:\Users\\Documents\Devart\UniDAC` for RAD Studio XX\Demos\TechnologySpecific\SecureBridge.

1. In RAD Studio, select **File > New > Windows VCL Application - Delphi**.

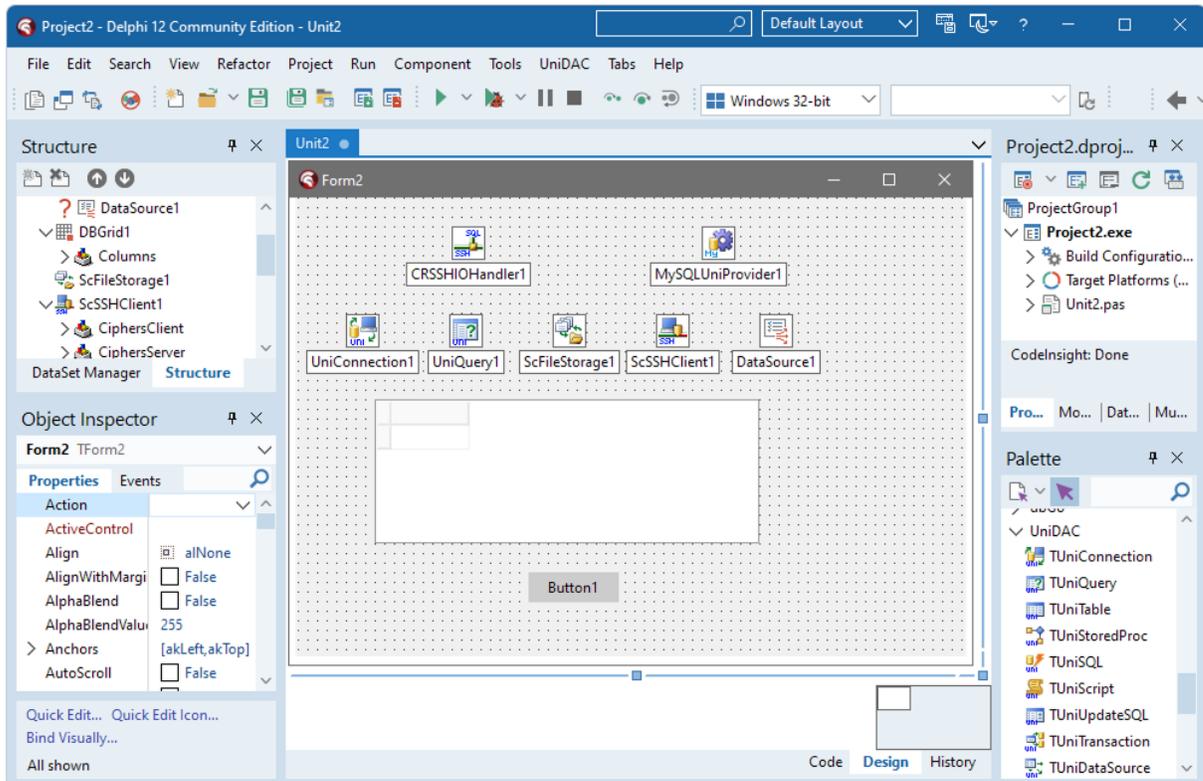


2. Place the following components from **Palette** on the form:

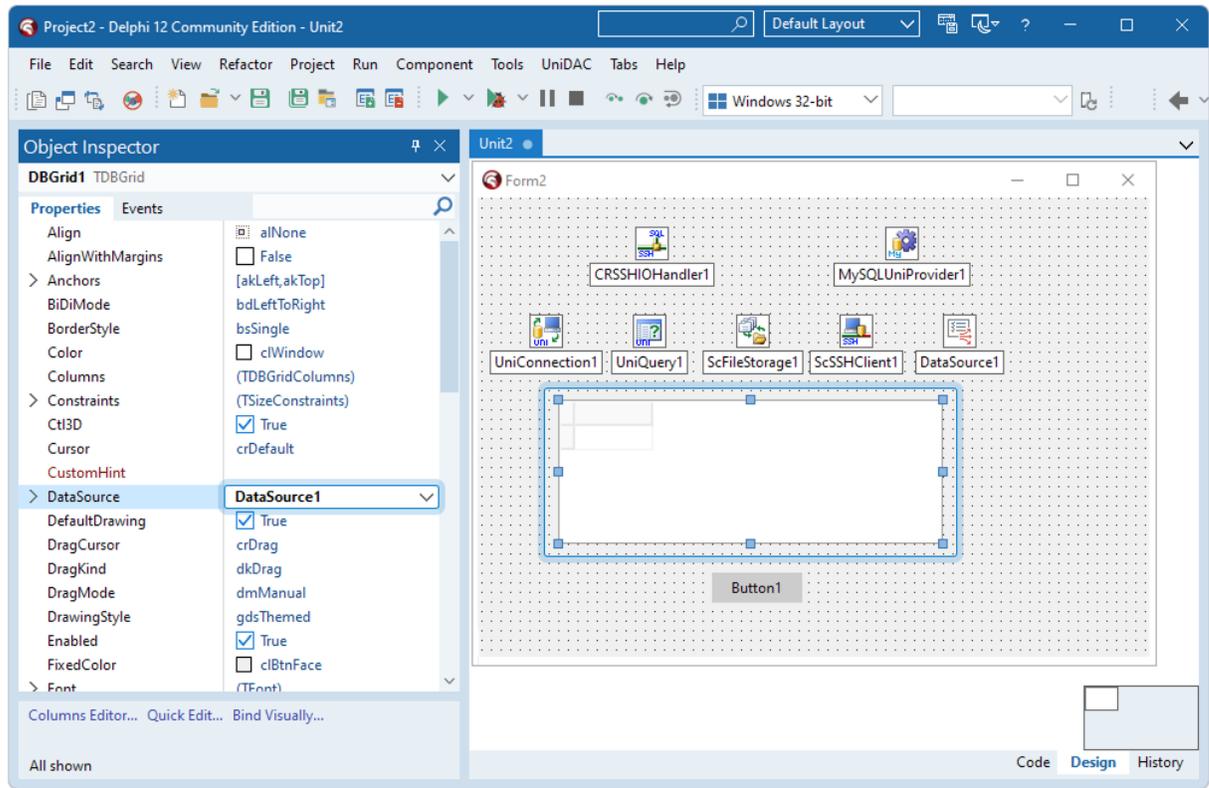
- **TCRSSHIOHandler**
- **TMySQLUniProvider**
- **TUniConnection**
- **TUniQuery**
- **TScFileStorage**
- **TScSSHClient**
- **TDataSource**

- TDBGrid
- TButton

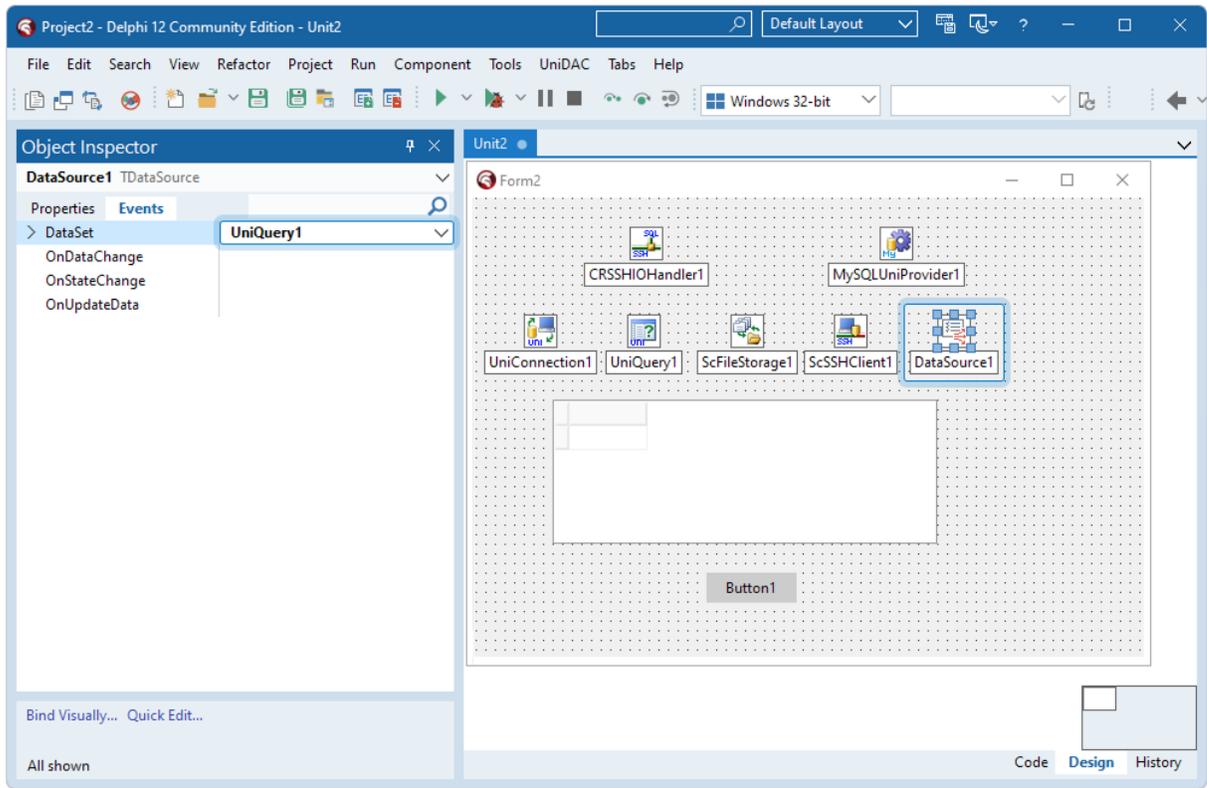
These components are required to establish an SSH connection to a MySQL server using UniDAC, execute a query, and display the retrieved data in a grid.



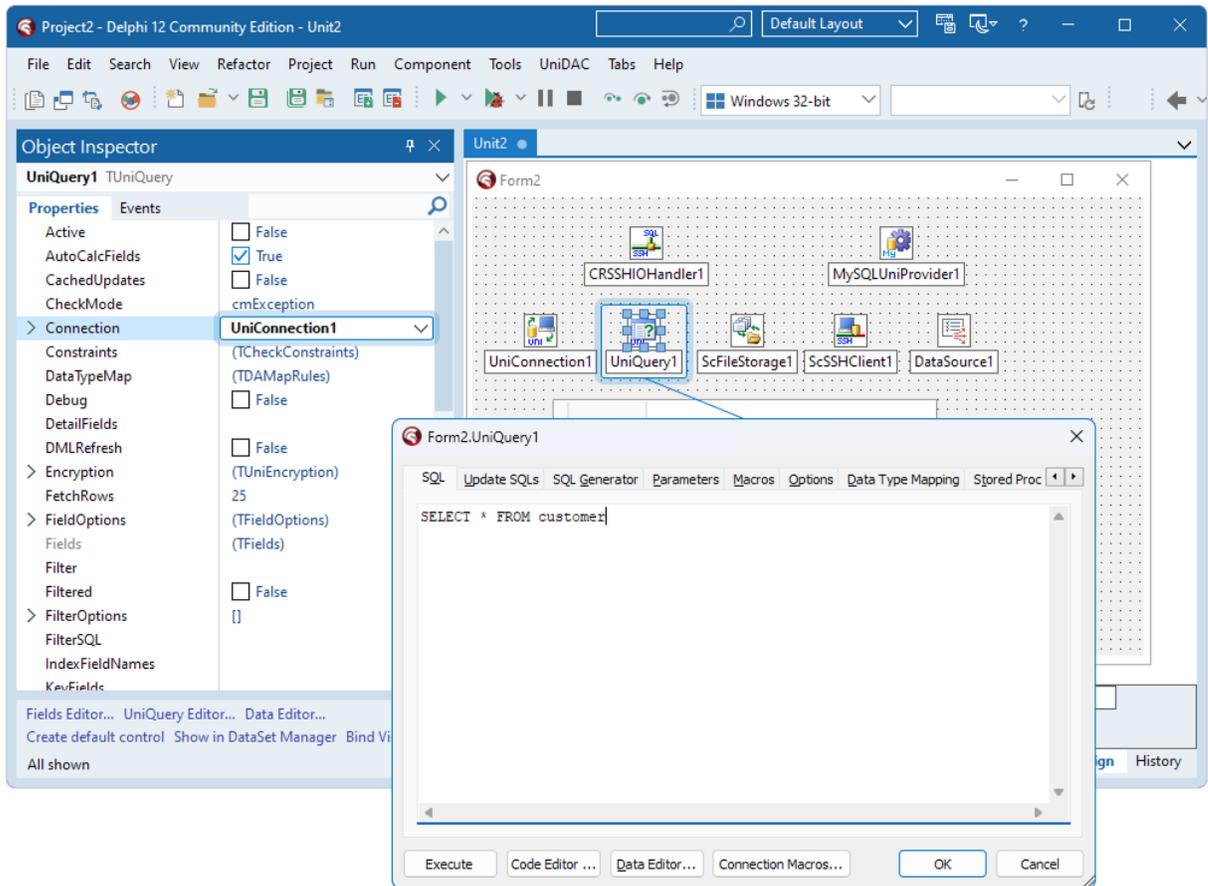
3. Select the **TDBGrid** component and, in **Object Inspector**, set the **DataSource** property to the instance of **TDataSource (DataSource1)**.



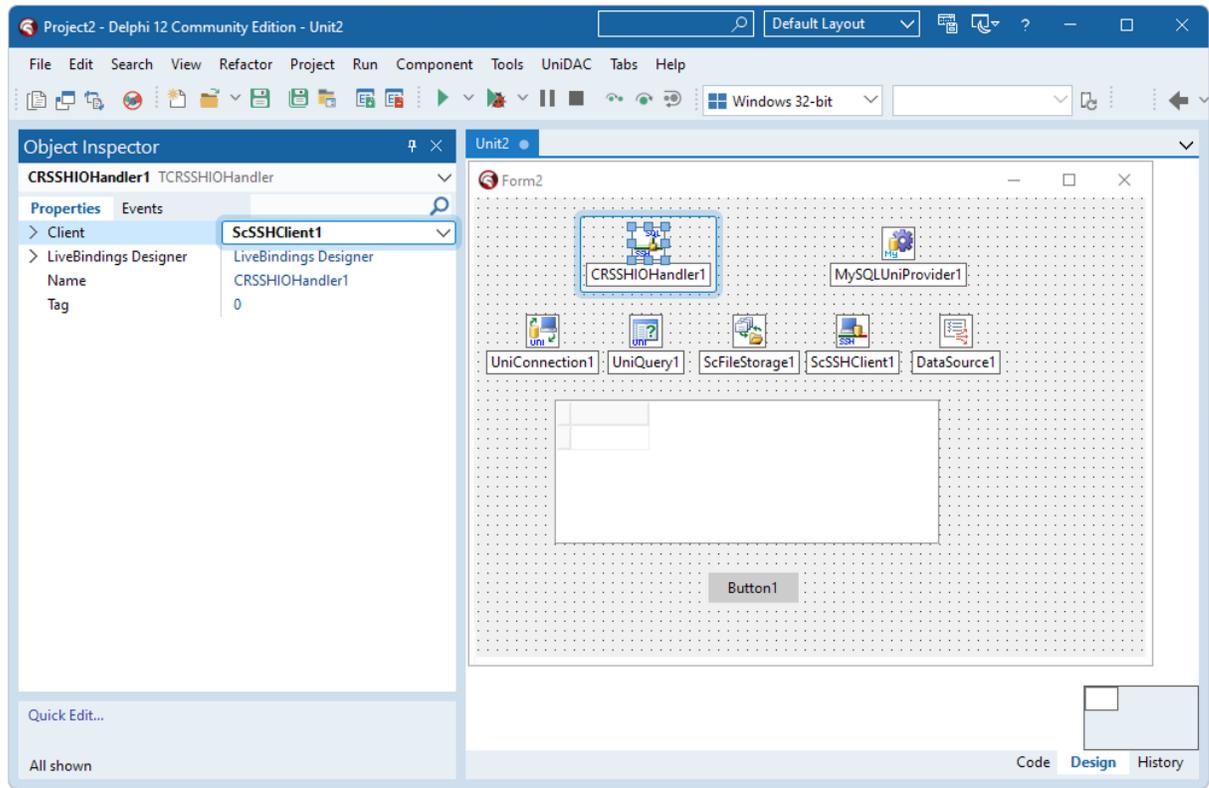
4. Select the **TDataSource** component and set the **DataSet** property to the instance of **TUniQuery (UniQuery1)**.



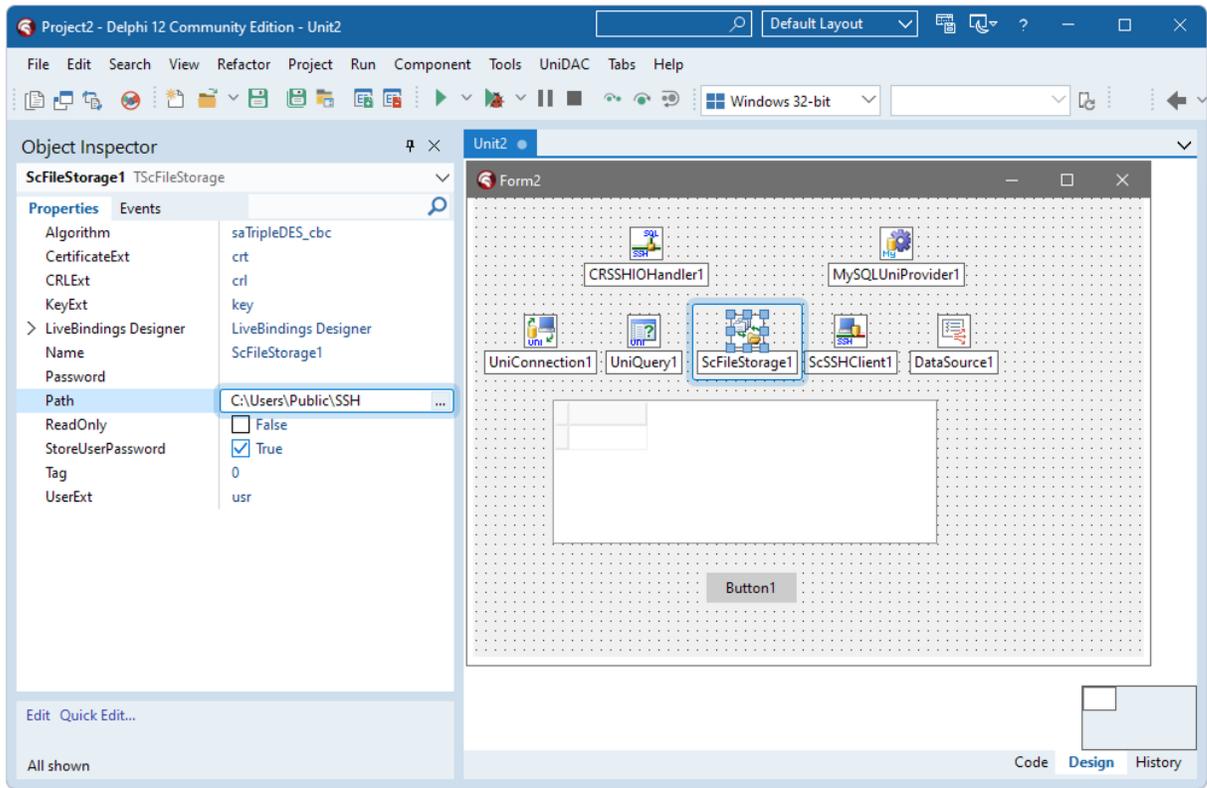
5. Select the **TUniQuery** component and set the **Connection** property to the instance of **TUniConnection (UniConnection1)**.
6. Then, double-click the **TUniQuery** component, enter a SQL query to be run against the MySQL database, and click **OK**.



7. Select the **TCRSSHIOHandler** component and set the **Client** property to the instance of **TScSSHClient (ScSSHClient1)**.

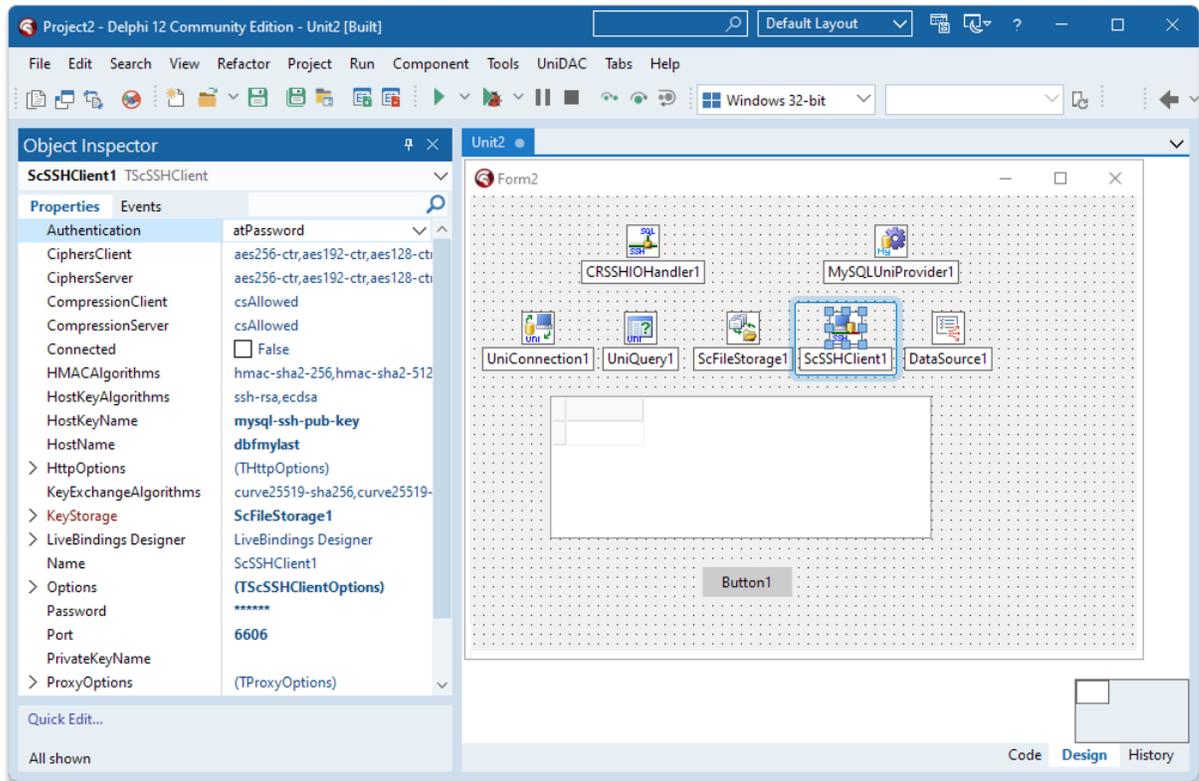


8. Select the **TScFileStorage** component and, in the **Path** property, specify the directory where the keys are stored.



9. Select the **TScSSHClient** component and assign values to the following properties:

- **Authentication** – Select the value depending on the authentication method applicable for your SSH server: **atPassword** or **atPublicKey**.
- **HostKeyName** – Specify the filename of the SSH server public key.
- **Hostname** – Enter the host name or IP address of the SSH server.
- **KeyStorage** – Set the property to the instance of **TScFileStorage** (**ScFileStorage1**).
- **Password** – For password authentication, enter the password for the account on the SSH server.
- **Port** – Specify the SSH port.
- **PrivateKeyName** – For public key authentication, specify the filename of the client private key.
- **User** – Enter the username for the account on the SSH server.

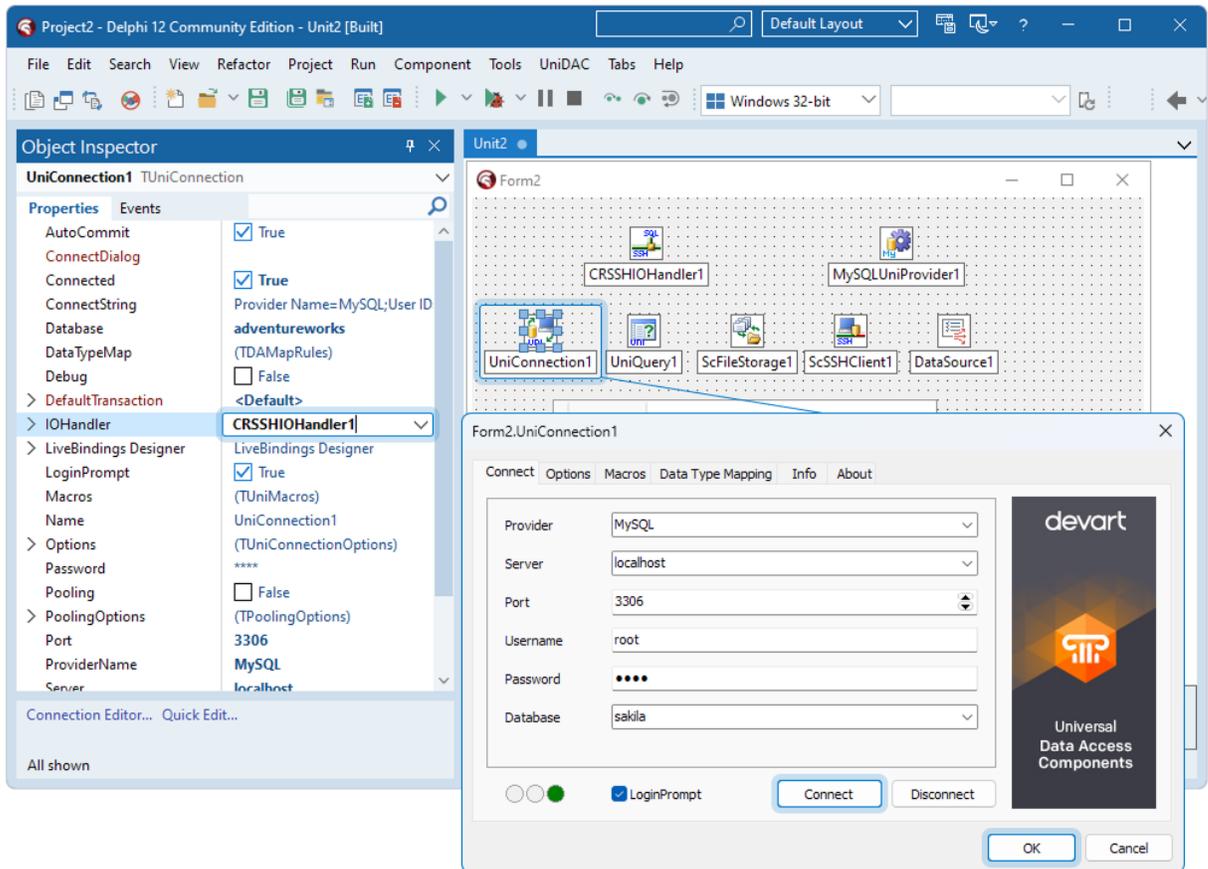


10. Select the **TUniConnection** component and set the **IOHandler** property to the instance of **TCRSSHIOHandler (CRSSHIOHandler1)**.

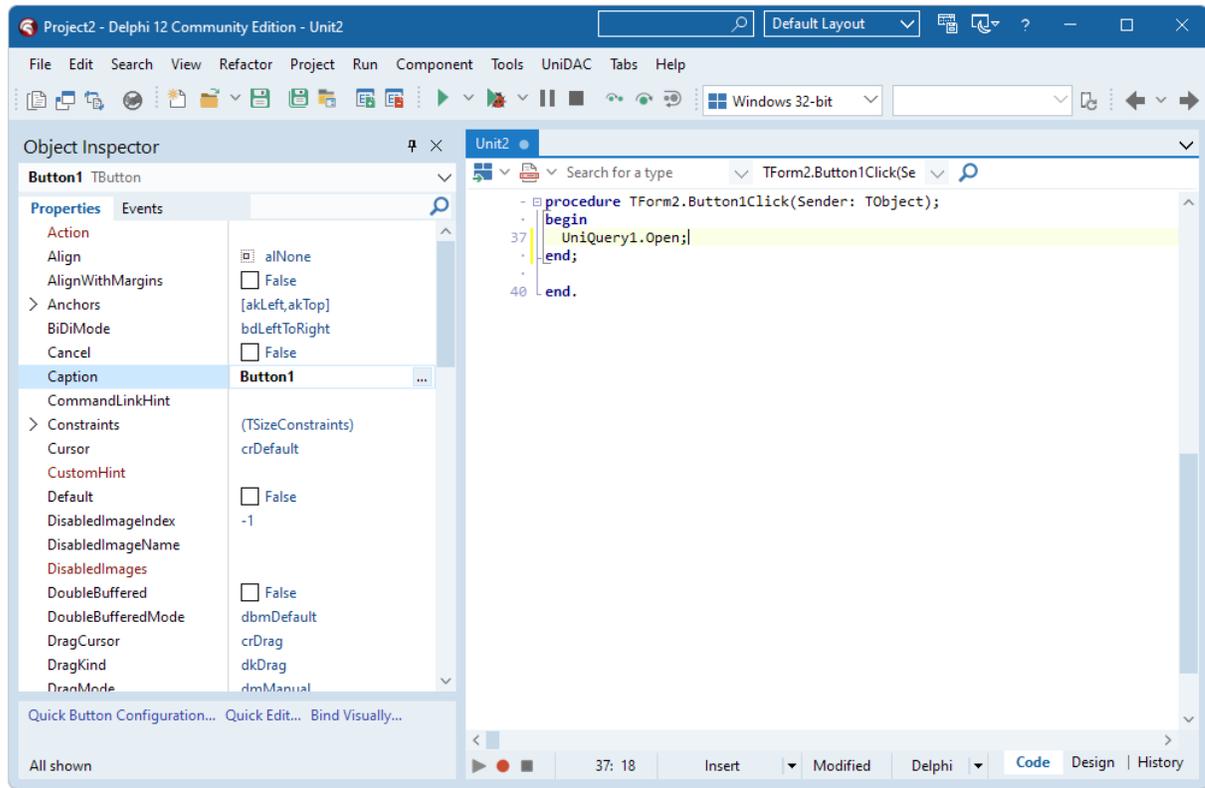
11. Double-click the **TUniConnection** component and fill out the fields on the **Connect** tab:

- **Provider** – Select **MySQL**.
- **Server** – Enter the host name or IP address of the MySQL server. If the SSH server and MySQL server are both running on the same machine, use *localhost*.
- **Port** – Specify the port of the MySQL server.
- **Username** – Enter the username for the account on the MySQL server.
- **Password** – Enter the password for the account on the MySQL server.
- **Database** – Select the database.

12. Click **Connect** to test the connection to the MySQL server, then click **OK** to close the dialog.

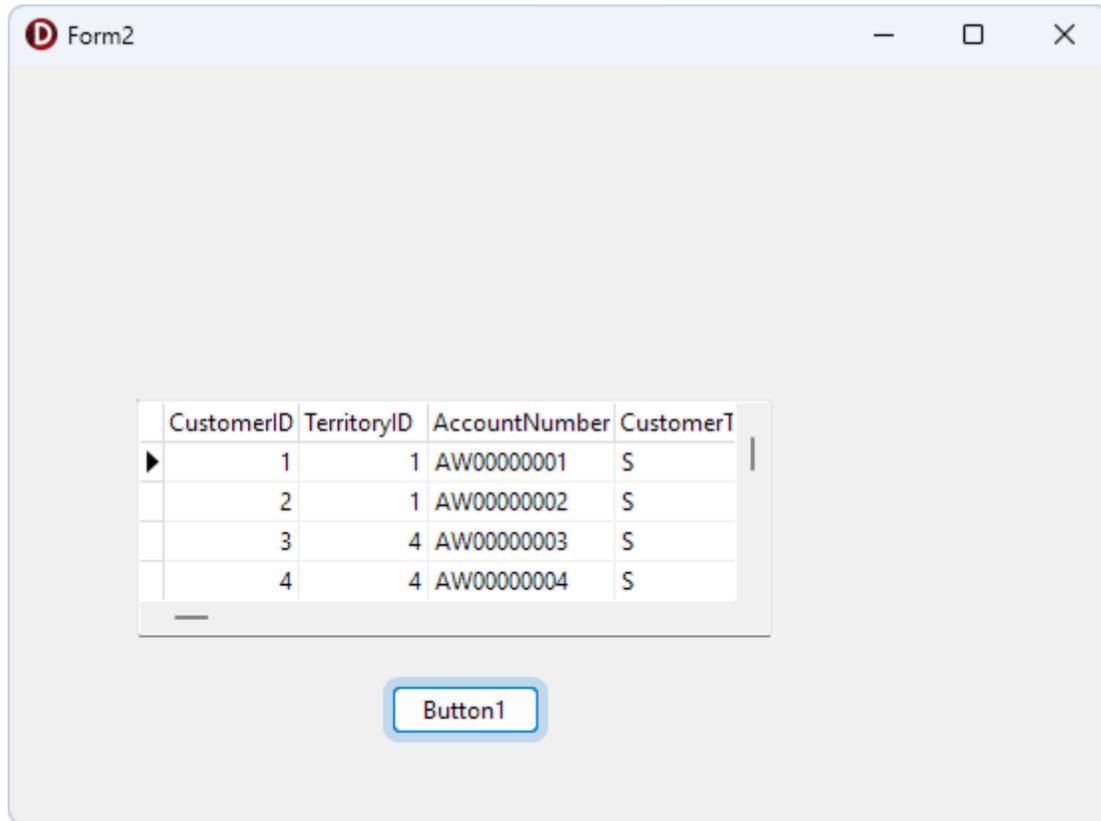


13. Double-click the **TButton** component and add code to call the **Open** method on the **UniQuery1** object to activate the dataset when the button is clicked.



14. Press **F9** to compile and run the application.

15. In the form that appears, click **Button1** to run the query. Data appears in the grid.



Connect Using OpenSSH or Any Other Third-Party SSH Tunneling Tools

Using SecureBridge and its components is not obligatory—you can use any other server that implements the SSH protocol.

The following steps describe a simple case of using OpenSSH for Windows. For a detailed description of each command, see the OpenSSH documentation.

1. [Download](#) OpenSSH for Windows.
2. Install an SSH server:
 - a. Choose a machine that will be used as the SSH server. It does not have to be the same machine as a MySQL server, but the communication channel between the SSH server and the MySQL server must be protected.
 - b. Using the Windows Control Panel, create a user (for example, SSH_user) and set a password for the user (for example, SSH_pass).

- c. Install OpenSSH. It is enough to install only the server components.
- d. Open the `openssh/bin` folder.
- e. Add `SSH_user` to the list of allowed users.

```
mkpasswd -l -u SSH_user >> ..\etc\passwd
```

- f. Use `mkgroup` to create a group permissions file.

```
mkgroup -l >> ..\etc\group
```

- g. Run the OpenSSH service.

```
net start opensshd
```

3. Install an SSH client:

- a. Choose a machine that will be used as the SSH client. It does not have to be the same machine where the MySQL client is running, but the communication channel between the SSH client and the MySQL client must be protected.
- b. Install OpenSSH to the SSH client. You do not need to install the server components.
- c. Run the SSH client.

```
ssh.exe -L <SSH_port>:<MySQL_server>:<MySQL_server_port> <SSH_user>@<SSH_server>
```

`<SSH_port>` – The port number of the SSH client that will be redirected to the corresponding port of the MySQL server.

`<MySQL_server>` – The name or IP address of the machine where the MySQL server is installed.

`<MySQL_server_port>` – The number of the MySQL server port, usually 3306.

`<SSH_user>` – The name of the user created at step 2.

`<SSH_server>` – The name or IP address of the machine where the SSH server was installed at step 2.

For example, `ssh.exe -L 3307:server:3306 SSH_user@192.168.0.116`.

At first launch, you'll be prompted to confirm the connection with the specified SSH server. Enter `yes` for confirmation.

At each launch of SSH, you must enter the password set at step 2.

4. Configure `TUniConnection`.

```
UniConnection1.Server := <SSH_client>;
UniConnection1.Port := <SSH_port>;
```

If the SSH client was installed on the same machine as the MySQL client, you can assign

```
localhost to UniConnection1.Server.
```

Note that in the provided steps, Windows checks the SSH_user authentication. For information about the methods of higher protection (key authentication, etc.), see the OpenSSH documentation.

For more information on using encrypted connections, see [MySQL Reference Manual](#).

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.10.4 Network Tunneling

In typical scenarios, a client connects to a server through a direct network path. However, modern security policies or complex network topologies often require the use of a proxy or traversal through a firewall. This article describes different ways to connect to a database server using UniDAC.

- [Direct connection](#)
- [Connection through an HTTP tunnel](#)
- [Connection through a proxy and an HTTP tunnel](#)
- [Additional information](#)

Direct connection

A direct connection to the server means that the server host is accessible from the client without any additional 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 to communicate with the server. Use it whenever possible.

The following example demonstrates how to configure a direct connection:

```
UniConnection := TUniConnection.Create(self);
UniConnection.ProviderName := '<provider_name>';
UniConnection.Server := 'localhost';
UniConnection.Port := 3306;
UniConnection.Username := '<user>';
UniConnection.Password := '<password>';
UniConnection.Connect;
```

Connection through an HTTP tunnel

Sometimes, client computers are shielded by a firewall that does not allow you to connect directly to the server on the specified port. If the firewall allows HTTP connections, you can use UniDAC and HTTP tunneling software to connect to the database server.

UniDAC supports HTTP tunneling based on a PHP script.

Consider a scenario where direct access to the database server port is blocked on a remote host, and only HTTP traffic on port 80 is permitted. In such cases, web script tunneling allows remote access to the database as if it were a direct connection.

You need to deploy the `tunnel.php` script included in the provider package on the web server. It allows access to the database server using HTTP tunneling. The script must be accessible through the HTTP protocol. You can check if it is accessible using a web browser. You can find the script in the HTTP subfolder of the installed provider folder, such as `%Program Files%\Devart\UniDac for Delphi X\HTTP\tunnel.php`. The only requirement is that the server must support PHP 5.

To connect to the database via HTTP tunneling, first configure `TUniConnection` as for a direct connection from the web server. Then, set the `Protocol` option to `mpHttp` and provide the following HTTP tunneling parameters:

Specific Option	Required	Description
<code>HttpUrl</code>	Yes	The URL of the tunneling PHP script. For example, if the script is in the server root, the URL can be <code>http://localhost/tunnel.php</code> .
<code>HttpUsername</code> , <code>HttpPassword</code>	No	A registered user credentials. Set these properties if access to the website folder with the script is available only for registered users authenticated with a username and password.

Connection through a proxy and an HTTP tunnel

Consider the previous case with one more complication.

The HTTP tunneling server is not directly accessible from the client computer. For example, the client address is 10.0.0.2, the server address is 192.168.0.10, and the database server is listening on port 3307. Since the client and the server are on different networks, the client can only access the server through a proxy located at 10.0.0.1 and listening on port 808. In this case, in addition to the HTTP-specific options, you need to set the proxy-specific options.

```
UniConnection := TUniConnection.Create(self);
UniConnection.ProviderName := '<provider_name>';
UniConnection.Server := '192.168.0.10';
UniConnection.Port := 3307;
UniConnection.Username := '<user>';
UniConnection.Password := '<password>';
UniConnection.SpecificOptions.Values['Protocol'] := 'mpHttp';
UniConnection.SpecificOptions.Values['HttpUrl'] := 'http://server/tunnel.php';
UniConnection.SpecificOptions.Values['ProxyHostname'] := '10.0.0.1';
UniConnection.SpecificOptions.Values['ProxyPort'] := '808';
UniConnection.SpecificOptions.Values['ProxyUsername'] := '<proxy_user>';
UniConnection.SpecificOptions.Values['ProxyPassword'] := '<proxy_password>';
UniConnection.Connect;
```

Note that when proxy-specific options are configured, UniDAC will automatically route the connection through the specified proxy server.

Additional information

Remember that traffic tunneling or encryption increases CPU usage and network load. Direct connections are recommended whenever possible.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.11 UniDAC and NexusDB

This article provides a brief overview of the NexusDB data access provider for UniDAC used to establish a connection to NexusDB databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [NexusDB-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)

- [TUniLoader](#)
- [TUniDump](#)
- [Data Type Mapping](#)

Overview

The main features of the NexusDB data access provider are:

- High performance
- Easy deployment
- Comprehensive support for the latest versions of NexusDB server

Both [Professional and Standard Editions](#) of UniDAC include the NexusDB provider. Express Edition of UniDAC does not include the NexusDB provider.

NexusDB provider is supplied with source code.

Compatibility

To learn about NexusDB compatibility, refer to the [Compatibility](#) section.

Requirements

You should have installed NexusDB components for corresponding IDE. NexusDB provider uses the following NexusDB libraries: NexusDBXXXdbXX, NexusDBXXXsdXX, NexusDBXXXIIXX, NexusDBXXXsrXX, NexusDBXXXptXX, NexusDBXXXtwXX, NexusDBXXXsqXX, NexusDBXXXseXX, NexusDBXXXstXX, NexusDBXXXreXX.

Before using the NexusDB provider, you have to rebuild and reinstall its provider package.

You can find the detailed steps describing the installation of the package in the UniDAC_Install_Dir\Source\NexusDBProvider\Readme.html file, where UniDAC_Install_Dir is a directory where you installed UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. If the option is set to True, you must deploy the package libraries listed in [Deployment](#) and additionally the following NexusDB libraries: NexusDBXXXdbXX, NexusDBXXXsdXX, NexusDBXXXIIXX, NexusDBXXXsrXX, NexusDBXXXptXX, NexusDBXXXtwXX, NexusDBXXXsqXX,

NexusDBXXXseXX, NexusDBXXXstXX, NexusDBXXXreXX.

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

NexusDB-specific options

Though UniDAC is components that provide unified interface to work with different database servers, it also lets you tune behaviour for each server individually. For thin setup of a certain database server, UniDAC provides server-specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a string list.

To assign a value to a specific option, use the following syntax:

```
ComponentName.SpecificOptions.Values['OptionName'] := 'OptionValue';
```

For example:

```
UniConnection.SpecificOptions.Values['CommandTimeout'] := '10';
UniQuery.SpecificOptions.Values['FetchAll'] := 'True';
```

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
CommandTimeout	Specifies the elapsed time in seconds before an attempt to execute a command is considered unsuccessful. The default value is 15.
ConnectionTimeout	Specifies the amount of time in seconds that can be expired before an attempt to make a connection is considered unsuccessful.
DatabaseReadOnly	If True, no writing is required, allows for sharing databases between servers.
HeartbeatInterval	Use the HeartbeatInterval option to specify how often the client will send a heartbeat message to the server. The default value is 10.
LostConnectionTime out	Specifies the amount of time in seconds that must expire before a connection is considered lost. The default value is 10.
WatchdogInterval	Use the WatchdogInterval option to specify how often the client will check all connections. The default value is 10.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ReadOnly	Use the ReadOnly option to prevent users from modifying data in the database. The default value is False.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
CursorUpdate	Specifies what way data updates reflect on database when modifying dataset by using server NexusDB cursors (the ServerCursor option is set to True). If True, all dataset modifications pass to database by server cursors. It increases performance but doesn't allow to use procedures or enhanced queries for additional data changes. If False, all dataset updates pass to server by SQL statements generated automatically or specified in SQLUpdate, SQLInsert or SQLDelete. The default value is True.
FetchAll	When set to True, all records of the query are requested from the database server when dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important. When the FetchAll property is False, the first call to Locate and LocateEx methods may take a lot of time to retrieve additional records to the client side.
ReadOnly	Use the ReadOnly option 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.
ServerCursor	By default, ServerCursor is False, meaning that NexusDB provider reads data to the own memory when dataset is opened. NexusDB provider performs all database operations using SQL statements generated automatically or specified in SQLUpdate, SQLInsert or SQLDelete. If True, then NexusDB provider calls server NexusDB cursor for resultset record access and then reads data from it. So, stored data aren't duplicated that allows you to decrease memory charges. Data to the server can be written using server cursor or SQL queries in dependence of CursorUpdate option. So the TCustomDADataset.FetchRows, FetchAll, CachedUpdates properties don't have any influence on such cursors and only the CursorUpdate option does.

TUniScript

The TUniScript component has no NexusDB-specific options.

TUniLoader

Option name	Description
DirectLoad	If True, all inserted data pass to database by server NexusDB cursors. If False, all inserted data pass to server by SQL statements. The default value is True.

TUniDump

The TUniDump component has no NexusDB-specific options.

Data Type Mapping

The following table lists the constants for mapping NexusDB data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
nxBoolean	Maps Boolean to Delphi data types.
nxChar	Maps Char to Delphi data types.
nxWideChar	Maps wideChar to Delphi data types.
nxByte	Maps Byte to Delphi data types.
nxWord16	Maps word16 to Delphi data types.
nxWord32	Maps word32 to Delphi data types.
nxInt8	Maps Int8 to Delphi data types.
nxInt16	Maps Int16 to Delphi data types.
nxInt32	Maps Int32 to Delphi data types.
nxInt64	Maps Int64 to Delphi data types.
nxAutoInc	Maps AutoInc to Delphi data types.
nxSingle	Maps Single to Delphi data types.
nxDouble	Maps Double to Delphi data types.
nxExtended	Maps Extended to Delphi data types.
nxCurrency	Maps Currency to Delphi data types.
nxDate	Maps Date to Delphi data types.
nxTime	Maps Time to Delphi data types.
nxDateTime	Maps DateTime to Delphi data types.
nxInterval	Maps Interval to Delphi data types.

nxBlob	Maps BLOB to Delphi data types.
nxBlobMemo	Maps BLOB Memo to Delphi data types.
nxBlobGraphic	Maps BLOB Graphic to Delphi data types.
nxByteArray	Maps Byte Array to Delphi data types.
nxShortString	Maps ShortString to Delphi data types.
nxNullString	Maps NullString to Delphi data types.
nxWideString	Maps WideString to Delphi data types.
nxRecRev	Maps Recrev to Delphi data types.
nxGuid	Maps GUID to Delphi data types.
nxBCD	Maps BCD to Delphi data types.
nxBlobWideMemo	Maps BLOB Wide Memo to Delphi data types.
nxFmtBCD	Maps FmtBCD to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.12 UniDAC and ODBC

This article provides a brief overview of the ODBC data access provider for UniDAC that allows ODBC connection to DBMSs from Delphi and Lazarus if a corresponding driver exists. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [ODBC-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [Data Type Mapping](#)

Overview

Main features of the ODBC data access provider are:

- High performance
- Easy deployment
- Support for any DBMS that comes with ODBC driver

The full list of the ODBC provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the ODBC provider. Express Edition of UniDAC does not include the ODBC provider.

Compatibility

ODBC provider supports ODBC 3.x.

Requirements

Applications that use the ODBC provider require ODBC to be installed on the client computer. In the current versions of Microsoft Windows, since Windows 2000, ODBC is already included as a standard package.

To use the ODBC provider with specific DBMS, ODBC driver for the required DBMS must be installed.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

ODBC-specific options

TUniConnection

Option name	Description
-------------	-------------

ColumnWiseBinding	<p>If True, enables Column-Wise Binding mode. The default value is False.</p> <p>Note: Row-Wise Binding mode is enabled by default. However, some ODBC drivers don't support this mode. In such case, set the ColumnWiseBinding option to True.</p>
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
DetectFieldsOnPrepare	<p>Detects fields on the Prepare command execution.</p> <p>The default value is True.</p> <p>Note: This functionality is not supported in some ODBC drivers.</p>
DriverManager	Specifies the dynamic-link library (DLL) that loads ODBC database drivers on behalf of an application.
DSNType	<p>The type of the data source name (DSN) assigned to the Server property.</p> <p>ntAuto The default value. Automatically identify the type of DSN.</p> <p>ntName User DSN or System DSN (registered with ODBC Administrator).</p> <p>ntFile File DSN (a .DSN file containing the data source information).</p> <p>ntConnectionString ODBC connection string.</p>
LongVarBinaryAsBlob	Specifies that all binary byte strings represented by the LONGVARBINARY type will be retrieved as BLOB fields and handled by the TBlobField class. The default value is True.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideStrings, and TStringField is replaced by TWideStringField.
UuidWithBraces	Specifies whether the values of UUID fields are returned with braces. The default value is True.
VarBinaryAsBlob	If set to True, all binary byte strings represented by the VARBINARY type will be retrieved as BLOB fields and handled by the TBlobField class. The default value is False.

TUniSQL

Option name	Description
-------------	-------------

CommandTimeout	The time to wait for a statement to be executed.
----------------	--

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	Used to store all non-BLOB fields as string. The default value is False.
UnknownAsString	Used to map fields of unknown data types to TStringField (TWideStringField). The default value is False. If False, fields of unknown data types (for example the ifnull function result) are mapped to TMemofield or TWideMemofield depending on the value of the UseUnicode option. Memo is used because maximum length of values from such fields is unknown. If True, fields of unknown data types are mapped to TStringField or TWideStringField depending on the value of the UseUnicode option. Size of fields is set to 8192. Values larger than this size are truncated.

TUniScript

The TUniScript component has no ODBC-specific options.

TUniLoader

The TUniLoader component has no ODBC-specific options.

TUniDump

The TUniDump component has no ODBC-specific options.

Data Type Mapping

The following table lists the constants for mapping ODBC data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
odbcChar	Maps SQL_CHAR to Delphi data types.
odbcWideChar	Maps SQL_WCHAR to Delphi data types.
odbcVarChar	Maps SQL_VARCHAR to Delphi data types.
odbcWideVarChar	Maps SQL_WVARCHAR to Delphi data types.
odbcLongVarChar	Maps SQL_LONGVARCHAR to Delphi data types.
odbcWideLongVarChar	Maps SQL_WLONGVARCHAR to Delphi data types.
odbcBit	Maps SQL_BIT to Delphi data types.
odbcTinyInt	Maps SQL_TINYINT to Delphi data types.
odbcUTinyInt	Maps SQL_TINYINT UNSIGNED to Delphi data types.
odbcSmallInt	Maps SQL_SMALLINT to Delphi data types.
odbcUSmallInt	Maps SQL_SMALLINT UNSIGNED to Delphi data types.
odbcInteger	Maps SQL_INTEGER to Delphi data types.
odbcUInteger	Maps SQL_INTEGER UNSIGNED to Delphi data types.
odbcBigInt	Maps SQL_BIGINT to Delphi data types.
odbcUBigInt	Maps SQL_BIGINT UNSIGNED to Delphi data types.
odbcReal	Maps SQL_REAL to Delphi data types.
odbcFloat	Maps SQL_FLOAT to Delphi data types.
odbcDouble	Maps SQL_DOUBLE to Delphi data types.
odbcDecimal	Maps SQL_DECIMAL to Delphi data types.
odbcNumeric	Maps SQL_NUMERIC to Delphi data types.
odbcDate	Maps SQL_TYPE_DATE to Delphi data types.
odbcTime	Maps SQL_TYPE_TIME to Delphi data types.
odbcTimeStamp	Maps SQL_TYPE_TIMESTAMP to Delphi data types.
odbcBinary	Maps SQL_BINARY to Delphi data types.
odbcVarBinary	Maps SQL_VARBINARY to Delphi data types.
odbcLongVarBinary	Maps SQL_LONGVARBINARY to Delphi data types.
odbcBlob	Maps SQL_BLOB to Delphi data types.
odbcClob	Maps SQL_CLOB to Delphi data types.
odbcXml	Maps SQL_XML to Delphi data types.
odbcVariant	Maps SQL_VARIANT to Delphi data types.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.13 UniDAC and Oracle

5.1.13.1 Oracle Provider

This article provides a brief overview of the Oracle data access provider for UniDAC used to establish a connection to Oracle databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Oracle-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [Oracle-specific notes](#)
- [Connecting in Direct mode](#)
- [Data Type Mapping](#)

Overview

Oracle data access provider is based on the Oracle Data Access Components ([ODAC](#)) library, which is one of the best known Delphi data access solutions for Oracle. The main features of Oracle data access provider are:

- Direct access to the server without Oracle client (OCI)
- High performance

- Easy deployment
- Full support for the latest versions of Oracle server

The full list of Oracle provider features can be found on the UniDAC [Features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Oracle provider. For Express Edition of UniDAC, the Oracle provider can be installed with ODAC.

Compatibility

To learn about Oracle database server compatibility, see the [Compatibility](#) section.

Requirements

If your application is using the Direct mode, you don't need to install any additional software on the client machine. In the Client mode, you need to install the Oracle client.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Oracle-specific options

In addition to providing a unified interface to work with different database server, it also allows you to tune application behavior for each server individually. UniDAC provides server-specific options for fine-tuning specific database servers. These options can be applied to such components as `TUniConnection`, `TUniQuery`, `TUniTable`, `TUniStoredProc`, `TUniSQL`, and `TUniScript` using the `SpecificOptions` property, which is a string list. You can use the following syntax to assign a value to `SpecificOptions`:

```
UniConnection.SpecificOptions.Values['CharLength'] := '1';
```

Below you will find the description of allowed options grouped by components.

TUniConnection

Option	Description
CharLength	The size of characters in bytes for national language support. The range of allowed values is between 0 and 6. The default value is 0 — the actual character length is determined by an Oracle server.
Charset	The character set for the character data transferred between the client and server. Supported with Oracle 8 client only.
ClientIdentifier	The client identifier in the session. The client identifier can be set in the session handle at any time in the session. Then, on the next request to the server, the information is propagated and stored in the server session. The first character of the ClientIdentifier must not be ':'. If it is, an exception will be raised. This property has no effect if you use the version of the server earlier than Oracle 9.
ConnectionTimeout	The time to wait for a connection to open before raising an exception. Works only when Direct is set to True. The default value is 0.
ConnectMode	The system privilege for the user who connects to the server. cmNormal The default value. Connect as a normal user. cmSysOper Connect with the SYSOPER privilege. cmSysDBA Connect with the SYSDBA privilege. cmSysASM Connect with the SYSASM privilege. cmSysBackup Connect with the SysBackup privilege. cmSysDG Connect with the SysDG privilege. cmSysKM Connect with the SYSKM privilege. The privilege must be granted to the user before connecting to the server. ConnectMode is not supported for OCI 7.
DateFormat	The default date format used when Oracle makes conversions from internal date format into string values and vice versa. An example of a valid expression is MM/DD/YYYY.
DateLanguage	The default language used when Oracle parses internal date format into strings and vice versa. Examples of valid expressions include French, German, etc.
Direct	Enables the Direct mode. When True, connection is performed directly over TCP/IP, without involving Oracle client software. When False, the Oracle provider connects in the Client mode. The default value is False.
EnableIntegers	Represents Oracle NUMBER fields with precision less than 10. When True, fields are represented as TIntegerField. When

	False, fields are represented as TFloatField. The default value is True.
EnableLargeint	Represents Oracle NUMBER fields with precision more than 9 and less than 18. When True, fields are represented as TIntegerField. When False, fields are represented as TFloatField. The default value is False.
HomeName	Specifies which Oracle client to use when two or more Oracle clients are present on the machine. The Oracle provider searches for available homes in the HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE registry folder. When the option is set to an empty string, the provider uses the first directory from the list of homes found in the PATH environment variable as the default Oracle home.
HttpPassword	The password for the password-protected directory that contains the HTTP tunneling script.
HttpTrustServerCertificate	Specifies whether to verify the server certificate during an SSL handshake. When True, ODAC bypasses walking the certificate chain to verify the certificate. The default value is False.
HttpUrl	The URL of the PHP script for HTTP tunneling.
HttpUsername	The username for the password-protected directory that contains the HTTP tunneling script.
IPVersion	The 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 property is set to ivIPBoth, a connection attempt is made via IPv6 if it is enabled in the operating system. If the connection attempt fails, a new connection attempt is made via IPv4.
OptimizerMode	The default optimizer mode for the connection. omDefault The session optimizer mode remains unchanged. omFirstRowsN The optimizer chooses the execution plan that returns the first N rows as quickly as possible. If you use Oracle 9.0 or earlier, these options have the same effect as omFirstRows. omFirstRows This mode is retained for backward compatibility and plan stability. It optimizes for the best execution plan to return the first row as soon as possible. omAllRows

	<p>The optimizer explicitly chooses the cost-based approach to optimize a statement block with a goal of best throughput (that is, minimum total resource consumption).</p> <p><code>omChoose</code></p> <p>The optimizer chooses between the rule-based and cost-based approaches for an SQL statement. The choice of the optimizer depends on the presence of statistics for the tables accessed by the statement: if the data dictionary has statistics for at least one of the tables, the optimizer uses the cost-based approach and optimizes with the goal of the best throughput. Otherwise, it uses the rule-based approach.</p> <p><code>omRule</code></p> <p>The optimizer chooses rule-based optimization (RBO). Any other value causes the optimizer to choose cost-based optimization (CBO). The rule-based optimizer is the archaic optimizer mode from the earliest releases of Oracle Database.</p>
<code>PoolingType</code>	<p>The connection pooling implementation.</p> <p><code>optLocal</code></p> <p>The default value. Our own connection pooling implementation.</p> <p><code>optOCI</code></p> <p>OCI connection pooling.</p> <p><code>optMTS</code></p> <p>Shared server (MTS) connection pooling.</p>
<code>PrecisionBCD</code>	<p>Represents Oracle NUMBER fields as <code>TBCDField</code> when their precision and scale are less than or equal to the precision and scale in <code>PrecisionBCD</code>, specified as two comma-separated values (BCD precision and scale). <code>PrecisionBCD</code> cannot be greater than 14,4. The default value is 14,4.</p>
<code>PrecisionFloat</code>	<p>Represents Oracle NUMBER fields as <code>TFloatField</code> when their precision is less than or equal to the precision in <code>PrecisionFloat</code>. The default value is 0.</p>
<code>PrecisionFMTBCD</code>	<p>Represents Oracle NUMBER fields as <code>TFMTBCDField</code> when their precision and scale are less than or equal to the precision and scale in <code>PrecisionFMTBCD</code>, specified as two comma-separated values (FMTBCD precision and scale). The default value is 39,39.</p>
<code>PrecisionInteger</code>	<p>Represents Oracle NUMBER fields as <code>TIntegerField</code> when their precision is less than or equal to the precision in <code>PrecisionInteger</code>. The default value is 9.</p>
<code>PrecisionLargeint</code>	<p>Represents Oracle NUMBER fields as <code>TLargeintField</code> when their precision is less than or equal to the precision in <code>PrecisionLargeint</code>. The default value is 18.</p>
<code>PrecisionSmallint</code>	<p>Represents Oracle NUMBER fields as <code>TSmallintField</code> when their precision is less than or equal to the precision in</p>

	PrecisionSmallint. The default value is 4.
ProxyHostname	The proxy hostname or IP address.
ProxyPassword	The proxy password.
ProxyPort	The proxy port.
ProxyUsername	The proxy username.
Schema	Changes the current schema of the session to the specified schema. This option offers a convenient way to perform operations on objects in a schema other than that of the current user, without having to qualify the objects with the schema name. It changes the current schema, but it does not change the session user or the current user, nor does it give you any additional system or object privileges for the session. If <code>TUniConnection.Connected</code> is <code>True</code> , read this property to get the name of the current schema.
SSL Key	The private client key.
SSLCACert	The server CA certificate.
SSLCert	The client certificate.
SSLCipherList	The list of allowed SSL ciphers.
SSLServerCertDN	The server's distinguished name (DN) to enable server DN matching. It checks whether the server is genuine by matching the server's global database name against the DN from the server certificate.
StatementCache	Enables statement caching. The default value is <code>False</code> .
StatementCacheSize	The size of statement cache. The default value is 20.
ThreadSafety	Enables the use of OCI in a multithreaded environment. This option must be set to <code>True</code> before any non-blocking fetch of rows or SQL statement execution takes place. The default value is <code>True</code> .
UnicodeEnvironment	Enables the use of OCI Unicode Environment. When <code>True</code> , Unicode characters can be used in SQL statements. Set the option to <code>False</code> if you encounter some issues with Unicode Environment. The default value is <code>False</code> .
UseOCI7	Forces <code>TUniConnection</code> to use the OCI 7 call style only. The default value is <code>False</code> .
UseUnicode	Enables Unicode support. The option affects character data fetched from the server. When <code>True</code> , all character data is stored as <code>WideString</code> , and <code>TStringField</code> is replaced with <code>TWideStringField</code> . Supported starting with Oracle 8. The default value is <code>False</code> .

TUniSQL

Option	Description
CommandTimeout	The wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement. The wait time is specified in seconds. The default value is 0. The value of 0 indicates there are no time limits (an attempt to execute a statement will wait indefinitely).
NonBlocking	Executes a SQL statement in a separate thread. The default value is False.
StatementCache	Enables statement caching. The default value is False.
TemporaryLobUpdate	Enables the use of a temporary LOB to write input and input/output LOB parameter into database when executing dataset's SQL statements. The default value is True.

TUniQuery, TUniTable, TUniStoredProc

Option	Description
AutoClose	Closes the OCI cursor after fetching all rows. The option allows to reduce the number of opened cursors on the server. The default value is False.
CacheLobs	Allocates a local memory buffer to hold a copy of the LOB content. When False, it is highly recommended to set the DeferredLobRead option to True. Otherwise, LOB values are fetched to the dataset, which may result in performance loss. The default value is True.
CommandTimeout	The wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement. The wait time is specified in seconds. The default value is 0. The default value 0 indicates there are no time limits (an attempt to execute a command will wait indefinitely).
DeferredLobRead	When True, all Oracle 8 LOB values are only fetched when they are explicitly requested. When False, an entire record set, including LOB values, is returned when a dataset is opened. The CacheLobs option specifies whether LOB values are cached locally to be reused later. The default value is False.
DynamicReadThreshold	Allows reading data of the VARCHAR2, NVARCHAR2, and RAW types. If a column size is greater than or equal to the value specified in this option, data will be fetched using the OCI_DYNAMIC_FETCH mode. While this mode can significantly reduce memory consumption during data retrieval, it may also negatively impact performance. The default value is 0, which disables reading data in the OCI_DYNAMIC_FETCH mode.
ExtendedFieldsInfo	Performs an additional query to get information about the returned fields and tables they belong to. This helps to generate

	correct updating SQL statements but may result in performance decrease. The default value is <code>False</code> .
<code>FetchAll</code>	When <code>True</code> , a query requests all records from a database server when opening a dataset. When <code>False</code> , records are retrieved when a data-aware component or an application requests them. The default value is <code>False</code> .
<code>FieldsAsString</code>	Treats all non-BLOB fields as strings. The default value is <code>False</code> .
<code>HideRowId</code>	Hides the <code>RowId</code> service field (the <code>Visible</code> property is set to <code>False</code>). The default value is <code>True</code> .
<code>KeySequence</code>	The name of a sequence that will be used to fill in a key field after a new record is inserted or posted to the database.
<code>NonBlocking</code>	Executes a SQL statement in a separate thread. The default value is <code>False</code> .
<code>PrefetchLobSize</code>	Retrieves the LOB length and the chunk size, as well as the beginning of the LOB data and the locator during a regular fetch. The option specifies the size of LOB data to be prefetched. If the total LOB size is less than or equal to the value of <code>PrefetchLobSize</code> , then all LOB data is fetched without additional round trips during a regular fetch, which may significantly improve performance. The default value is <code>0</code> Note: LOB data prefetching is available in Oracle 11 and higher.
<code>PrefetchRows</code>	The number of rows to be prefetched during the execution of a query. Setting the property to a value greater than <code>0</code> reduces the server round-trip count and increases the performance of the application. The default value is <code>0</code> — the number of prefetched rows is determined automatically. To disable row prefetching, set the property to <code>-1</code> . Note: Some queries can return invalid rows count when prefetching is enabled — for example, <code>SELECT * FROM DUAL CONNECT BY LEVEL <= 5</code> returns 1 row when prefetching is enabled, and 5 rows when it is disabled.
<code>ProcNamedParams</code>	Specifies a notation method of passing parameter values to the stored PL/SQL object. By default, positional notation is used. To enable named notation, set the option to <code>True</code> . Named notation allows passing parameter values in any order regardless of the position.
<code>RawAsString</code>	Treats all RAW fields as hexadecimal strings. The default value is <code>False</code> .
<code>ScrollableCursor</code>	When <code>True</code> , <code>TUniDataSet</code> does not cache data on the client side, but uses a scrollable server cursor (available since Oracle 9 only). This option can be used to reduce memory usage, since dataset stores only the current fetched block. Unlike the <code>UniDirectional</code> option, <code>ScrollableCursor</code> allows bidirectional dataset navigation. Note: Scrollable cursor is read-only by nature.

SequenceMode	<p>Specifies the method to be used internally to generate a sequenced field.</p> <p>smInsert New record is inserted into the dataset with the first key field populated with a sequenced value. Application may modify this field before posting the record to the database.</p> <p>smPost The default value. Database server populates key field with a sequenced value when application posts the record to the database. Any value put into key field before post, is overwritten.</p>
StatementCache	Enables statement caching. The default value is False.
TemporaryLobUpdate	Enables the use of a temporary LOB to write input and input/output LOB parameter into database when executing dataset's SQL statements. The default value is True

TUniScript

The `TUniScript` component has no Oracle-specific options.

TUniLoader

Option	Description
DirectPath	When True, data is loaded using the Oracle Direct Path Load interface. When False, data is loaded by executing an INSERT statement. The default value is True.
QuoteNames	Quotes all database object names in automatically generated SQL statements, such as UPDATE statements. The default value is False.

`TUniLoader` has the following limitations when Oracle Direct Path Load is used:

- triggers are not supported
- check constraints are not supported
- referential integrity constraints are not supported
- clustered tables are not supported
- loading of remote objects is not supported
- user-defined types are not supported
- LOBs must be specified after all scalar columns
- LONGs must be specified last

- You cannot use TUniLoader in a threaded OCI environment with Oracle client 8.17 or lower.

TUniDump

The `TUniDump` component has no Oracle-specific options.

Oracle-specific notes

This section describes how to connect to Oracle in the Direct mode.

Connecting in Direct mode

By default the Oracle provider uses the Oracle Call Interface (OCI) to connect to the Oracle database server. This is referred to as connecting in the Client mode, and is the usual way to develop Oracle applications with a third-generation language. All OCI routines are stored in external libraries, so the executables for applications that work through OCI are small. However, working through OCI requires the Oracle client software to be installed on client machines. It is rather inconvenient and causes additional installation and administration expenses. Furthermore, there are some situations where the installation of the Oracle client is not advisable or may be even impossible—for example, if you deploy an application to remote machines that are not overseen by a proficient system administrator.

To overcome these challenges, the Oracle provider includes an option to connect to Oracle directly over the network using the TCP/IP protocol. This is referred to as connecting in the *Direct mode*. Connecting in the Direct mode does not require Oracle client software to be installed on client machines. The only requirement for running an application that uses ODAC in the Direct mode, is that the operating system must support the TCP/IP protocol.

To connect to Oracle server in the Direct mode, set the `Direct` property of your `TUniConnection` instance to `True`, and the `Server` property to a string that contains the host address of the database server, port number, and Oracle Service Name (SN) or Oracle System Identifier (SID) in the following format:

if you connect to Oracle using Service Name:

```
Host:Port/ServiceName
```

or

```
Host:Port:sn=ServiceName
```

if you connect to Oracle using SID that is the same as Service Name:

Host:Port:SID

if you connect to Oracle using SID that is different from Service Name:

Host:Port:sid=SID

Host is the server's IP address or DNS name.

Port is the port number that the server listens to.

SID is a system identifier that specifies the name of an Oracle database instance.

ServiceName is a system alias to an Oracle database instance (or multiple instances).

Note that the syntax used to set up the `Server` property in the Direct mode is different from the Client mode. In the Client mode, this property must be set to the TNS name of the Oracle server.

Note that if `sid=` or `sn=` is not defined, the connection will be established using SID. If SID and Service Name are the same, then either of them can be used to set the

`TUniConnection.Server` property.

An example below illustrates the connection to Oracle in the Direct mode. The IP address of the Oracle server is `205.227.44.44`, the port number is `1521` (the most commonly used port for Oracle), and the SID is `orcl` (standard Oracle SID):

```
var
  UniConnection: TUniConnection;
...
UniConnection.Username := 'Scott';
UniConnection.Password := 'tiger';
UniConnection.Server := '205.227.44.44:1521:orcl';
UniConnection.SpecificOptions.Values['Direct'] := 'True';
UniConnection.Connect;
```

connecting to Oracle with Service Name:

```
...
UniConnection.Server := '205.227.44.44:1521/orcl';
...
or
...
UniConnection.Server := '205.227.44.44:1521:sn=orcl';
...
```

connecting to Oracle with SID:

```
...
UniConnection.Server := '205.227.44.44:1521:orcl';
...
or
...
```

```
UniConnection.Server := '205.227.44.44:1521:sid=orcl';  
...
```

This is all you need to do to enable the Direct mode in your application. You do not have to rewrite other parts of your code.

To return to the OCI mode, set `UniConnection.SpecificOptions.Values['Direct']` to `'False'` and `UniConnection.Server` to the TNS name of your server.

You can connect to Multi-Threaded Server using the Direct mode. The server must be configured to use a specific port and the TTC protocol. This can help you avoid firewall conflicts.

Note: The Direct mode is available in UniDAC Professional Edition and UniDAC Trial. An attempt to set the `UniConnection.SpecificOptions.Values['Direct']` property to `'True'` in UniDAC Standard Edition will generate a *"Feature is not supported"* error.

Client Mode vs. Direct Mode

Applications that use the Client mode and those that use the Direct mode have similar performance and file size. In terms of security, using the Direct mode is the same as using Oracle Client without Oracle Advanced Security. In the Direct mode, ODAC uses DES authentication and does not support Oracle Advanced Security.

Advantages of the Direct mode:

- No need to install and administer Oracle client.
- Reduced system requirements.

Limitations of the Direct mode:

- only TCP/IP connections are supported;
- some issues may occur when using firewalls;
- NLS conversion on the client side is not supported;
- Transparent Application Failover is not supported;
- statement caching is not supported;
- OS Authentication and password changing are not supported;
- Oracle Advanced Security is not supported;

- stable operation of multithreaded applications is not guaranteed; it is highly recommended that you use a separate TUniConnection component for each thread when multiple threads use UniDAC.

A connection in the Direct mode is managed transparently by an instance of TUniConnection, and you can easily switch back to OCI in the Client mode at any time if the above limitations become critical to you.

We tested the Direct mode with all versions of Oracle server for Windows in a local network, but we do not guarantee compatibility with all Oracle servers on other platforms in different networks.

Data Type Mapping

The following table lists the constants for mapping Oracle data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
oraAnyData	Maps ANYDATA to Delphi data types.
oraBFile	Maps BFILE to Delphi data types.
oraBinaryDouble	Maps BINARY_DOUBLE to Delphi data types.
oraBinaryFloat	Maps BINARY_FLOAT to Delphi data types.
oraBlob	Maps BLOB to Delphi data types.
oraCFile	Maps CFILE to Delphi data types.
oraChar	Maps CHAR to Delphi data types.
oraClob	Maps CLOB to Delphi data types.
oraCursor	Maps CURSOR to Delphi data types.
oraDate	Maps DATE to Delphi data types.
oraDoublePrecision	Maps DOUBLE PRECISION to Delphi data types.
oraFloat	Maps FLOAT to Delphi data types.
oraInteger	Maps INTEGER to Delphi data types.
oraIntervalDS	Maps INTERVAL DAY TO SECOND to Delphi data types.
oraIntervalYM	Maps INTERVAL YEAR TO MONTH to Delphi data types.
oraLabel	Maps MLSLABEL to Delphi data types.
oraLong	Maps LONG to Delphi data types.
oraLongRaw	Maps LONG RAW to Delphi data types.
oraNChar	Maps NCHAR to Delphi data types.
oraNClob	Maps NCLOB to Delphi data types.
oraNumber	Maps NUMBER to Delphi data types.

oraNvarchar2	Maps NVARCHAR2 to Delphi data types.
oraObject	Maps OBJECT to Delphi data types.
oraRaw	Maps RAW to Delphi data types.
oraReference	Maps REF to Delphi data types.
oraRowID	Maps ROWID to Delphi data types.
oraTimeStamp	Maps TIMESTAMP to Delphi data types.
oraTimeStampWithLocalTimeZone	Maps TIMESTAMP WITH LOCAL TIME ZONE to Delphi data types.
oraTimeStampWithTimeZone	Maps TIMESTAMP WITH TIME ZONE to Delphi data types.
oraUndefined	Maps UNDEFINED to Delphi data types.
oraUrowID	Maps UROWID to Delphi data types.
oraVarchar2	Maps VARCHAR2 to Delphi data types.
oraXML	Maps XML to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.13.2 Connecting via SSL

Secure Sockets Layer (SSL) is an industry-standard protocol for secure access to a remote machine over untrusted networks. It runs on top of TCP/IP to secure client-server communications by allowing SSL-enabled clients and servers to authenticate each other.

During server authentication, an SSL-enabled client application uses standard public-key cryptography techniques to verify the server's identity by checking that the server's certificate was issued by a trusted Certificate Authority (CA) and proves the ownership of the public key. SSL also supports client authentication, allowing the server to verify the client's identity.

After establishing an SSL connection, the client and server can exchange messages that are symmetrically encrypted using the shared secret key. SSL is the recommended method for establishing a secure connection to Oracle because it is easier to configure and performs better than SSH.

You can establish a secure connection to an Oracle server using [SecureBridge](#), a solution provided by Devart. It allows you to embed an SSL client into a Delphi or C++ Builder application.

This page demonstrates how to connect a client application to Oracle using UniDAC with SSL as the encryption method.

- [Connect using an Oracle wallet](#)
- [Connect using SSL certificates and keys](#)

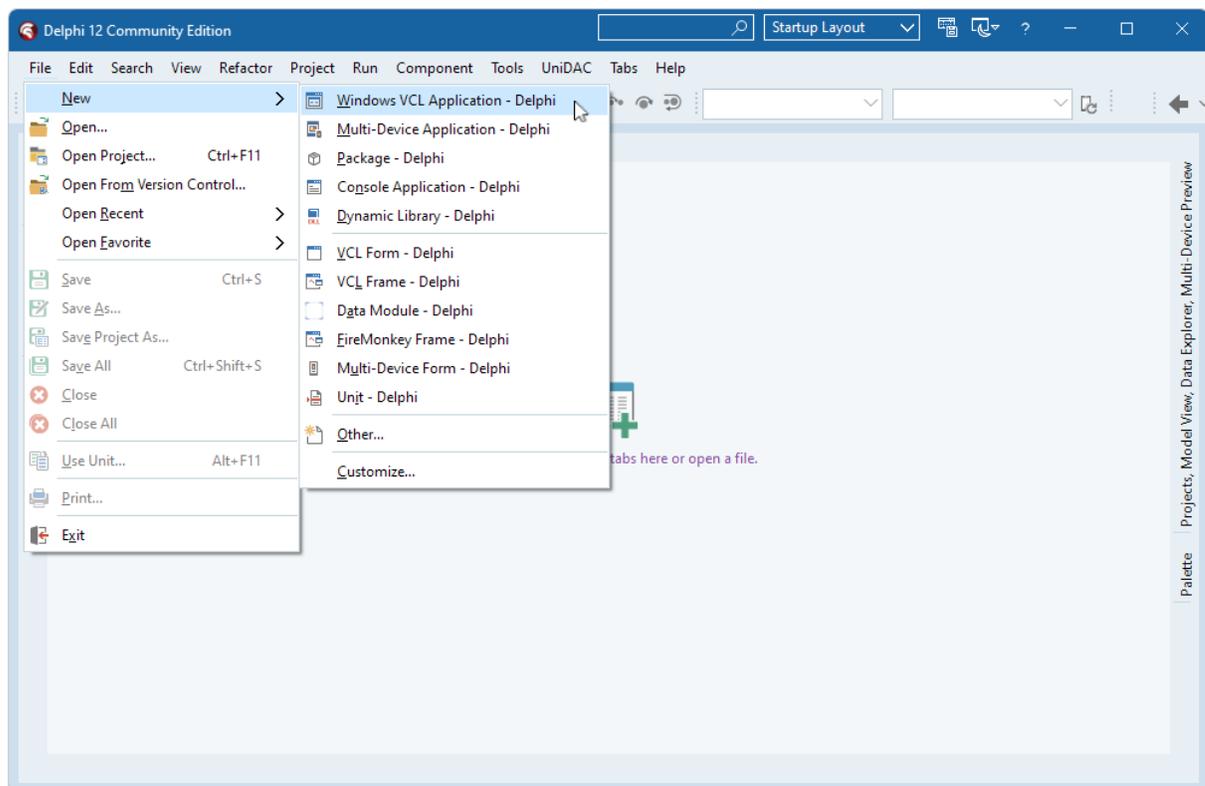
Connect using an Oracle wallet

Prerequisites:

- Install SecureBridge.
- Install the TCRSSLIOMHandler component in RAD Studio to bind UniDAC with SecureBridge.
For instructions, see [Readme.html](#)—by default, it is located in `C:\Users\\Documents\Devart\UniDAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge`.

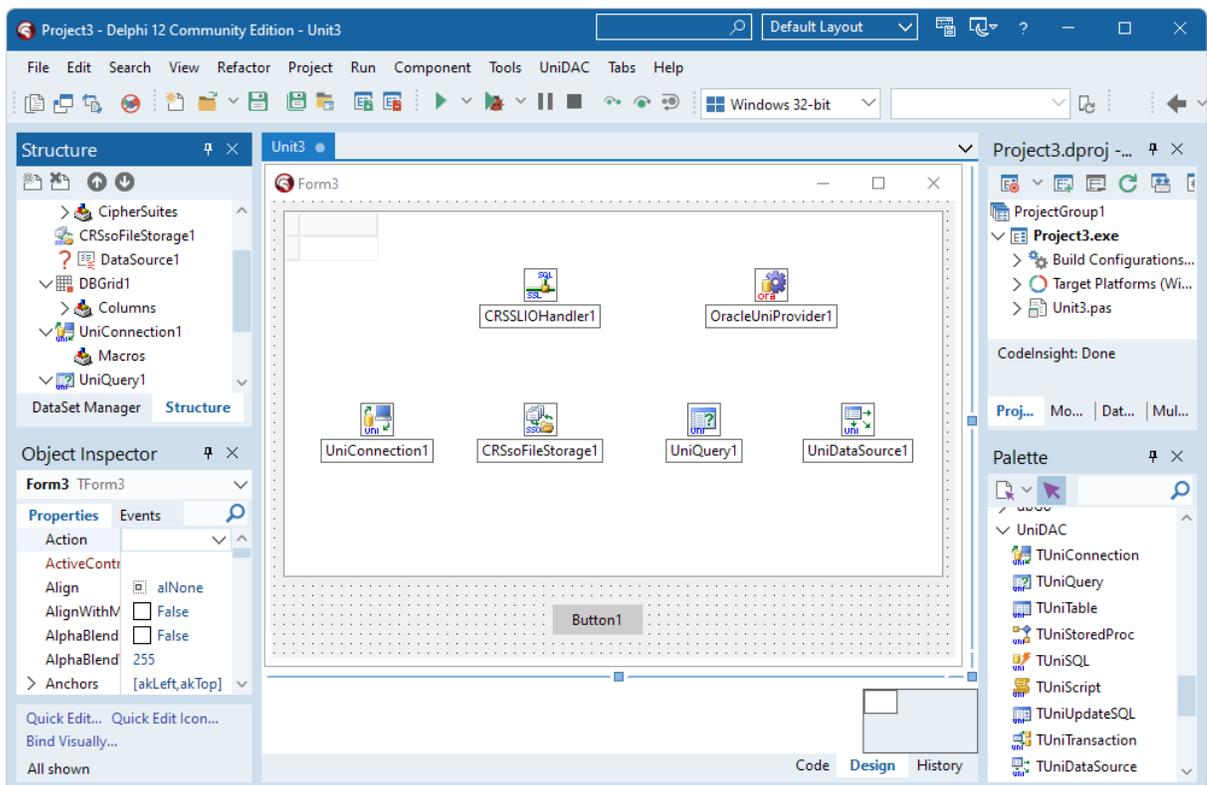
An Oracle wallet is a container for storing authentication and signing credentials, including certificates and keys needed by SSL. For information on creating and obtaining an Oracle wallet, see the official documentation for [Oracle Database](#) or [Oracle Cloud](#).

1. In RAD Studio, select **File > New > Windows VCL Application - Delphi**.

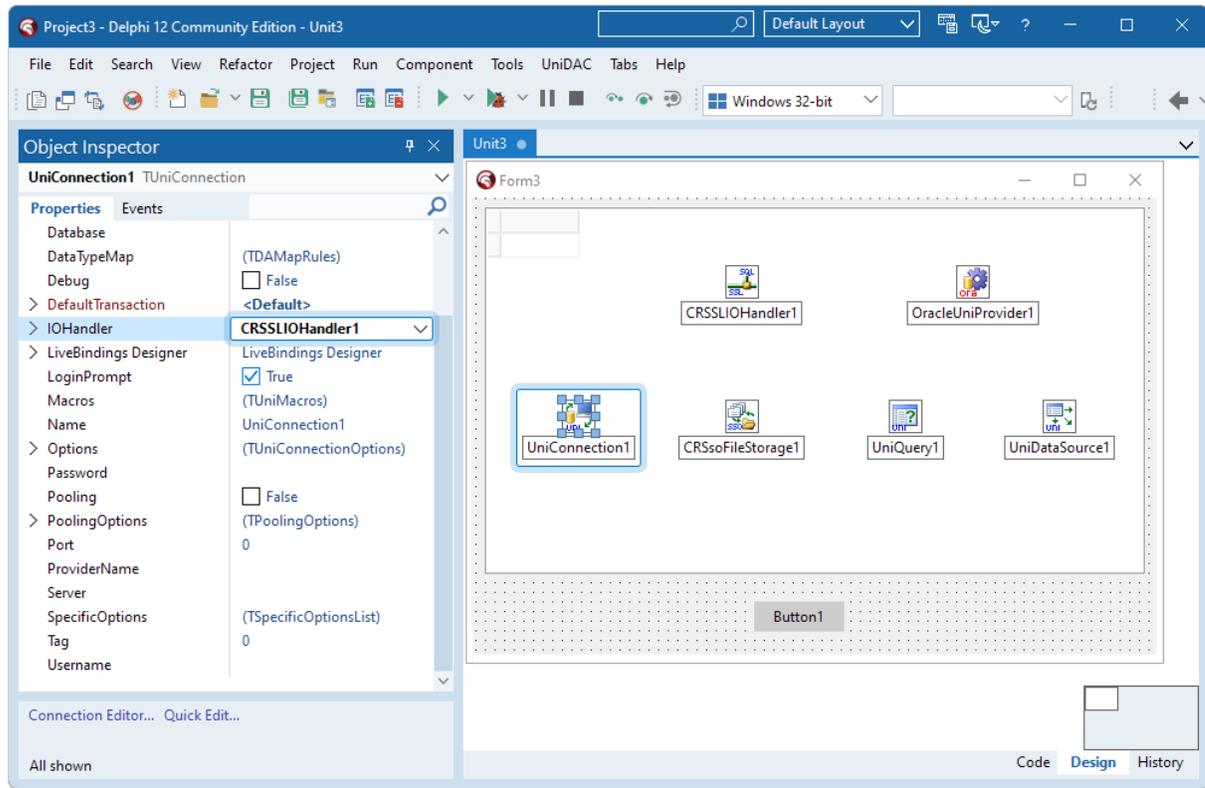


2. Place the following components from **Palette** on the form:

- TDBGrid
- TButton
- TCRSSLIOHandler
- TOracleUniProvider
- TUniConnection
- TCRSsoFileStorage
- TUniQuery
- TUniDataSource



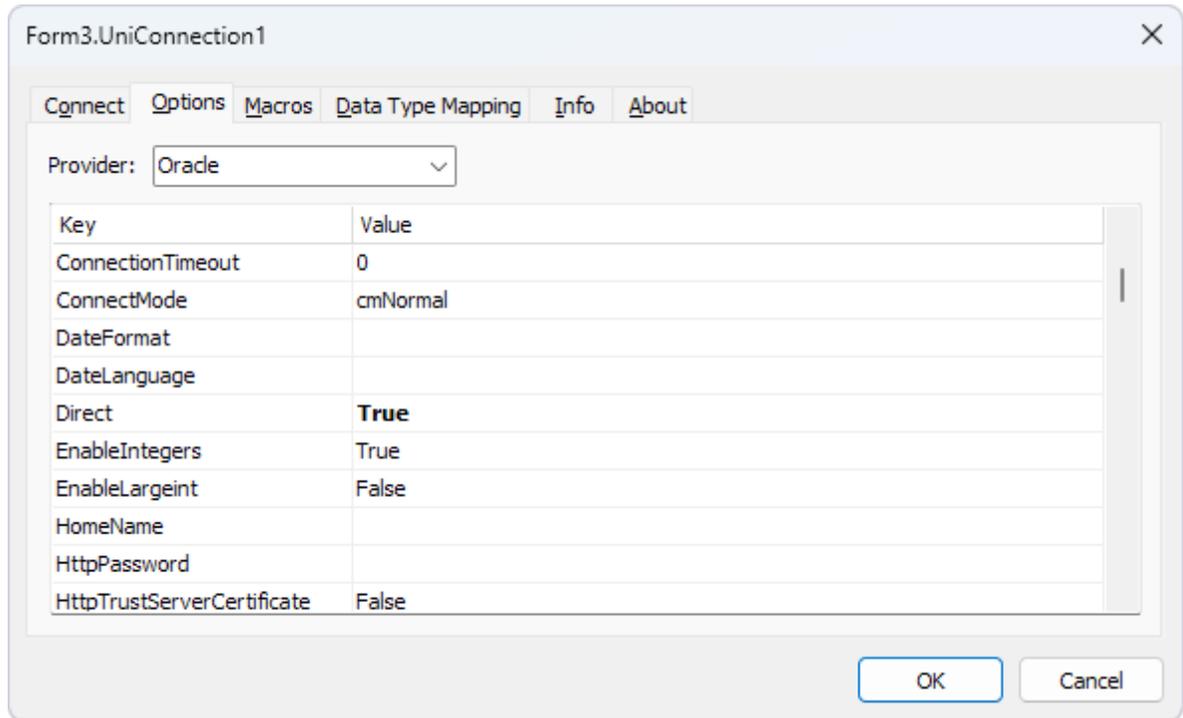
3. Select the **TUniConnection** component and, in **Object Inspector**, set the **IOHandler** property to the instance of **TCRSSLIOHandler (CRSSLIOHandler1)**.



4. Double-click the **TUniConnection** component.

5. On the **Options** tab, fill out the fields:

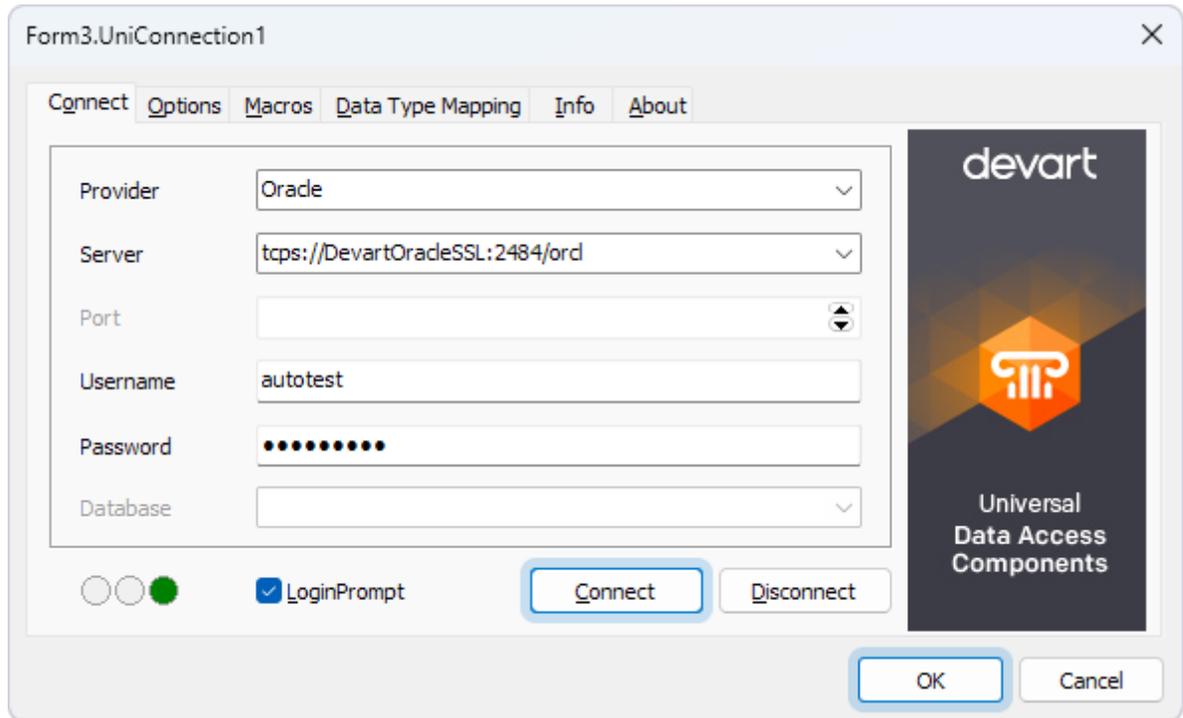
- **Provider** – Select **Oracle**.
- **Direct** – Select **True**.



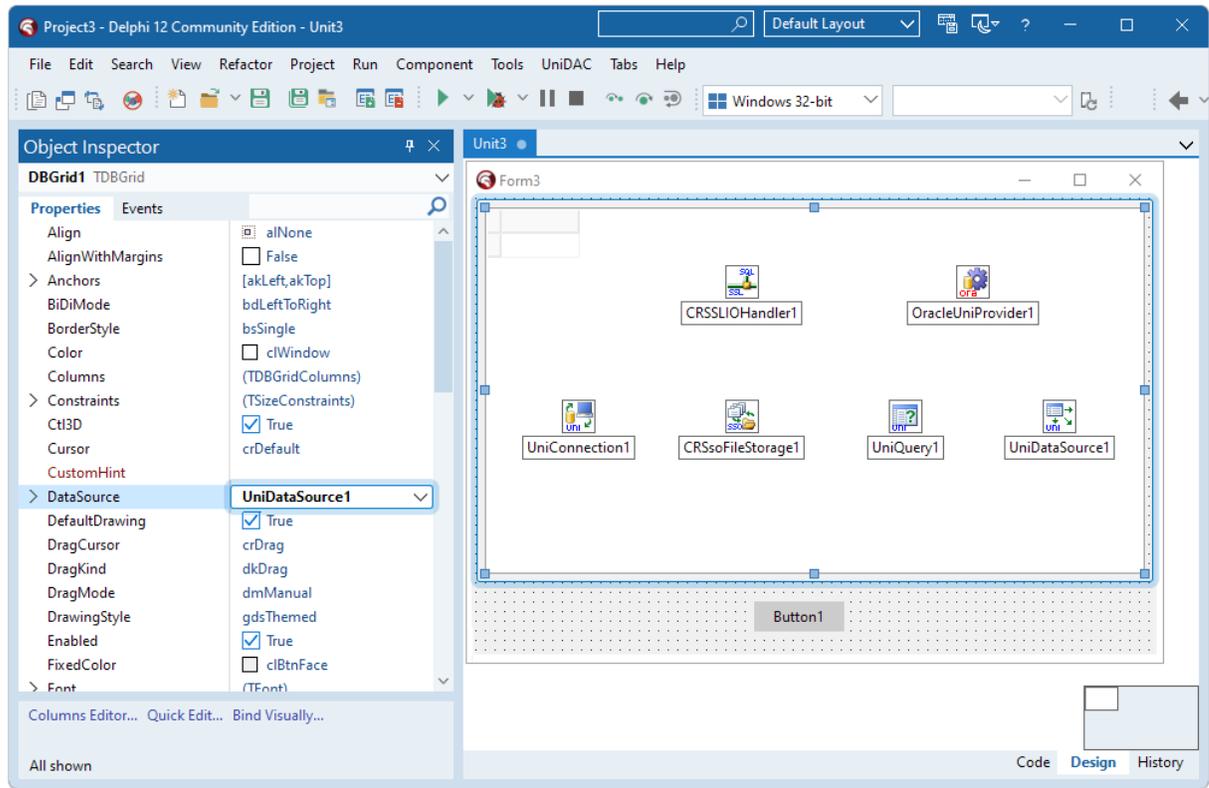
6. On the **Connect** tab, fill out the fields:

- **Provider** – Select **Oracle**.
- **Server** – Enter the host name or IP address of the Oracle server, port, and service name.
- **Username** – Enter the username for the account on the Oracle server.
- **Password** – Enter the password for the account on the Oracle server.

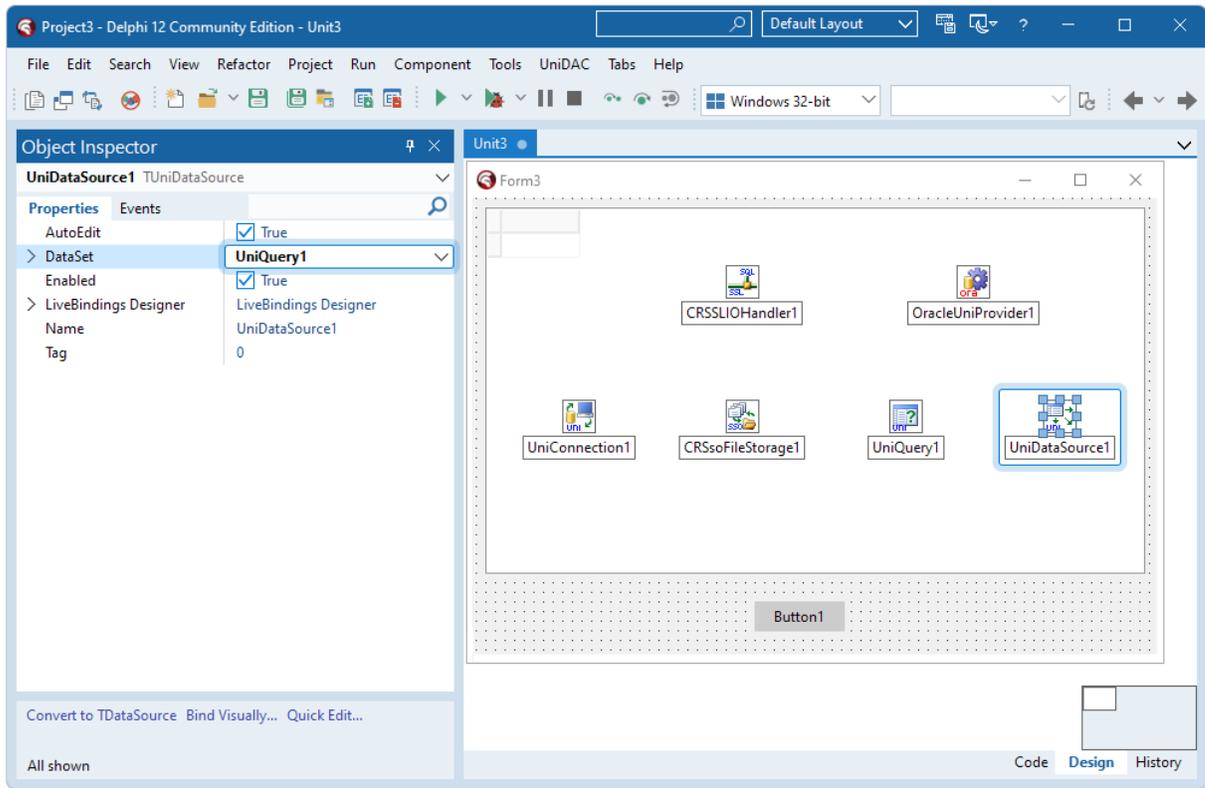
7. Click **Connect** to test the connection to the Oracle server, then click **OK** to close the dialog.



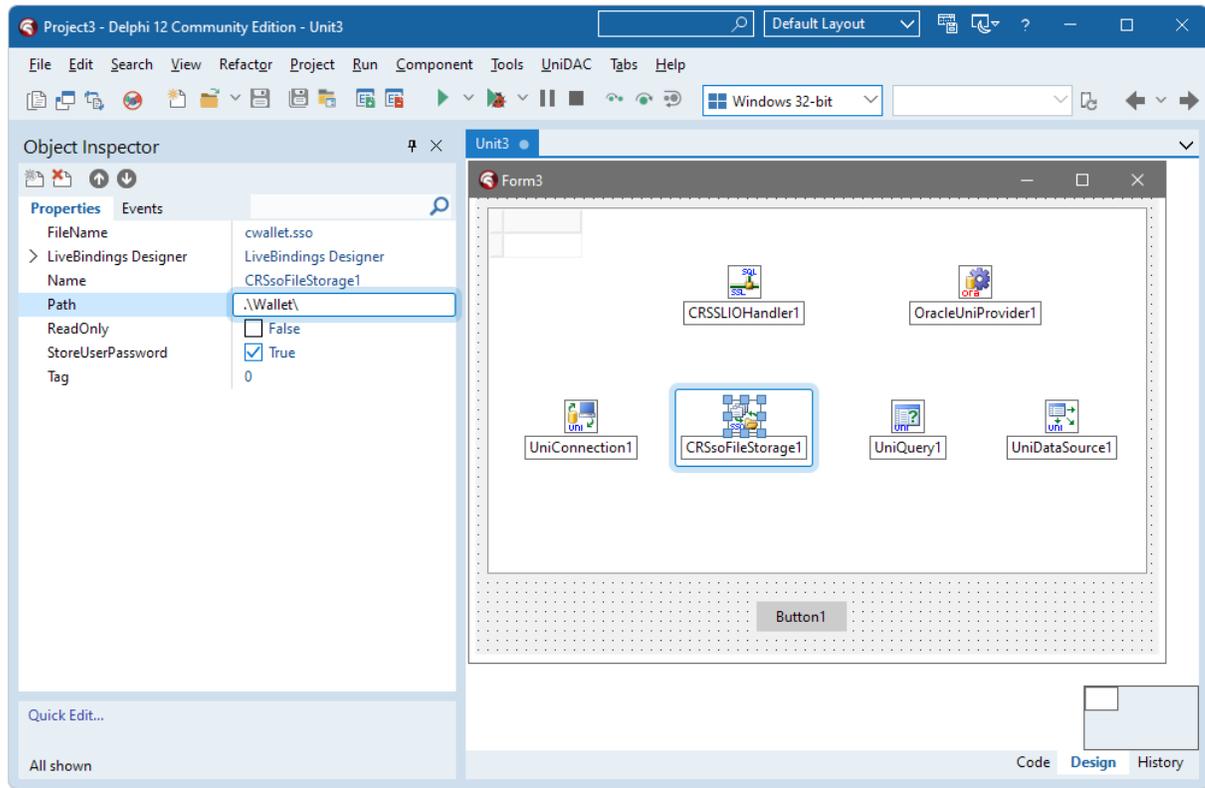
8. Select the **TDBGrid** component and set the **DataSource** property to the instance of **TUniDataSource (UniDataSource1)**.



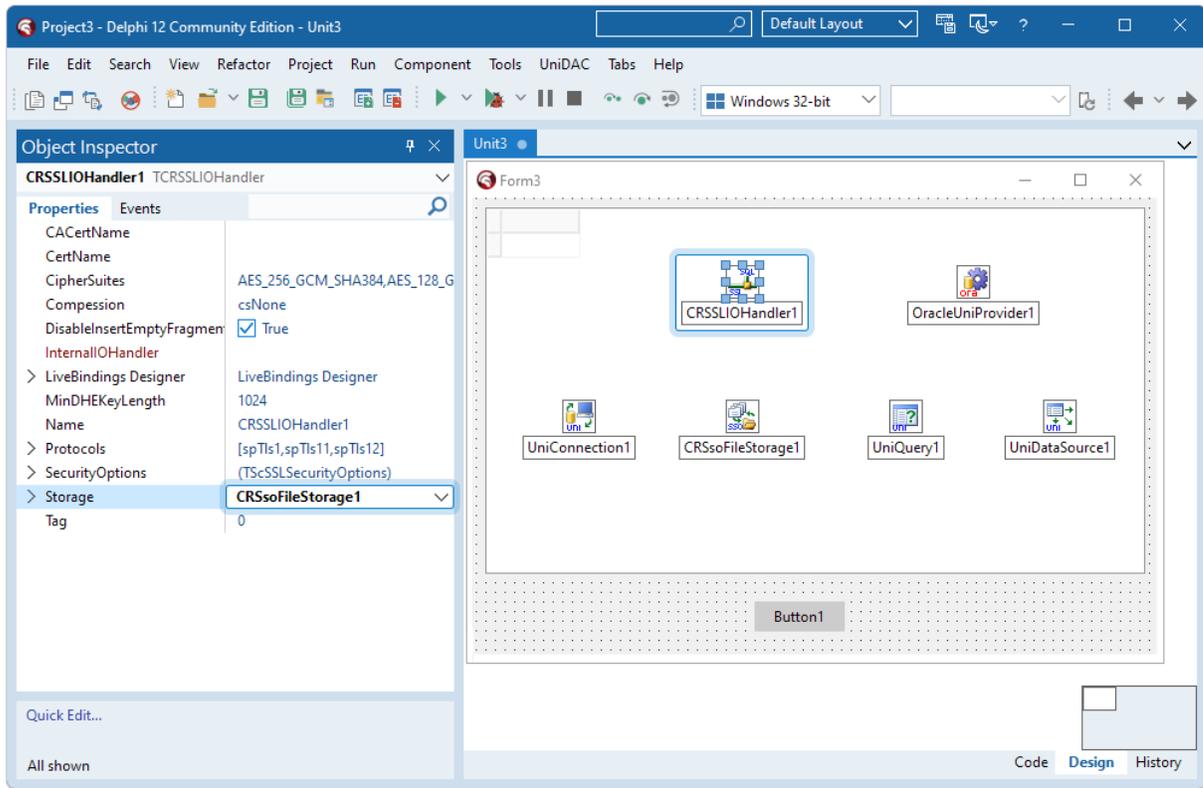
9. Select the **TUniDataSource** component and set the **DataSet** property to the instance of **TUniQuery (UniQuery1)**.



10. Select the **TCRSSoFileStorage** component and specify the path to the Oracle wallet file in the **Path** property.

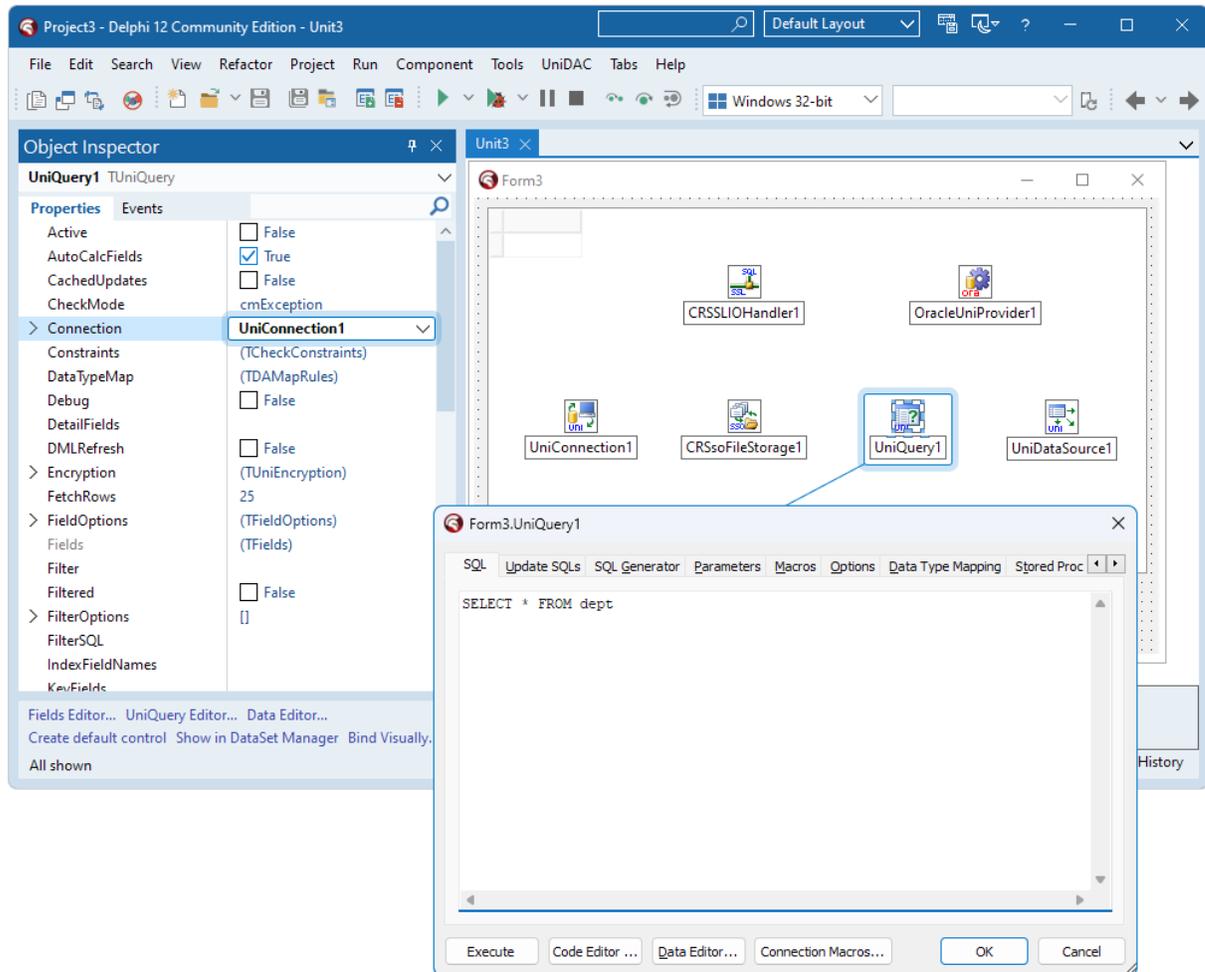


11. Select the **TCRSSLIOHandler** component and set the **Storage** property to the instance of **TCRSsoFileStorage** (**CRSsoFileStorage1**).

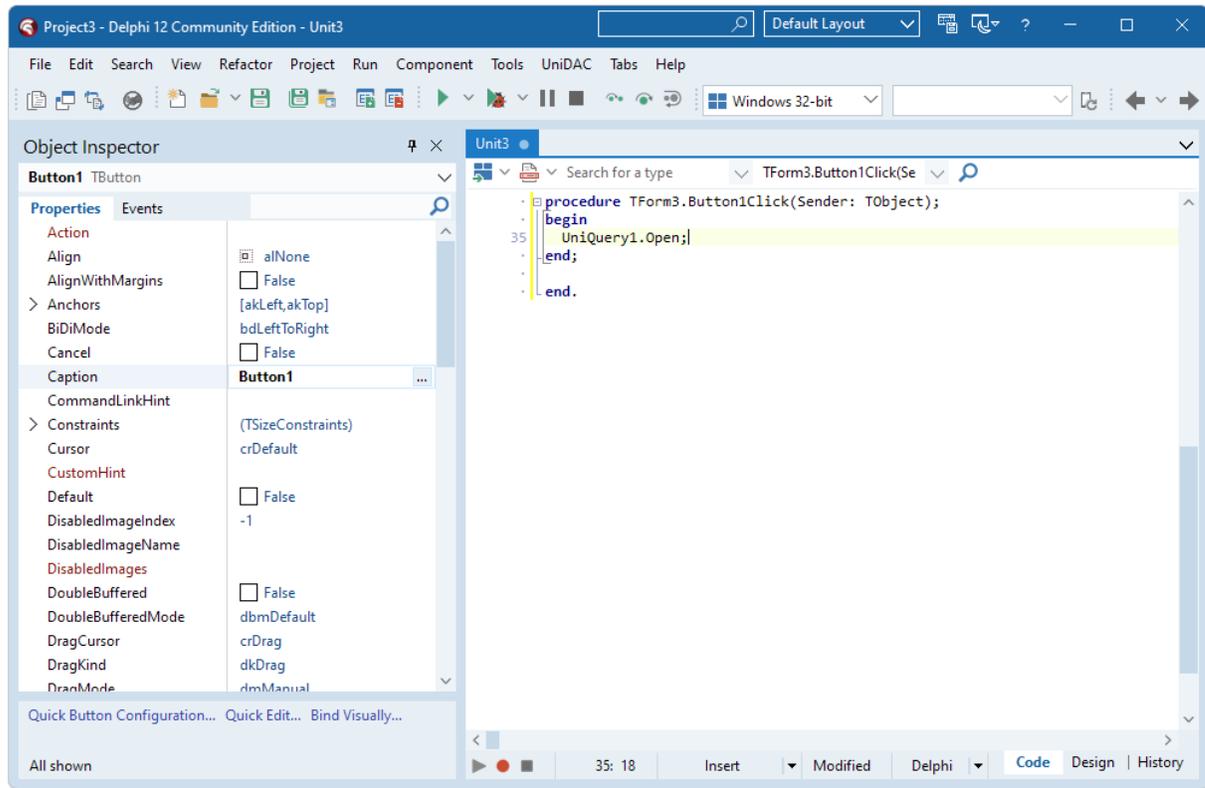


12. Select the **TUniQuery** component and set the **Connection** property to the instance of **TUniConnection (UniConnection1)**.

13. Double-click the **TUniQuery** component, enter a SQL query to be run against the Oracle database, and then click **OK**.

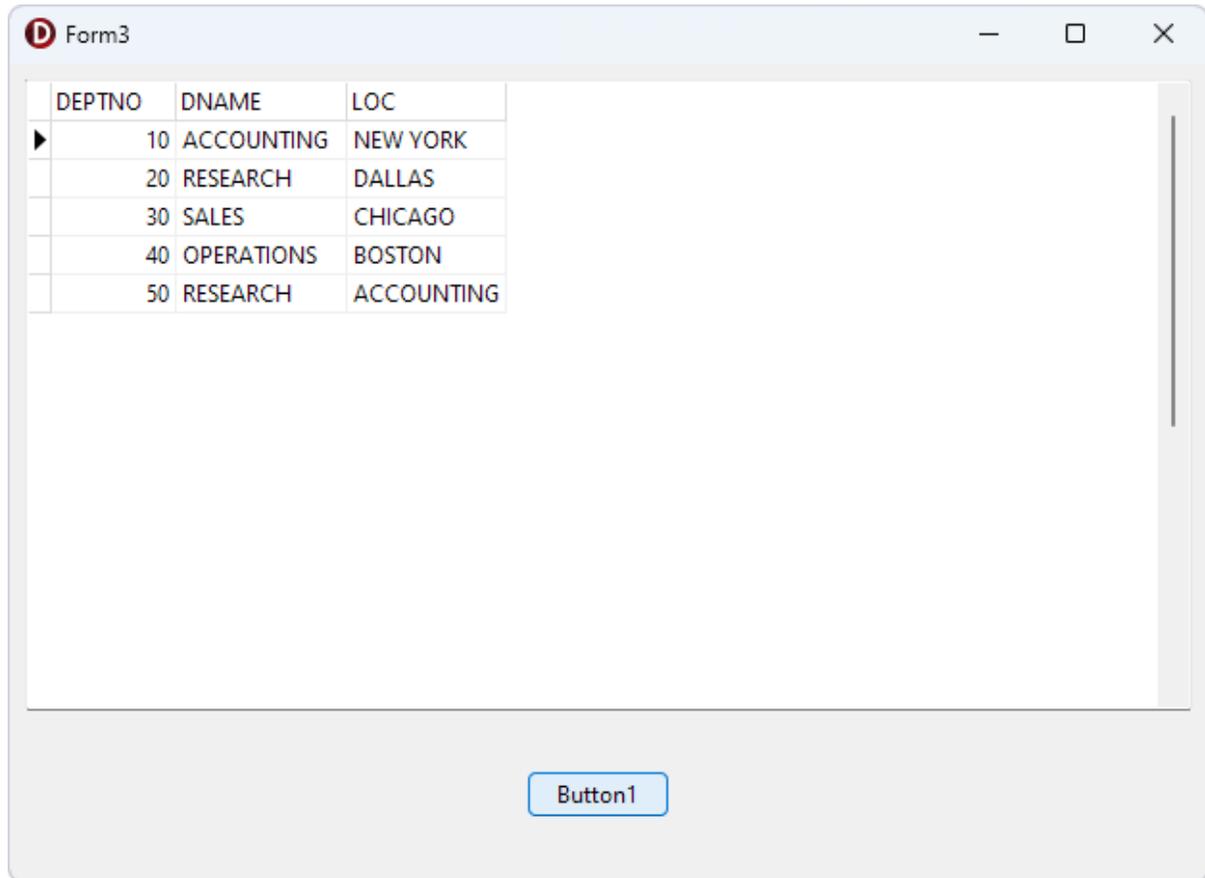


14. Double-click the **TButton** component and add code to call the **Open** method on the **TUniQuery** component instance to activate the dataset when the button is clicked.



15. Press **F9** to compile and run the application.

16. In the form that appears, click **Button1** to run the query. Data appears in the grid.



The screenshot shows a Java Swing window titled "Form3" with a standard title bar (minimize, maximize, close). The window contains a table with the following data:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	RESEARCH	ACCOUNTING

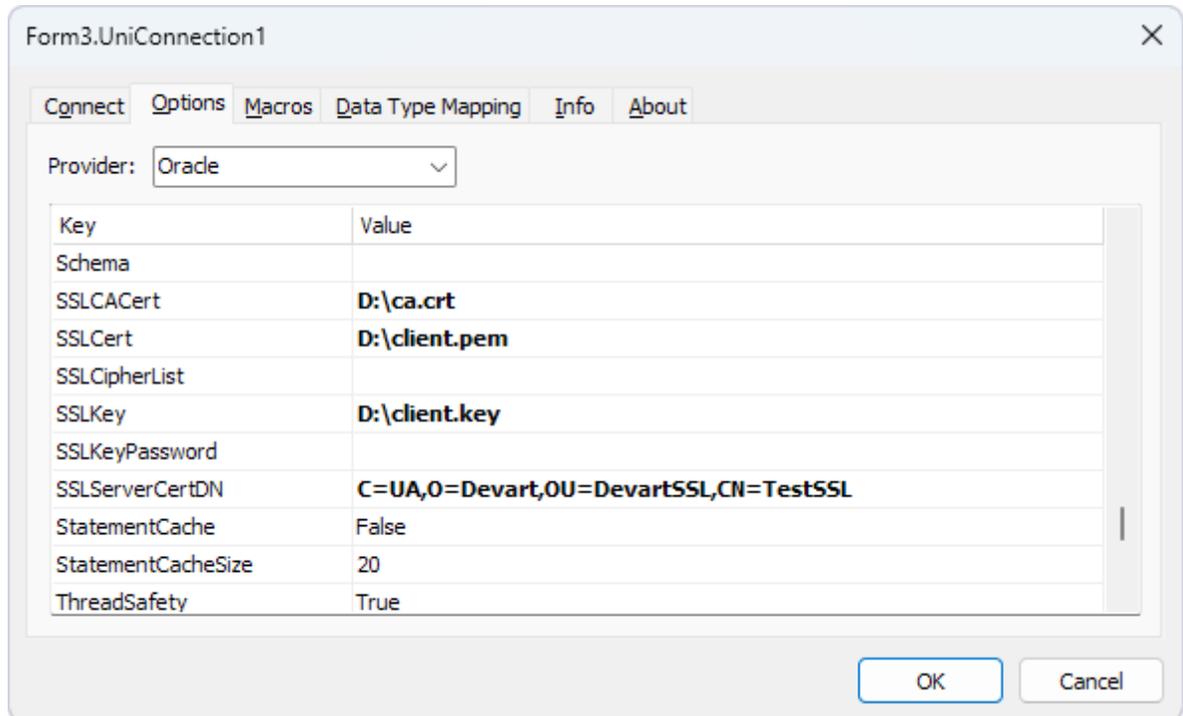
Below the table, there is a button labeled "Button1".

Connect using SSL certificates and keys

If you want to use SSL certificates and keys instead of an Oracle wallet, follow the steps in the previous section with the following modifications:

- In step 2, don't add the **TCRSsoFileStorage** component.
- In step 5, fill out the fields on the **Options** tab as follows:
 - **Provider** – Select **Oracle**.
 - **SSLCACert** – Specify the full path of the Certificate Authority's certificate file.
 - **SSLCert** – Specify the full path of the client certificate file.
 - **SSLKey** – Specify the full path of the client private key file.
 - **SSLServerCertDN** – Specify the server's distinguished name (DN) to enable server DN matching. It is used to check whether the server is genuine by matching the server's

global database name against the DN from the server certificate.



- Skip steps 12 and 13.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.13.3 Connecting via SSH

Secure Shell (SSH) is a network protocol for secure remote login to another system over the internet by connecting an SSH client to an SSH server. SSH provides a mechanism for establishing a secure connection between the client and the remote server, which authenticate each other and exchange messages. It employs different forms of symmetrical encryption, asymmetrical encryption, and hashing. The SSH client initiates a connection and uses public key cryptography to verify the identity of the SSH server.

This page demonstrates how to connect a Delphi application to Oracle with UniDAC using SSH as the encryption method.

You need an SSH client and an SSH server to replicate the steps on this page. You can create them using [SecureBridge](#), a solution provided by Devart. It enables you to embed the SSH client functionality into your Delphi application and, if needed, configure an SSH server.

For more information, see the SecureBridge [tutorial on configuring an SSH server](#).

You can also build the SSHServer [demo project](#), which is distributed with SecureBridge.

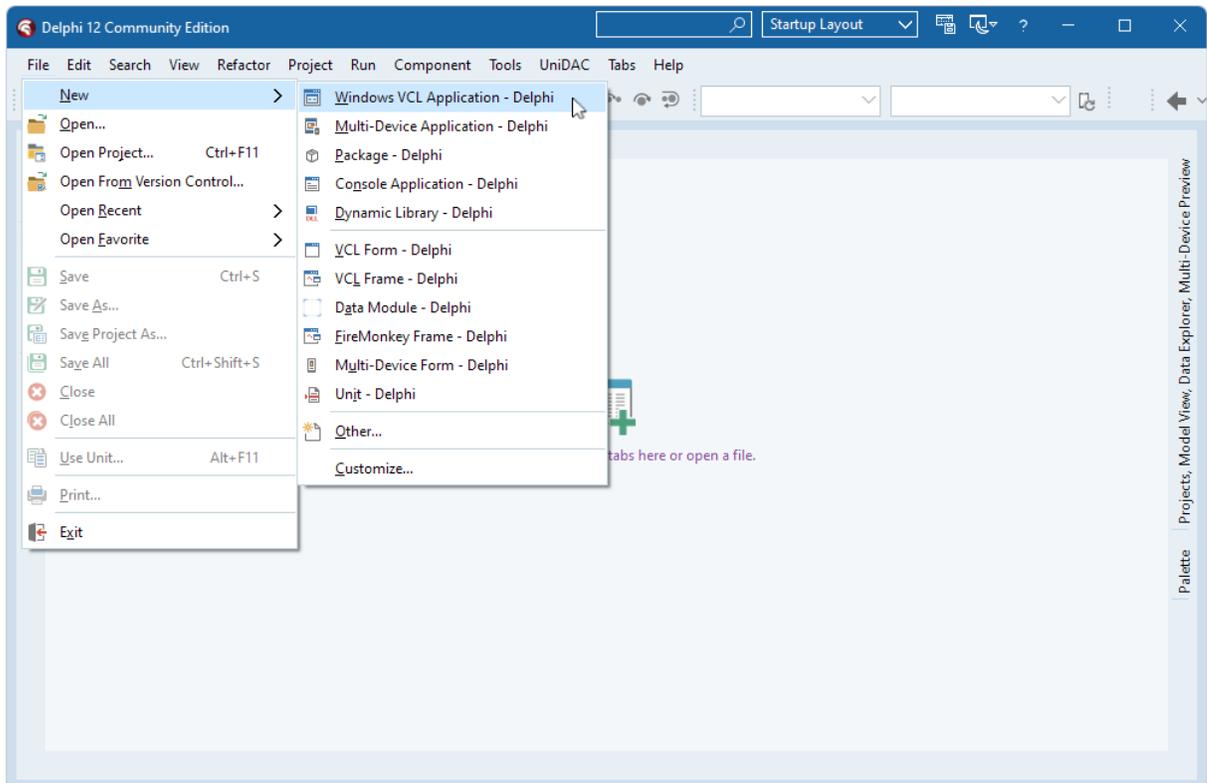
- [Connect Using SecureBridge](#)
- [Connect Using Any Third-Party SSH Tunnel](#)

Connect Using SecureBridge

Prerequisites:

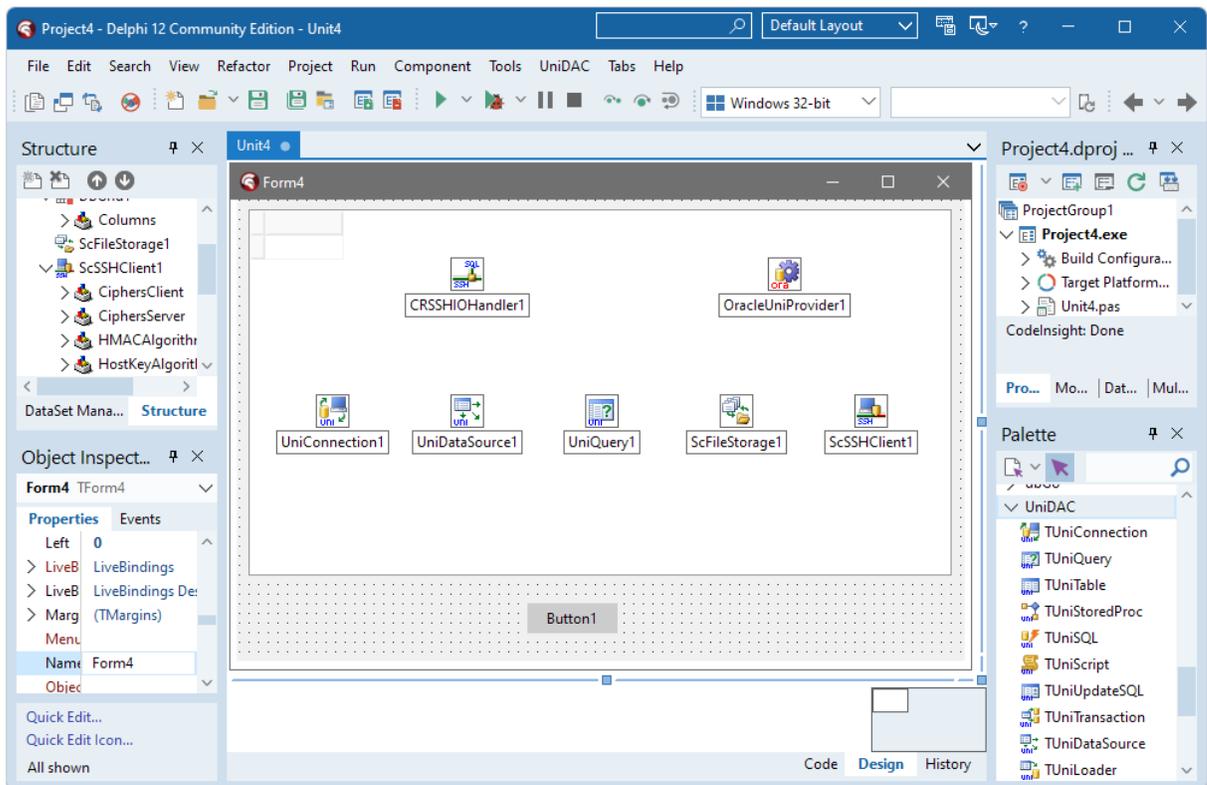
- Install SecureBridge.
- Install the TCRSSHIOHandler component in RAD Studio to bind UniDAC with SecureBridge. For instructions, see `Readme.html`—by default, it is located in `c:\Users\<user>\Documents\Devart\UniDAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge`.

1. In RAD Studio, select **File > New > Windows VCL Application - Delphi**.

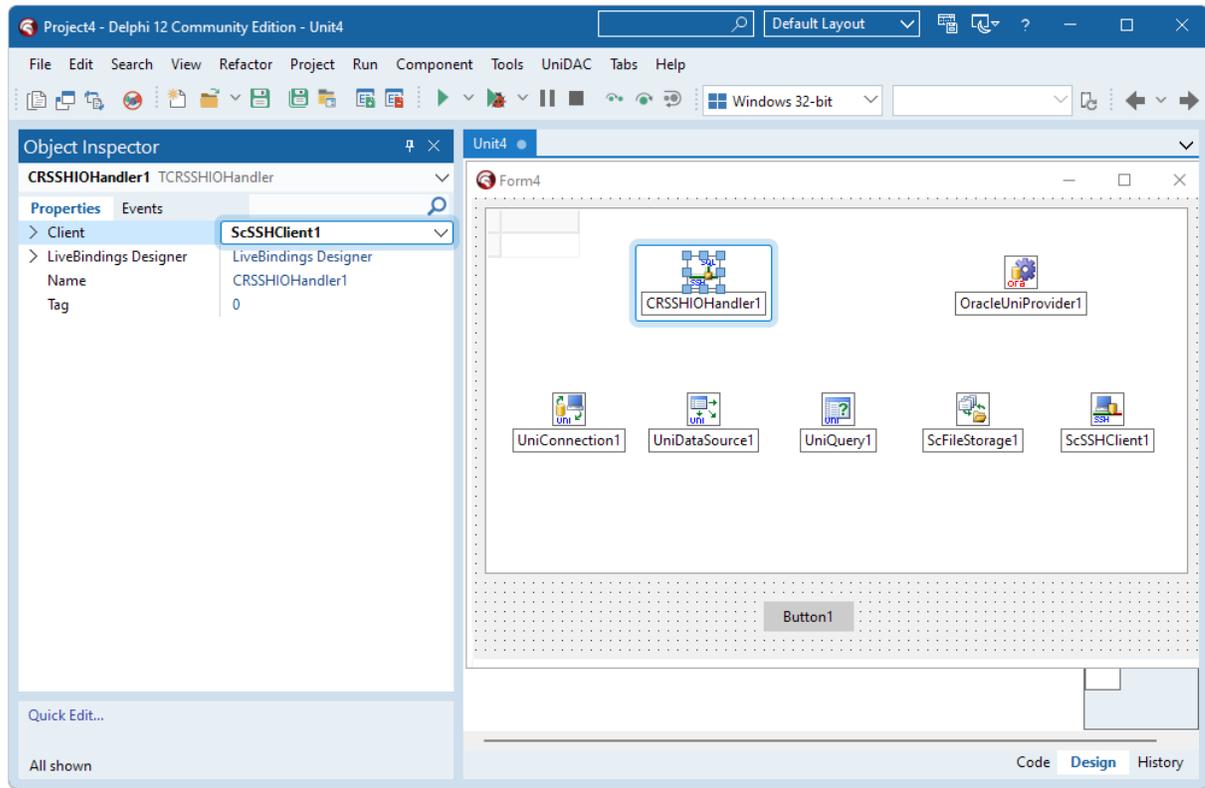


2. Place the following components from **Palette** on the form:

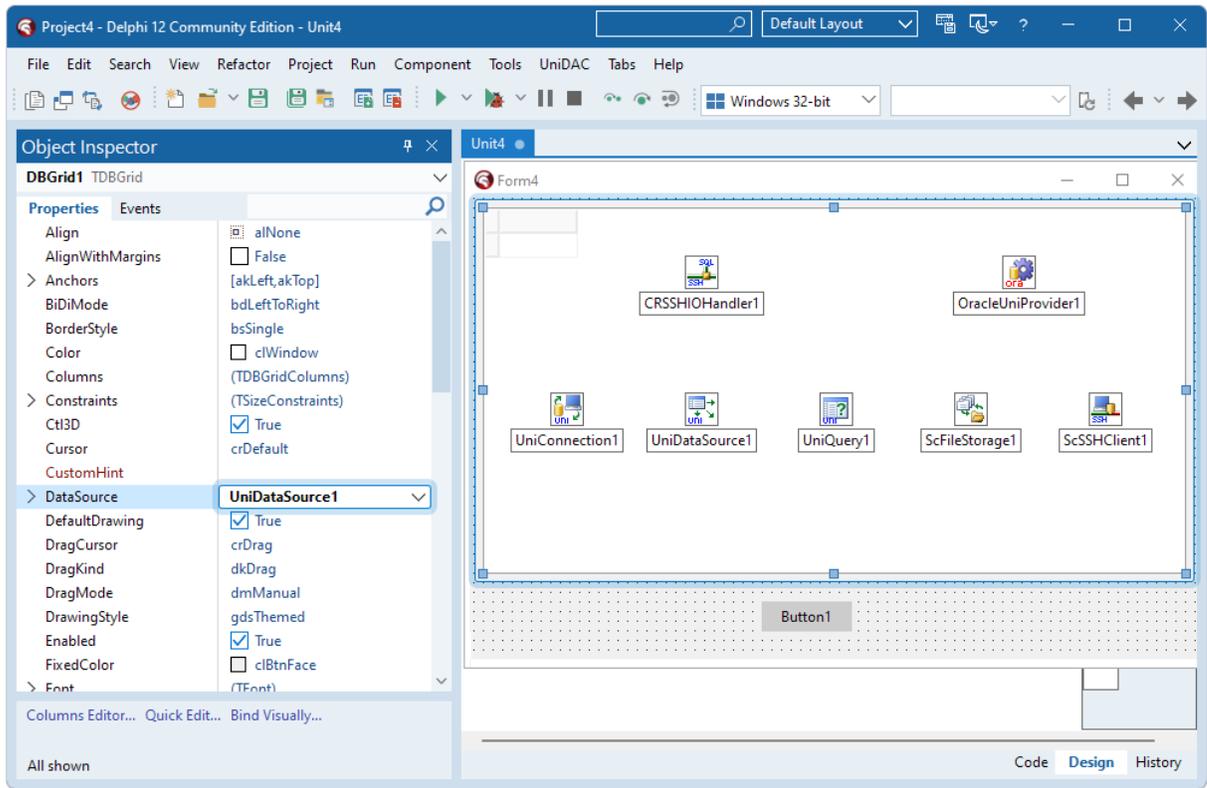
- TDBGrid
- TButton
- TCRSSHIOHandler
- TOracleUniProvider
- TUniConnection
- TUniDataSource
- TUniQuery
- TScFileStorage
- TScSSHClient



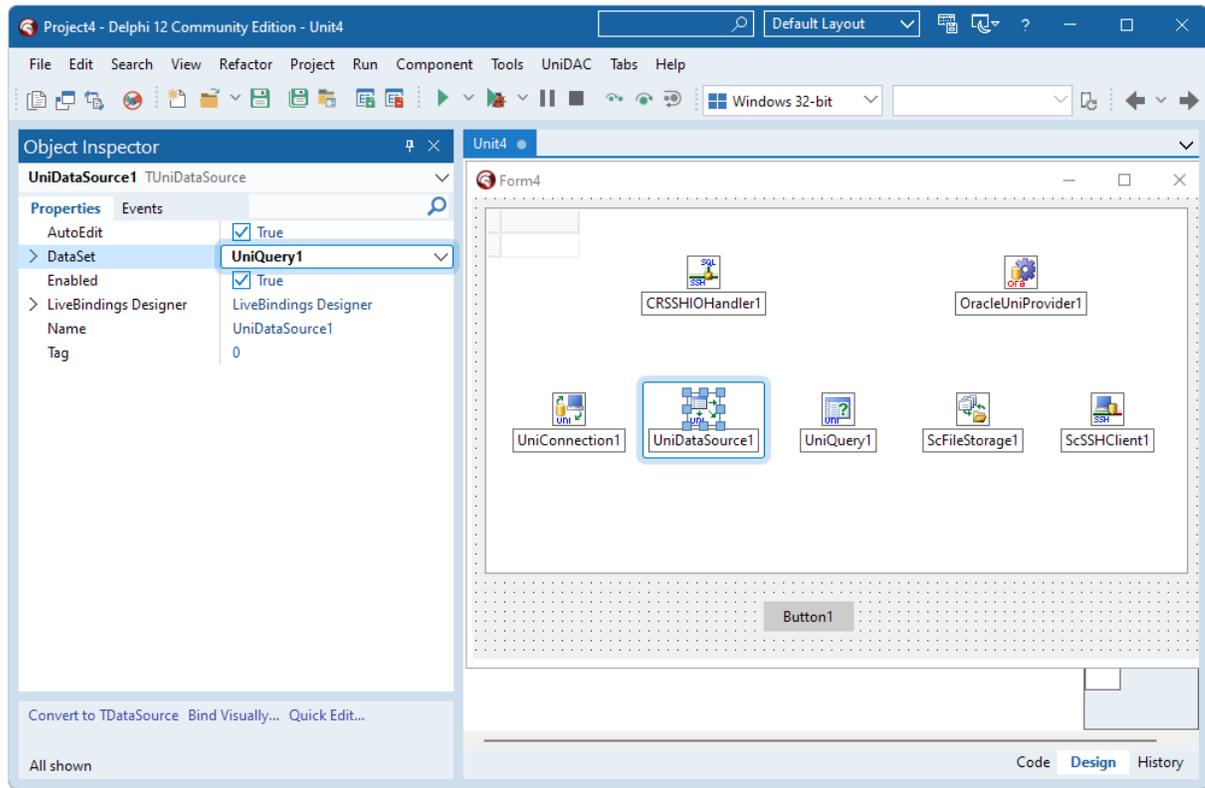
3. Select the **TCRSSHIOHandler** component and set the **Client** property to the instance of **TScSSHClient** (**ScSSHClient1**).



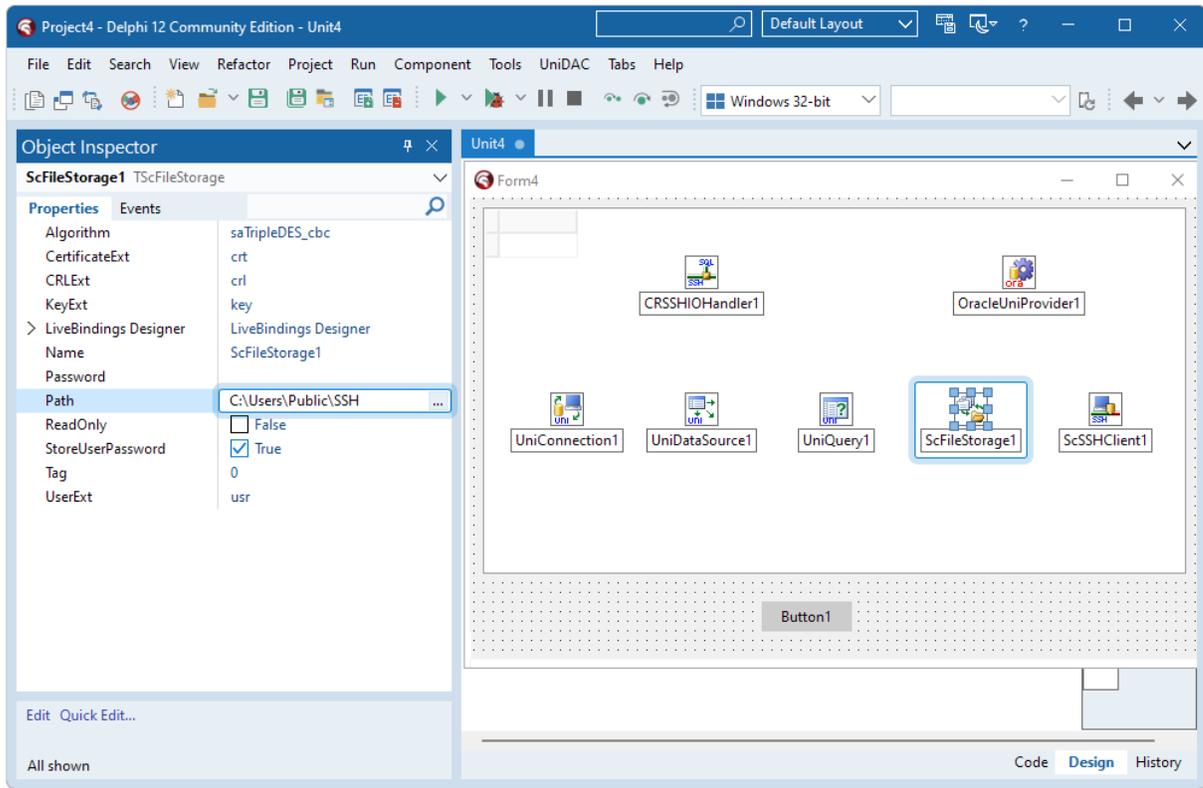
4. Select the **TDBGrid** component and, in **Object Inspector**, set the **DataSource** property to the instance of **TUniDataSource** (**UniDataSource1**).



5. Select the **TUniDataSource** component and set the **DataSet** property to the instance of **TUniQuery (UniQuery1)**.



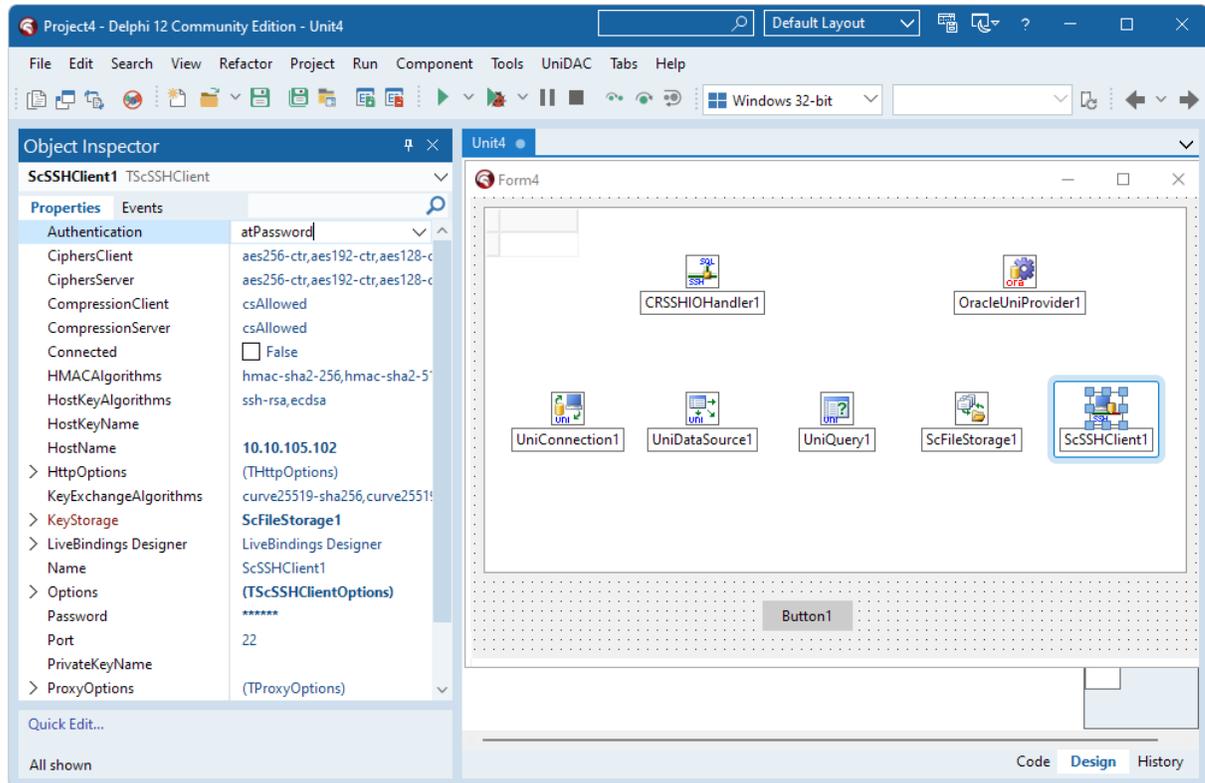
6. Select the **TScFileStorage** component and, in the **Path** property, specify the directory where information about keys and users is stored.



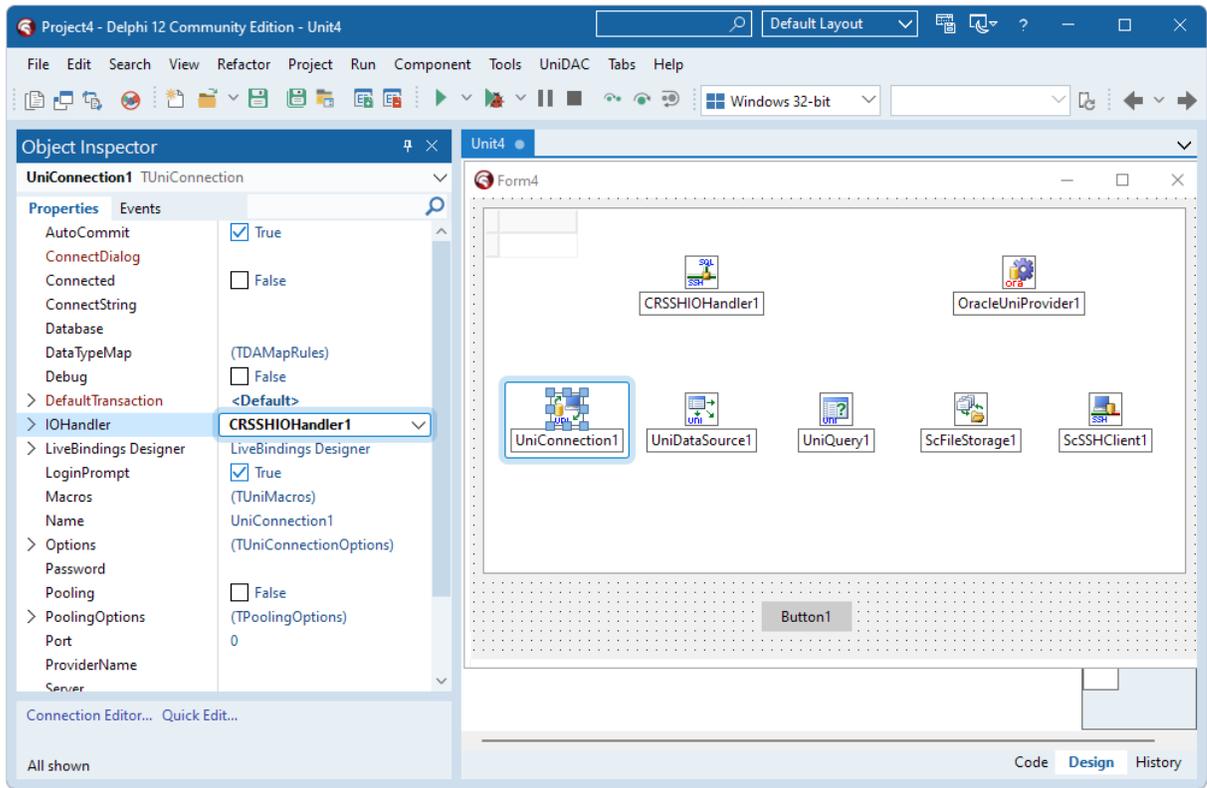
7. Select the **TScSSHClient** component and assign values to the following properties:

- **Authentication** – Select the value depending on the authentication method applicable for your SSH server: **atPassword** or **atPublicKey**.
- **HostKeyName** – For public key authentication, specify the filename of the SSH server public key.
- **Hostname** – Enter the host name or IP address of the SSH server.
- **KeyStorage** – Set the property to the instance of **TScFileStorage** (**ScFileStorage1**).
- **Password** – For password authentication, enter the password for the account on the SSH server.
- **Port** – Specify the SSH port.
- **PrivateKeyName** – For public key authentication, specify the filename of the client private key.
- **User** – Enter the username for the account on the SSH server.

If you are connecting to **Oracle Cloud**, leave the **Password** and **HostKeyName** properties empty—only specify **PrivateKeyName** and **User** (the default username is **opc**).



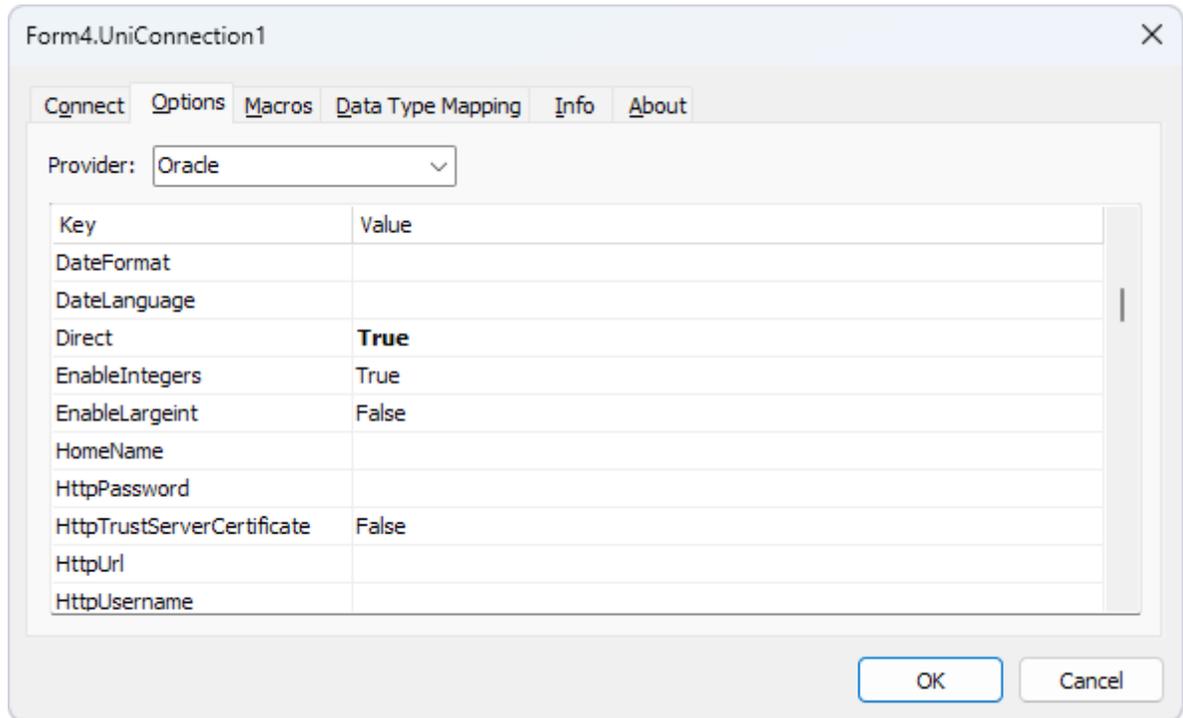
8. Select the **TUniConnection** component and set the **IOHandler** property to the instance of **TCRSSHIOHandler** (**CRSSHIOHandler1**).



9. Double-click the **TUniConnection** component.

10. On the **Options** tab, fill out the fields:

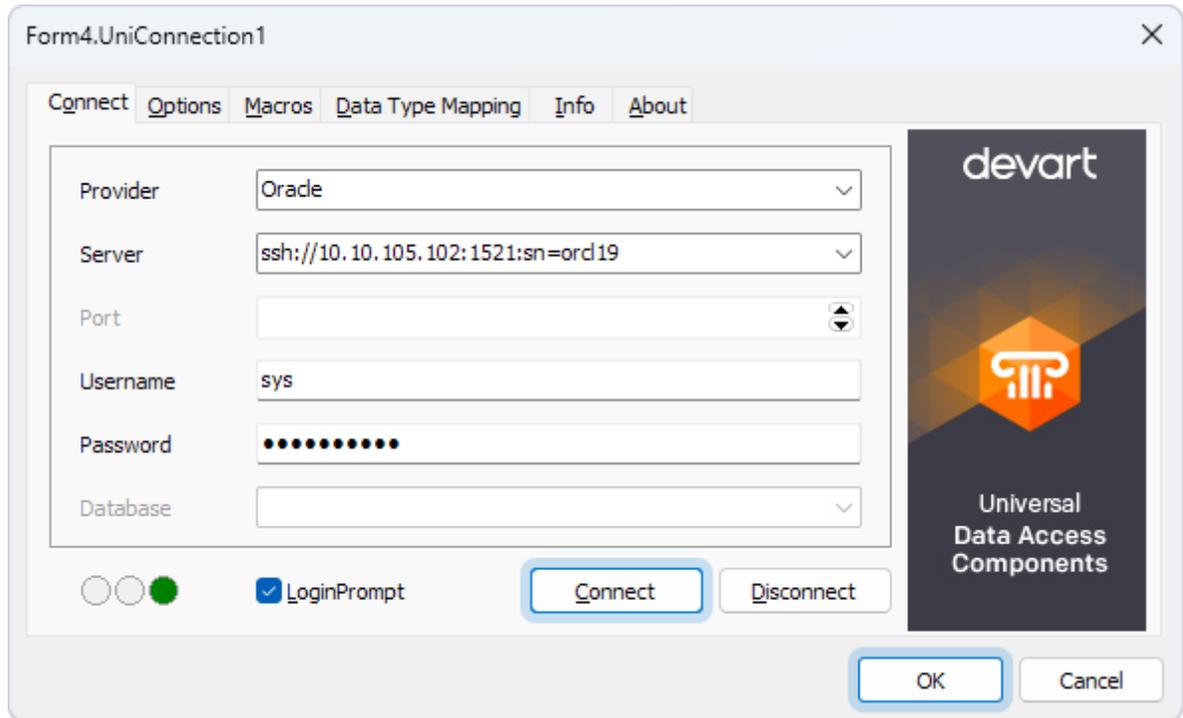
- **Provider** – Select **Oracle**.
- **Direct** – Select **True**.



11. On the **Connect** tab, fill out the fields:

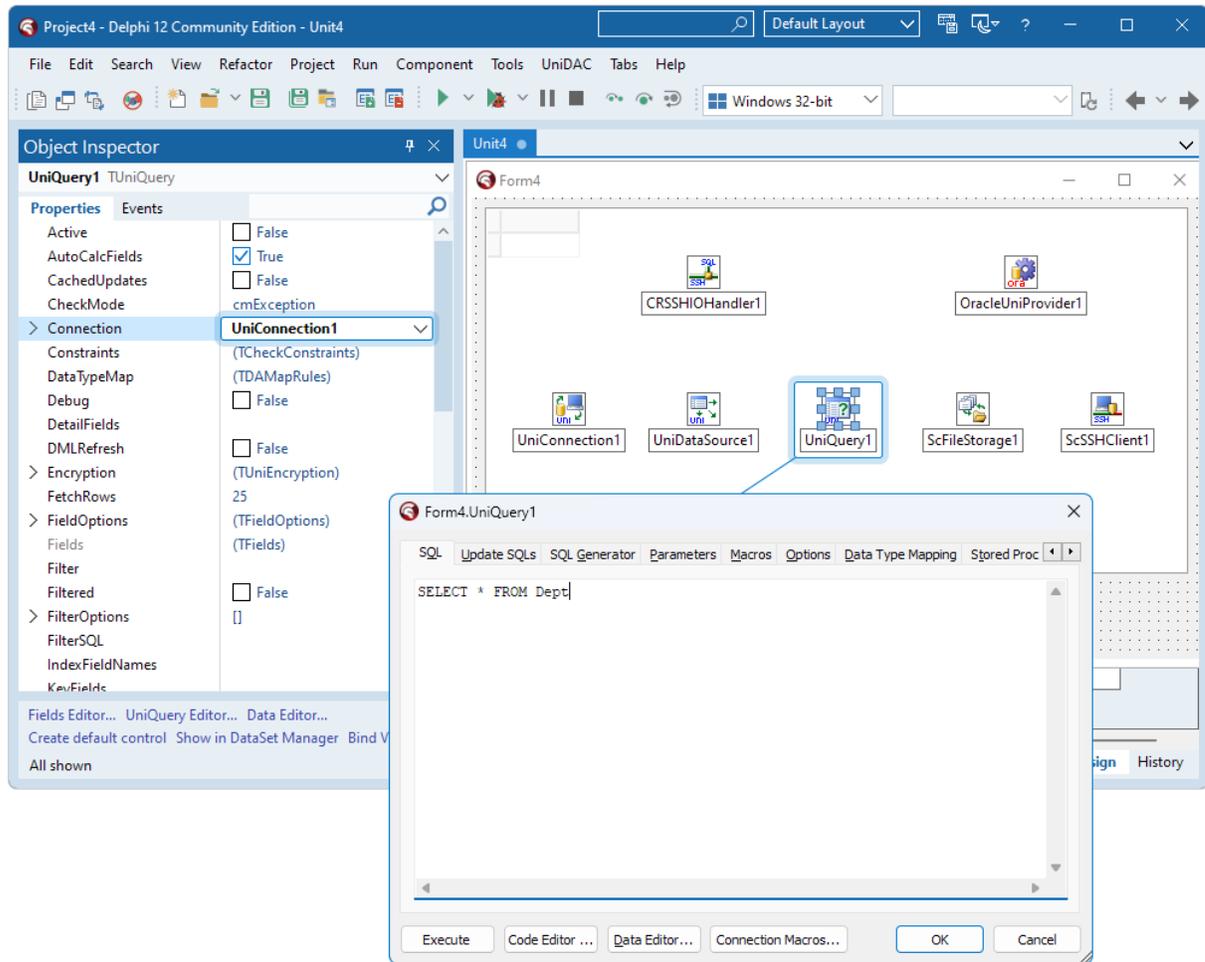
- **Provider** – Select **Oracle**.
- **Server** – Enter the host name or IP address of the Oracle server, port, and service name.
- **Username** – Enter the username for the account on the Oracle server.
- **Password** – Enter the password for the account on the Oracle server.

12. Click **Connect** to test the connection to the Oracle server, then click **OK** to close the dialog.

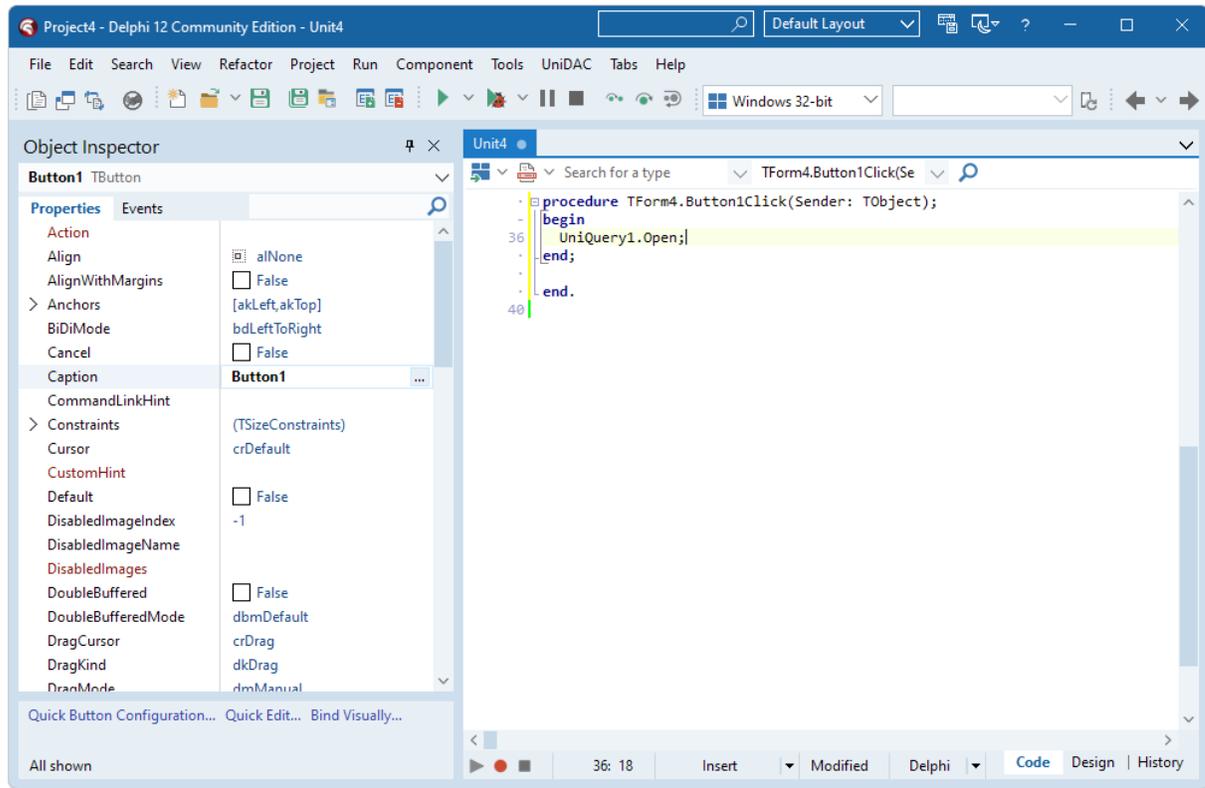


13. Select the **TUniQuery** component and set the **Connection** property to the instance of **TUniConnection (UniConnection1)**.

14. Double-click the **TUniQuery** component, enter a SQL query to be run against the Oracle database, and click **OK**.

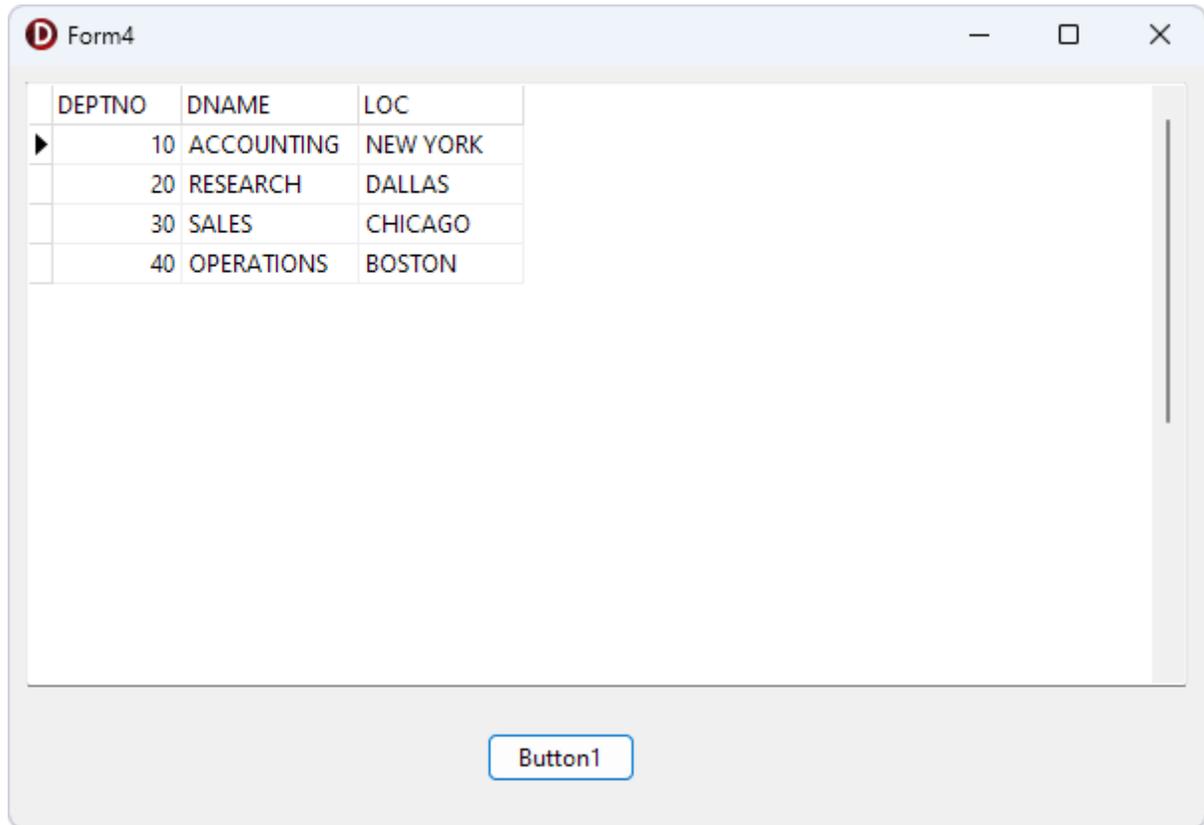


15. Double-click the **TButton** component and add code to call the **Open** method on the **TUniQuery** component instance to activate the dataset when the button is clicked.



16. Press **F9** to compile and run the application.

17. In the form that appears, click **Button1** to run the query. Data appears in the grid.



Connect Using Any Third-Party SSH Tunnel

It is not mandatory to use SecureBridge and its components—you can use any other server that implements the SSH protocol.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.13.4 Network Tunneling

In typical scenarios, a client connects to a server through a direct network path. However, modern security policies or complex network topologies often require the use of a proxy or traversal through a firewall. This article describes different ways to connect to an Oracle server using UniDAC.

- [Direct connection](#)
- [Connection through an HTTP tunnel](#)
- [Connection through a proxy and an HTTP tunnel](#)

- [Connection through a proxy and an HTTPS tunnel](#)
- [Additional information](#)

Direct connection

A direct connection to the server means that the server host is accessible from the client without any additional 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 to communicate with the server. Use it whenever possible.

The following example demonstrates how to configure a direct connection:

```
UniConnection := TUniConnection.Create(self);
UniConnection.ProviderName := 'Oracle';
UniConnection.Server := 'host_name:1521/service_name';
UniConnection.Username := 'user';
UniConnection.Password := 'password';
UniConnection.SpecificOptions.Values['Direct'] := 'True';
UniConnection.Connect;
```

Connection through an HTTP tunnel

Sometimes, client computers are shielded by a firewall that does not allow you to connect directly to the server on the specified port. If the firewall allows HTTP connections, you can use UniDAC and HTTP tunneling software to connect to the Oracle server.

UniDAC supports HTTP tunneling based on a PHP script.

Consider a scenario where direct access to the Oracle server port is blocked on a remote host, and only HTTP traffic on port 80 is permitted. In such cases, web script tunneling allows remote access to Oracle as if it were a direct connection.

You need to deploy the `tunnel.php` script included in the provider package on the web server. It allows access to the Oracle server using HTTP tunneling. The script must be accessible through the HTTP protocol. You can check if it is accessible using a web browser. You can find the script in the HTTP subfolder of the installed provider folder, such as `%Program Files%\Devart\UniDac for Delphi X\HTTP\tunnel.php`. The only requirement is that the server must support PHP 5.

To connect to Oracle via HTTP tunneling, configure `TUniConnection` as for a direct connection from the web server and provide the following HTTP tunneling parameters:

Specific Option	Required	Description
HttpUrl	Yes	The URL of the tunneling PHP script. For example, if the script is in the server root, the URL can be <code>http://localhost/tunnel.php</code> .
HttpUsername, HttpPassword	No	A registered user credentials. Set these properties if access to the website folder with the script is available only for registered users authenticated with a username and password.

Connection through a proxy and an HTTP tunnel

Consider the previous case with one more complication.

The HTTP tunneling server is not directly accessible from the client computer. For example, the client address is 10.0.0.2 and the server address is `host_name:1521/service_name`. Since the client and the server are on different networks, the client can only access the server through a proxy located at 10.0.0.1 and listening on port 808. In this case, in addition to the HTTP-specific options, you need to set the proxy-specific options.

```

UniConnection := TUniConnection.Create(self);
UniConnection.ProviderName := 'Oracle';
UniConnection.Server := 'host_name:1521/service_name';
UniConnection.Username := 'user';
UniConnection.Password := 'password';
UniConnection.SpecificOptions.Values['HttpUrl'] := 'http://server/tunnel.php';
UniConnection.SpecificOptions.Values['ProxyHostname'] := '10.0.0.1';
UniConnection.SpecificOptions.Values['ProxyPort'] := '808';
UniConnection.SpecificOptions.Values['ProxyUsername'] := 'proxy_user';
UniConnection.SpecificOptions.Values['ProxyPassword'] := 'proxy_password';
UniConnection.SpecificOptions.Values['Direct'] := 'True';
UniConnection.Connect;

```

Note that when proxy-specific options are configured, UniDAC will automatically route the connection through the specified proxy server.

Connection through a proxy and an HTTPS tunnel

The following example shows the use of an HTTPS tunnel in the same scenario as previously.

```

UniConnection := TUniConnection.Create(self);
UniConnection.ProviderName := 'Oracle';
UniConnection.Server := 'host_name:1521/service_name';
UniConnection.Username := 'user';
UniConnection.Password := 'password';
UniConnection.SpecificOptions.Values['HttpsUrl'] := 'https://server/tunnel.p

```

```
UniConnection.SpecificOptions.Values['ProxyHostname'] := '10.0.0.1';
UniConnection.SpecificOptions.Values['ProxyPort'] := '808';
UniConnection.SpecificOptions.Values['ProxyUsername'] := 'proxy_user';
UniConnection.SpecificOptions.Values['ProxyPassword'] := 'proxy_password';
UniConnection.SpecificOptions.Values['Direct'] := 'True';
UniConnection.Connect;
```

Additional information

Remember that traffic tunneling or encryption increases CPU usage and network load. Direct connections are recommended whenever possible.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.14 UniDAC and SQLite

5.1.14.1 SQLite Provider

This article provides a brief overview of the SQLite data access provider for UniDAC used to establish a connection to SQLite databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [SQLite-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [Data Type Mapping](#)
- [Encryption](#)



Overview

The main features of the SQLite data access provider are:

- High performance
- Easy deployment
- Comprehensive support for the latest versions of SQLite

The full list of SQLite provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the SQLite provider. Express Edition of UniDAC does not include the SQLite provider.

Compatibility

To learn about SQLite compatibility, refer to the [Compatibility](#) section.

Requirements

Applications that use the SQLite provider require SQLite client library (sqlite3.dll). The SQLite provider dynamically loads SQLite client DLL available on user systems. To locate DLL you can set the ClientLibrary specific option of TUniConnection with the path to the client library. By default the SQLite provider searches a client library in directories specified in the PATH environment variable.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

SQLite-specific options

Though UniDAC is components that provide unified interface to work with different database servers, it also lets you tune behaviour for each server individually. For thin setup of a certain database server, UniDAC provides server-specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list. Therefore you can use the following syntax to assign an option value:

```
UniConnection.SpecificOptions.Values['CharLength'] := '1';
```

Below you will find the description of allowed options grouped by components.

TUniConnection

Option	Description
ASCIIDataBase	Enables or disables ASCII support. The default value is <code>False</code> . Note: Set the <code>UseUnicode</code> option to <code>False</code> before enabling ASCII support.
BusyTimeout	Sets the timeout in milliseconds for a locked resource (database or table). If the resource is not unlocked during this time, SQLite returns the <code>SQLITE_BUSY</code> error. The default value is 0.
CipherLicense	Holds a license key for SQLCipher Commercial Edition. Note that SQLCipher is not supported in the Direct mode.
ConnectMode	The connection mode. <code>cmDefault</code> The default value. The database is opened for reading and writing. Corresponds to the SQLite default behavior. <code>cmReadWrite</code> The database is opened for reading and writing. <code>cmReadOnly</code> The database is opened in read-only mode. Note that the <code>ForceCreateDatabase</code> option can be used to enable the automatic creation of the database when it doesn't already exist.
ClientLibrary	Use the <code>ClientLibrary</code> option to set or get the location of the client library.
DateFormat	The format for storing dates in the database. If no format is specified, the default format <code>yyyy-mm-dd</code> is used. The default value is an empty string.

DefaultCollations	Enables or disables automatic default collations registration on connection establishing.
Direct	When True, UniDAC connects to the database directly using the embedded SQLite3 engine, without the SQLite3 client library. The Direct mode also enables you to work with an encrypted database using the EncryptionAlgorithm and EncryptionKey options and the EncryptDatabase method.
EnableLoadExtension	Enables loading and using an SQLite extension: <code>UniConnection.ExecSQL('SELECT load_extension(''C:\</code>
EnableSharedCache	Enables or disables the SQLite shared-cache mode. The default value is False.
EncryptionAlgorithm	Used to specify the encryption algorithm for an encrypted database.
EncryptionKey	This property is used for password input and for working with encrypted database. Password can be set or changed using EncryptDatabase method.
ForceCreateDatabase	Forces TLiteConnection to create a new database before opening a connection, if the database doesn't already exist.
ForeignKeys	Enables or disables foreign keys constraints without explicitly executing the "PRAGMA foreign_keys = ON;" and "PRAGMA foreign_keys = OFF;" statements. The default value is True.
IntegerAsLargeInt	Maps INT (INTEGER) columns to TLargeIntField fields. If True, INT (INTEGER) columns are mapped to TLargeIntField fields. If False, INT (INTEGER) columns are mapped to TIntegerField fields. The default value is False.
JournalMode	The journal mode. jmDelete The rollback journal is deleted at the conclusion of each transaction. jmTruncate The rollback journal is stored in volatile RAM. It reduces disk I/O, but decreases database safety and integrity. If the application using SQLite crashes in the middle of a transaction, the database file may become corrupt. jmPersist The rollback journal file is not deleted when the transaction is committed. The header of the journal is filled with zeroes to prevent other connections rolling back from the journal. This mode optimizes performance on platforms where deleting or truncating a file is much more expensive than overwriting the first block of a file with zeros. jmMemory The rollback journal is stored in volatile RAM. This reduces the disk I/O, but decreases database safety and integrity. If the

	<p>application using SQLite crashes in the middle of a transaction in this mode, the database file is likely to become corrupt.</p> <p><code>jmWAL</code> A write-ahead log is used instead of a rollback journal to implement transactions. When data database is updated, the original content is preserved in the database file and the changes are appended in a separate WAL file. All the transactions that are appended in the WAL file are eventually transferred back into the original database.</p> <p><code>jmOff</code> The rollback journal is completely disabled. No rollback journal is created, thus there is no rollback journal to delete. The ROLLBACK command does not work -- it behaves in an undefined way. Avoid using the ROLLBACK command when the journal mode is disabled.</p> <p><code>jmDefault</code> The default value. If the database was previously opened in the WAL mode, then Default will open the database in the WAL mode; otherwise, the database will be opened in the Delete mode.</p>
<p>LockingMode</p>	<p>The database locking mode.</p> <p><code>lmExclusive</code> The database connection never releases file locks. The first time the database is read or written in this mode, a shared lock is obtained and held. Use this mode if you want to prevent other processes from accessing the database file, reduce the number of filesystem operations, or access WAL databases without using the shared memory.</p> <p><code>lmNormal</code> The database connection unlocks the database file at the conclusion of each read or write transaction. Note: Keep the default LockingMode=<code>lmExclusive</code> and Synchronous=<code>smOff</code> for the best performance.</p>
<p>NativeDate</p>	<p>If the option is set to True, the date and time values will be stored in the database in the native SQLite format, and when retrieved, they will be converted to the TDateTime type. If set to False, no conversion to the TDateTime type will be made. The default value is True.</p>
<p>ReadUncommitted</p>	<p>Enables or disables the Read Uncommitted isolation mode. A database connection in this mode doesn't attempt to acquire a read lock on the table before reading from it. This can lead to inconsistent query results if another database connection modifies data in the table while it is being read, but it also means that a read transaction opened by a connection in the Read Uncommitted mode can neither block nor be blocked by another</p>

	connection. The default value is False.
RealAsDouble	<p>Is used to specify whether REAL columns are mapped either to TFloatField fields or to TSingleField fields.</p> <p>True REAL columns are mapped to TFloatField fields.</p> <p>False REAL columns are mapped to TSingleField fields.</p> <p>The default value is False.</p>
Synchronous	<p>The database synchronization mode when writing to disk.</p> <p>smOff The database engine continues without syncing after handing data off to the operating system. If the application running SQLite crashes, the data will safe, unless the operating system crashes or the computer loses power before data has been written to disk, in which case the database might become corrupted. This is the fastest mode.</p> <p>smNormal The database engine still syncs at the most critical moments, but less often than in the FULL mode. The Normal mode is faster than the Full mode. When using the WAL mode (and probably the DELETE mode) with synchronous=NORMAL, data is safe from corruption. The synchronous=NORMAL setting is a reasonable choice for most applications running in the WAL mode.</p> <p>smFull The database engine ensures that all content is safely written to disk before continuing. This preserves database integrity even in case of an operating system failure or power outage. It is a safe, but slower mode, and is most commonly used when not in the WAL mode.</p> <p>smExtra This mode is similar to the FULL mode, but in the DELETE mode, the directory containing the rollback journal is synced after that journal is unlinked to commit a transaction. This provides additional durability if a power outage occurs right after the commit.</p> <p>Note: Keep the default Synchronous=smOff and LockingMode=lmExclusive for the best performance.</p>
TimeFormat	The format for storing time in the database. If no format is specified, the default format hh24:mi:ss will be used. The default value is an empty string.
UseUnicode	Enables or disables Unicode support. When set to True, all character data is stored as WideString, and TStringField is used instead of TWideStringField. The default value is False.

TUniSQL

The TUniSQL component has no SQLite-specific options.

TUniQuery, TUniTable, TUniStoredProc

Option	Description
AdvancedTypeDetection	When <code>False</code> , standard metadata retrieval is performed when detecting the field type in a database. When <code>True</code> , a number of records will be prefetched from a table, and the field type will be detected based on the type of data stored in the corresponding column in the table. The default value is <code>False</code> .
FetchAll	When <code>True</code> , all records of a query are requested from the database server when opening the dataset. If <code>False</code> , records are retrieved when a data-aware component or a program requests it. The default value is <code>False</code> .
ExtendedFieldsInfo	If <code>True</code> , the driver performs additional queries to the database when opening a dataset. These queries return information about which fields of the dataset are required or autoincrement. Set this option to <code>True</code> , if you need the <code>Required</code> property of fields be set automatically.
UnknownAsString	If set to <code>True</code> , all SQLite data types that are fetched as text and don't have the size limit, are mapped to <code>TStringField</code> with the default size 8192 bytes. If <code>False</code> (default value), such types are mapped to <code>TMemoField</code> . The <code>TEXT</code> data type is always mapped to <code>TMemoField</code> regardless of the value of this option.

TUniScript

The TUniScript component has no SQLite-specific options.

TUniLoader

Option	Description
AutoCommit	Used to automatically perform a commit after loading a certain amount of records. When the property is set to <code>True</code> , a transaction implicitly starts before loading the block of records and commits automatically after records were loaded. The default value is <code>True</code> .
AutoCommitRowCount	Use the <code>AutoCommitRowCount</code> property to specify the number of records, after which the transaction will be committed automatically when the <code>TUni.AutoCommit</code> property is set to <code>True</code> . The default value is 1000.
QuoteNames	Use the <code>QuoteNames</code> option to quote all database object names in automatically generated SQL statements, such as <code>UPDATE</code> statements. The default value is <code>False</code> .

TUniDump

The TUniDump component has no SQLite-specific options.

Data Type Mapping

The following table lists the constants for mapping SQLite data types to Delphi data types.

See [Data Type Mapping](#) for more information.

Constant	Description
liteInteger	Maps INTEGER to Delphi data types.
liteReal	Maps REAL to Delphi data types.
liteText	Maps TEXT to Delphi data types.
liteBlob	Maps BLOB to Delphi data types.
liteNull	Maps NULL to Delphi data types.
liteBit	Maps BIT to Delphi data types.
liteTinyInt	Maps TINYINT to Delphi data types.
liteSmallInt	Maps SMALLINT to Delphi data types.
liteInt2	Maps INT2 to Delphi data types.
liteInt	Maps INT to Delphi data types.
liteMediumInt	Maps MEDIUMINT to Delphi data types.
liteBigInt	Maps BIGINT to Delphi data types.
liteUBigInt	Maps UNSIGNED BIG INT to Delphi data types.
liteInt8	Maps INT8 to Delphi data types.
liteInt64	Maps INT64 to Delphi data types.
liteChar	Maps CHAR to Delphi data types.
liteVarChar	Maps VARCHAR to Delphi data types.
liteClob	Maps CLOB to Delphi data types.
liteFloat	Maps FLOAT to Delphi data types.
liteDouble	Maps DOUBLE to Delphi data types.
liteNumeric	Maps NUMERIC to Delphi data types.
liteDecimal	Maps DECIMAL to Delphi data types.
liteNumber	Maps NUMBER to Delphi data types.
liteMoney	Maps MONEY to Delphi data types.
liteBool	Maps BOOLEAN to Delphi data types.
liteBinary	Maps BINARY to Delphi data types.
liteVarBinary	Maps VARBINARY to Delphi data types.
liteDate	Maps DATE to Delphi data types.
liteTime	Maps TIME to Delphi data types.
liteDateTime	Maps DATETIME to Delphi data types.

liteTimestamp	Maps TIMESTAMP to Delphi data types.
liteTimestampTZ	Maps TIMESTAMP WITH TIME ZONE to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.14.2 Database File Encryption

What constitutes Database File Encryption

The SQLite architecture provides the functionality for work with encrypted databases. This means that encoding/decoding is applied to a database file, in the moment of execution of the file read/write operations. This is a low-level encryption "on the fly", it is implemented at the level of the SQLite client library and is completely transparent to the applications working with the database.

But, the fact is that in the client libraries available at the official SQLite website, the algorithms of database file encryption are not implemented. Therefore, usually, to work with encrypted databases one has to either use a custom-built client library with encryption support, or create an own library from the source code, available on the SQLite website.

UniDAC functionality for Database File Encryption

UniDAC provides built-in capabilities for Database File Encryption, which becomes available when working in [Direct mode](#). Database File Encryption, built in UniDAC, allows to:

- encrypt a database;
- create a new encrypted database;
- connect and work with the encrypted database;
- change the encryption key of the encrypted database;
- decrypt the encrypted database.

To encrypt/decrypt the database file, one of the following encryption algorithms can be used:

- the Triple DES encryption algorithm;
- the Blowfish encryption algorithm;
- the AES encryption algorithm with a key size of 128 bits;

- the AES encryption algorithm with a key size of 192 bits;
- the AES encryption algorithm with a key size of 256 bits;
- the Cast-128 encryption algorithm;
- the RC4 encryption algorithm.



Note: There are no strict standard requirements for implementing database file encryption in SQLite. In UniDAC, the implementation of database file encryption is compatible with the following Devart products:

- [LiteDAC](#)
- [ODBC Driver for SQLite](#)
- [dbExpress Driver for SQLite](#)
- [Python Connector for SQLite](#)
- [dotConnect for SQLite](#)
- [Excel Add-In for SQLite](#)
- [SSIS Data Flow Components for SQLite](#)

UniDAC can only work with databases that are encrypted by the driver itself or the products listed above. In turn, no third-party application can operate with encrypted databases created by UniDAC or the products mentioned above.

The difference between Database File Encryption and Data Encryption.

The functionality of [Data Encryption](#), which is realized with the help of the [TUniEncryptor](#) component, allows to encrypt individual fields in database tables. In this case, the database itself is not encrypted. I.e. on the one hand, the information in this database (with the exception of the encrypted fields) is easily accessible for viewing by any SQLite DB-tools. On the other hand, such database is more simple in terms of modification of data structures.

Database File Encryption encrypts all the data file. Both structure and information on such database becomes unavailable for any third-party applications. An indisputable advantage is the increased level of secrecy of information. The disadvantage is that, for making any changes in the structure of the database, developers will have to use only UniDAC.

Both Database File Encryption and Data Encryption methods are not mutually exclusive and can be used at the same time.

The usage of Database File Encryption in UniDAC

To control database encryption in UniDAC, the following properties and methods of the [TUniConnection](#) component are used:

- The `TUniConnection.Options.EncryptionAlgorithm` property - specifies the encryption algorithm that will be used to connect to an encrypted database, or to create a new encrypted database.
- The `TUniConnection.EncryptionKey` property - specifies the encryption key that will be used to connect to an encrypted database, or to create a new encrypted database.
- The `TUniConnection.EncryptDatabase` method - is used to change the encryption key in an encrypted database, or to decrypt the database.

Encrypt a database

The following example shows how to encrypt an existing database:

```
UniConnection.Database := 'C:\sqlite.db3';
UniConnection.SpecificOptions.Values['ForceCreateDatabase'] := 'False';
UniConnection.SpecificOptions.Values['Direct'] := 'True';
UniConnection.SpecificOptions.Values['EncryptionAlgorithm'] := '1eBlowfish';
UniConnection.SpecificOptions.Values['EncryptionKey'] := '';
UniConnection.Open;
TLiteUtils.EncryptDatabase(UniConnection, '11111');
```

Creating of a new encrypted database

The following example shows creating a new encrypted database:

```
UniConnection.Database := 'C:\sqlite_encoded.db3';
UniConnection.SpecificOptions.Values['ForceCreateDatabase'] := 'True';
UniConnection.SpecificOptions.Values['Direct'] := 'True';
UniConnection.SpecificOptions.Values['EncryptionAlgorithm'] := '1eBlowfish';
UniConnection.SpecificOptions.Values['EncryptionKey'] := '11111';
UniConnection.Open;
```

Connecting to an encrypted database

To connect to an existing encrypted database, the following should be performed:

```
UniConnection.Database := 'C:\sqlite_encoded.db3';
UniConnection.SpecificOptions.Values['ForceCreateDatabase'] := 'False';
```

```
UniConnection.SpecificOptions.Values['Direct'] := 'True';  
UniConnection.SpecificOptions.Values['EncryptionAlgorithm'] := '1eB1owfish';  
UniConnection.SpecificOptions.Values['EncryptionKey'] := '11111';  
UniConnection.Open;
```

Changing the encryption key for the database

To change the encryption key in the encrypted database, you must perform the following:

```
UniConnection.Database := 'C:\sqlite_encoded.db3';  
UniConnection.SpecificOptions.Values['ForceCreateDatabase'] := 'False';  
UniConnection.SpecificOptions.Values['Direct'] := 'True';  
UniConnection.SpecificOptions.Values['EncryptionAlgorithm'] := '1eB1owfish';  
UniConnection.SpecificOptions.Values['EncryptionKey'] := '11111';  
UniConnection.Open;  
TLiteUtils.EncryptDatabase(UniConnection, '22222');
```

After changing the encryption key, the database connection remains open and the further work with the database can continue. However, if disconnected from the database and for subsequent connection, the new value of the encryption key should be assigned to the `UniConnection.EncryptionKey` property.

Decryption of the database

The encrypted database can be decrypted, after that it becomes available for viewing and editing in third-party applications. To decrypt the database you must first connect to it, as shown in the examples above, and then execute the `UniConnection.EncryptDatabase("")` method, specifying an empty string as a new key.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.15 UniDAC and PostgreSQL

5.1.15.1 PostgreSQL Provider

This article provides a brief overview of the PostgreSQL data access provider for UniDAC used to establish a connection to PostgreSQL databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)

- [PostgreSQL-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [Data Type Mapping](#)

Overview

Main features of PostgreSQL data access provider are:

- Direct access to server without PostgreSQL client library
- High performance
- Easy deployment
- Comprehensive support for the latest versions of PostgreSQL server

The full list of PostgreSQL provider features can be found on the UniDAC [Features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the PostgreSQL provider. For Express Edition of UniDAC, the PostgreSQL provider can be installed with PostgreSQL Data Access Components (PgDAC).

Compatibility

To learn about PostgreSQL database server compatibility, refer to the [Compatibility](#) section.

Requirements

The provider does not require installation of any additional software on the client.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more

information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

PostgreSQL-specific options

Though UniDAC is components that provide unified interface to work with different database servers, it also lets you tune behaviour for each server individually. For thin setup of a certain database server, UniDAC provides server-specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list. Therefore you can use the following syntax to assign an option value:

```
UniConnection.SpecificOptions.Values['CharLength'] := '1';
```

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
ApplicationName	The name of a client application. The default value is the name of the executable file of your application.
Charset	Setups the character set which will be used to transfer character data between client and server.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
HttpPassword	Use the HttpPassword option to specify the password for HTTP authorization.
HttpTrustServerCertificate	This option specifies whether or not the driver should trust the server certificate when connecting to the server. The default value is False – the driver won't trust the server certificate and will verify validity of the server certificate instead. If set to True, the driver will trust the server certificate.
HttpUrl	Use the HttpUrl option to specify the URL of the PHP tunneling script.
HttpUsername	Use the HttpUsername option to specify the username for HTTP authorization.
IPVersion	Use the IPVersion property to specify Internet Protocol Version. Supported values: ivIPBoth Specifies that either Internet Protocol Version 6 (IPv6) or Version

	<p>4 (IPv4) will be used.</p> <p>ivIPv4 (default) Specifies that Internet Protocol Version 4 (IPv4) will be used.</p> <p>ivIPv6 Specifies that Internet Protocol Version 6 (IPv6) will be used.</p> <p>Note: When the TIPVersion property is set to ivIPBoth, a connection attempt will be made via IPv6 if it is enabled on the operating system. If the connection attempt fails, a new connection attempt will be made via IPv4.</p>
MessagesCharset	Specifies the character set that will be used to transfer error messages from the server to the client.
MultipleConnections	Enables or disables the creation of an additional internal connection for TUniQuery, when necessary. The default value is True.
ProtocolVersion	<p>Specifies protocol version to be used when several versions are available.</p> <p>Supported values:</p> <p>pv20 Set ProtocolVersion to pv20 to work with PostgreSQL server version 7.3 or older that don't support protocol version 3.0.</p> <p>pv30 Set ProtocolVersion to pv30 to enforce protocol version 3.0.</p> <p>pvAuto (default) Set ProtocolVersion to pvAuto to automatically select between protocol versions depending on the specific query for the best possible performance.</p>
ProxyHostname	Use the ProxyHostName option to specify the host name or IP address to connect to the proxy server.
ProxyPassword	Use the ProxyPassword option to specify the password for the proxy server.
ProxyPort	Use the ProxyPort option to specify the port for a TCP/IP connection with the proxy server.
ProxyUsername	Use the ProxyUsername option to specify the username for the proxy server.
Schema	Use the Schema property to set the search path for the connection to the specified schema. This setting offers a convenient way to perform operations on objects in a schema other than that of the current user without having to qualify the objects with the schema name.

SSLCACert	The pathname to the certificate authority file.
SSLCert	The pathname to the certificate file.
SSLCipherList	The list of allowable ciphers to use for SSL encryption.
SSLKey	The pathname to the key file.
SSLMode	<p>This option determines whether or with what priority an SSL connection will be negotiated with the server.</p> <p>Supported values:</p> <p>smAllow Negotiates trying first a non-SSL connection, then if that fails, tries an SSL connection.</p> <p>smDisable (default) Only an unencrypted SSL connection will be attempted.</p> <p>smPrefer Negotiates trying first an SSL connection, then if that fails, tries a regular non-SSL connection.</p> <p>smRequire Tries only an SSL connection.</p> <p>smVerifyCA Verifies server identity by validating the server certificate chain up to the root certificate installed on the client machine.</p> <p>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.</p> <p>Note: 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.</p>
UseHttp	The UseHttp option enables the use of HTTP tunneling to connect to the server. The default value is False.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.
UuidWithBraces	Use the UuidWithBraces option to specify whether the values of UUID fields are returned with braces. The default value is True.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
UnpreparedExecute	If 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.
UseParamTypes	Set this 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.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
AutoDeleteBlob	If True (the default value), the BLOBs are deleted from database automatically when a record that holds these BLOBs' OIDs is deleted from dataset.
CacheBlobs	If True (the default value), then local memory buffer is allocated to hold a copy of the BLOB content.
CommandTimeout	The time to wait for a statement to execute.
CursorWithHold	When this option is False (default), an active transaction is required to open a query in 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.
DeferredBlobRead	If True, all BLOB values are fetched only when they are explicitly requested. Otherwise entire record set with any BLOB values is returned when dataset is opened. Whether BLOB values are cached locally to be reused later is controlled by the CacheLobs option.
ExtendedFieldsInfo	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. The default value is True.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is True.

KeySequence	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 the database.
OIDAsInt	If True, OID fields are mapped on TIntegerField. If False, values of OID fields are treated as large objects' OID, and these fields are mapped on TBlobField.
PrefetchRows	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 and 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.
SequenceMode	Set the SequenceMode property to specify which method is used internally to generate sequenced field. The following values are allowed for this property: smInsert New record is inserted into the dataset with the first key field populated with a sequenced value. Application may modify this field before posting the record to the database. smPost Database server populates key field with a sequenced value when application posts the record to the database. Any value put into the key field before post will be overwritten.
UnknownAsString	If True, all PostgreSQL data types that are fetched as text, and don't have limited field size, are mapped on TStringField with default size 8192. If False, such types are mapped on TMemoField. The TEXT data type is always mapped on TMemoField regardless of this option.
UnpreparedExecute	If 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.
UseParamTypes	Set this 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.

TUniScript

The TUniScript component has no PostgreSQL-specific options.

TUniLoader

Option name	Description
BufferSize	This property contains the size of the memory buffer used by TPgLoader. When buffer is filled, the loader sends block of data to the server.
TextMode	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.
QuoteNames	Use the QuoteNames option to quote all database object names in automatically generated SQL statements, such as UPDATE statements. The default value is False.

TUniDump

The TUniDump component has no PostgreSQL-specific options.

Data Type Mapping

The following table lists the constants for mapping PostgreSQL data types to Delphi data types. See the [Data Type Mapping](#) tutorial for more information.

Constant	Description
pgBigInt	Maps bigint to Delphi data types.
pgBigSerial	Maps bigserial to Delphi data types.
pgBit	Maps bit to Delphi data types.
pgBitVarying	Maps bit varying to Delphi data types.
pgBoolean	Maps boolean to Delphi data types.
pgBytea	Maps bytea to Delphi data types.
pgCharacter	Maps character to Delphi data types.
pgCharacterVarying	Maps character varying to Delphi data types.
pgDate	Maps date to Delphi data types.
pgDoublePrecision	Maps double precision to Delphi data types.
pgInteger	Maps integer to Delphi data types.
pgMoney	Maps money to Delphi data types.
pgNumeric	Maps numeric to Delphi data types.
pgReal	Maps real to Delphi data types.
pgSerial	Maps serial to Delphi data types.
pgSmallint	Maps smallint to Delphi data types.

pgText	Maps text to Delphi data types.
pgTime	Maps time to Delphi data types.
pgTimeStamp	Maps timestamp to Delphi data types.
pgTimeStampWithTimeZone	Maps timestamp with time zone to Delphi data types.
pgTimeWithTimeZone	Maps time with time zone to Delphi data types.
pgUUID	Maps uuid to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.15.2 Connecting via SSL

Security is critical when exchanging messages between the server and client. Data protection can be achieved through several methods, including establishing an SSL-encrypted connection from a Delphi application to a remote PostgreSQL server. PostgreSQL supports data transfer over TCP/IP, both with or without SSL encryption.

You can establish a secure connection to a PostgreSQL server using [SecureBridge](#), a solution offered by Devart. It allows you to embed an SSL client into a Delphi or C++ Builder application.

This page demonstrates how to connect to PostgreSQL with UniDAC using SSL as the encryption method.

- [Connect Using SecureBridge](#)
- [Connect Using the OpenSSL library](#)

Connect Using SecureBridge

To connect to PostgreSQL using UniDAC and SecureBridge components, no external libraries are required. SecureBridge provides the necessary SSL functionality directly within the components.

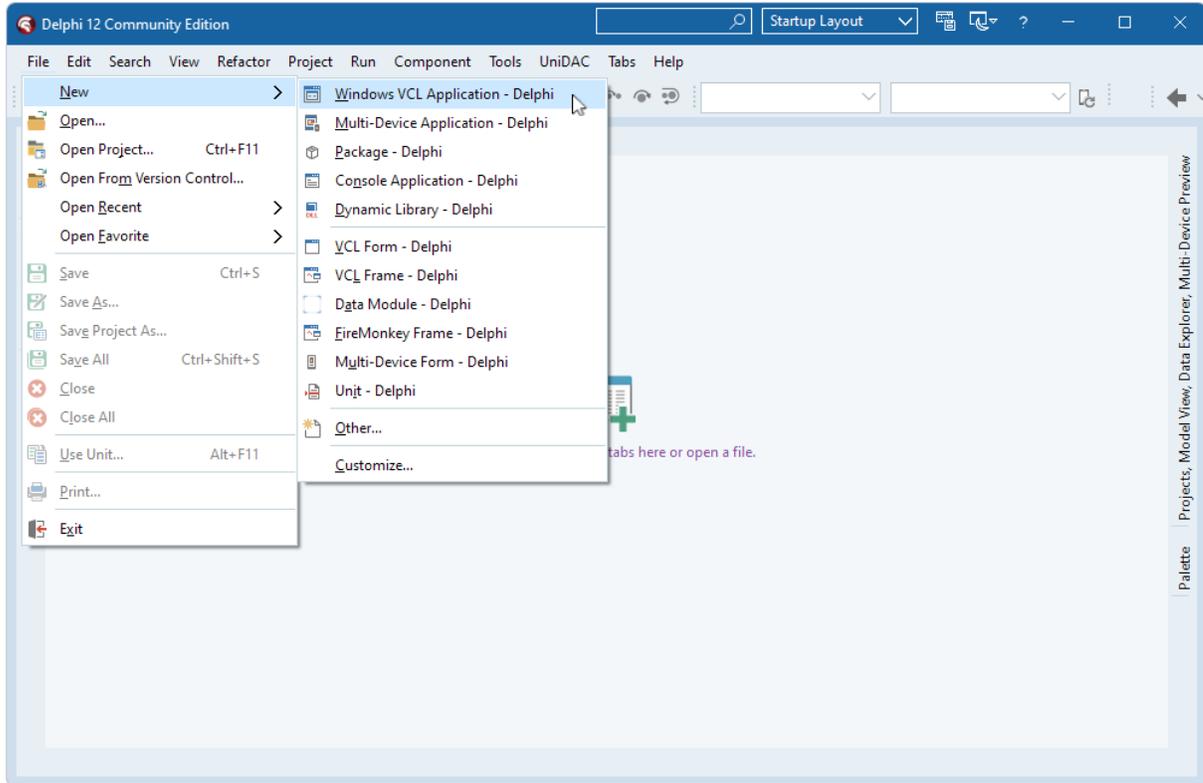
Prerequisites:

- Install SecureBridge.
- Install the TCRSSLIOHandler component in RAD Studio to bind UniDAC with SecureBridge.

For instructions, see [Readme.html](#)—by default, it is located in `C:\Users\\Documents`

```
\Devart\UniDAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge.
```

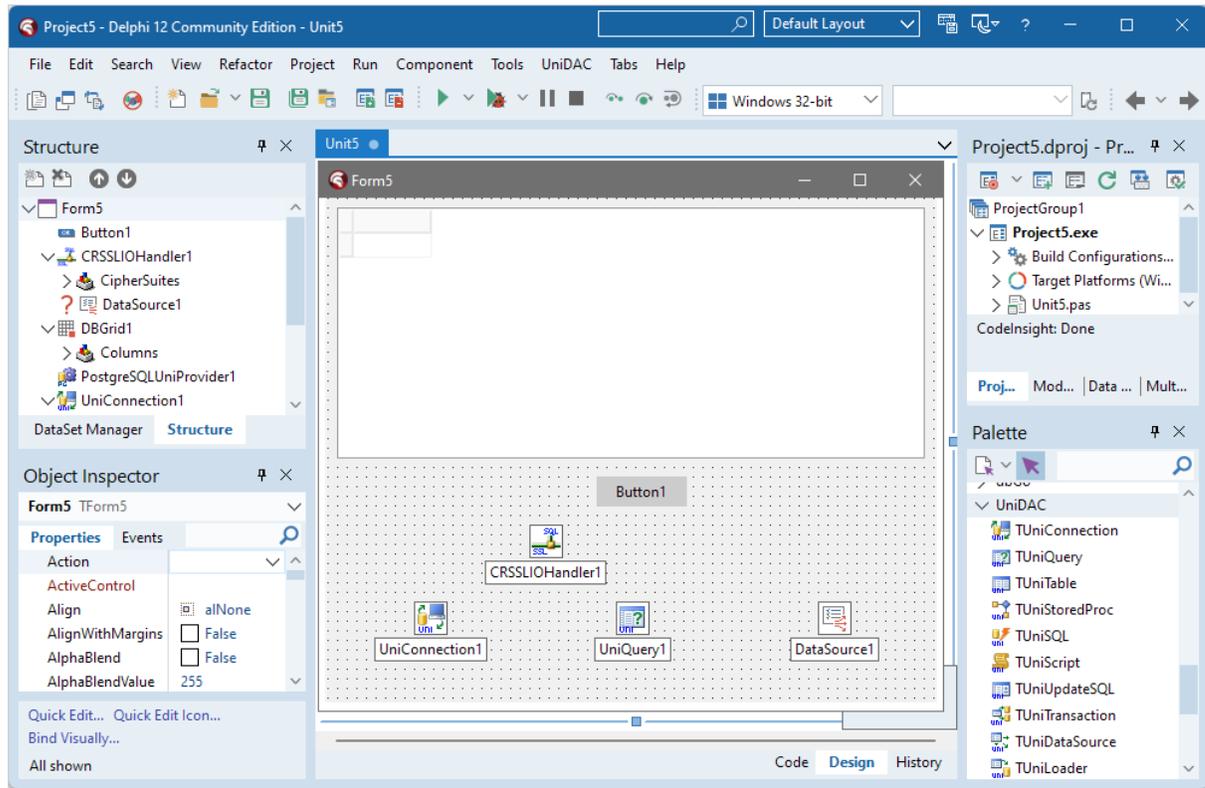
1. In RAD Studio, select **File > New > Windows VCL Application - Delphi**.



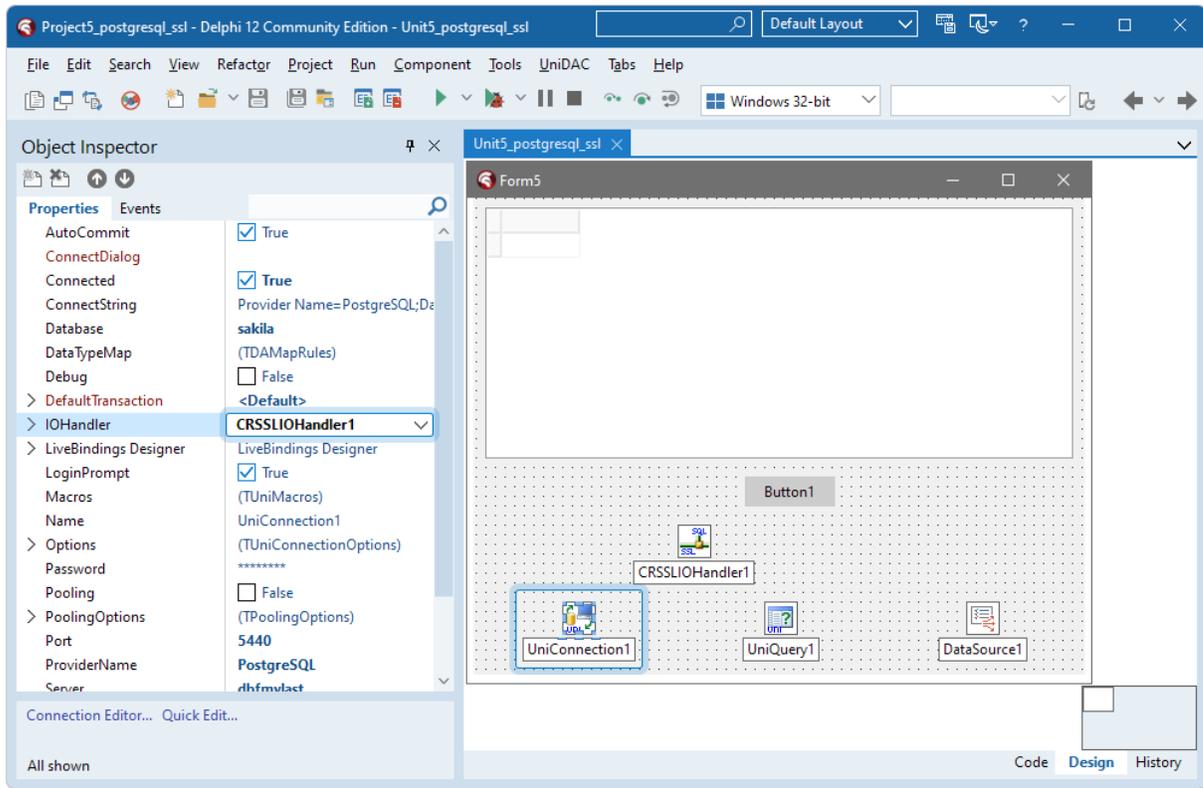
2. Place the following components from **Palette** on the form:

- **TCRSSLIHandler**
- **TUniConnection**
- **TUniQuery**
- **TDataSource**
- **TDBGrid**
- **TButton**

These components are required to establish an SSL connection to a PostgreSQL server using UniDAC, execute a query, and display the retrieved data in a grid.

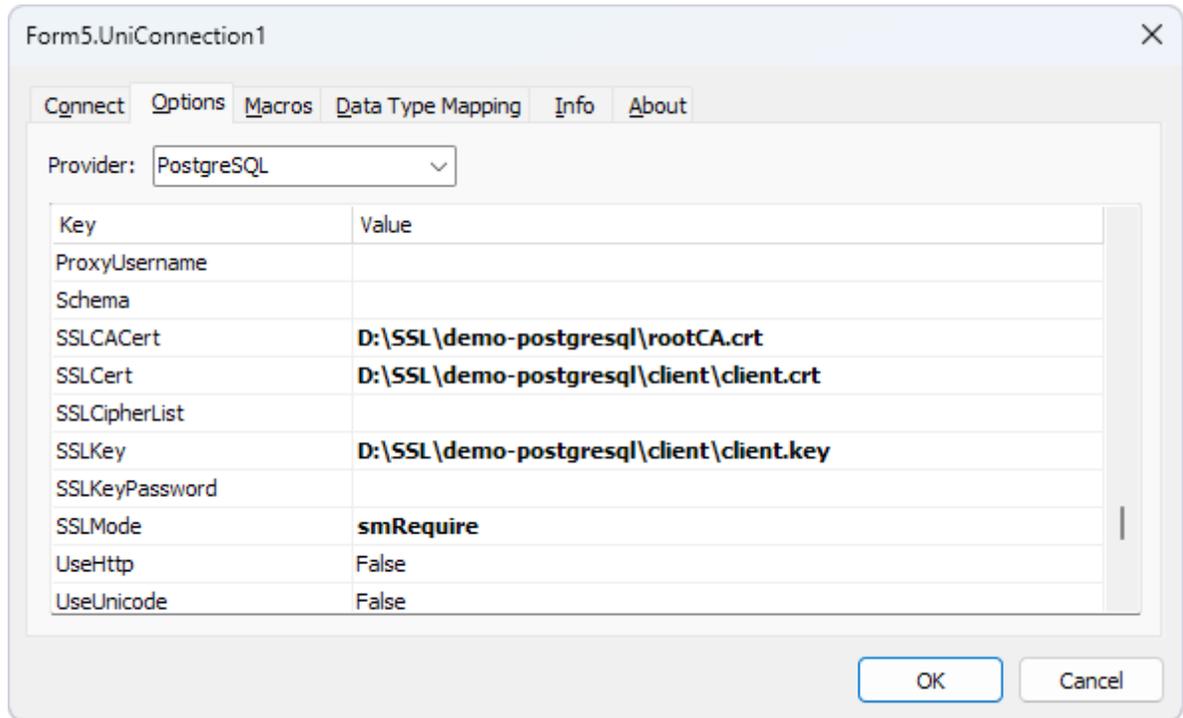


3. Select the **TUniConnection** component and, in **Object Inspector**, set the **IOHandler** property to the instance of **TCRSSLIHandler (CRSSLIHandler1)**.



4. Double-click the **TUniConnection** component and, on the **Options** tab, fill out the fields:

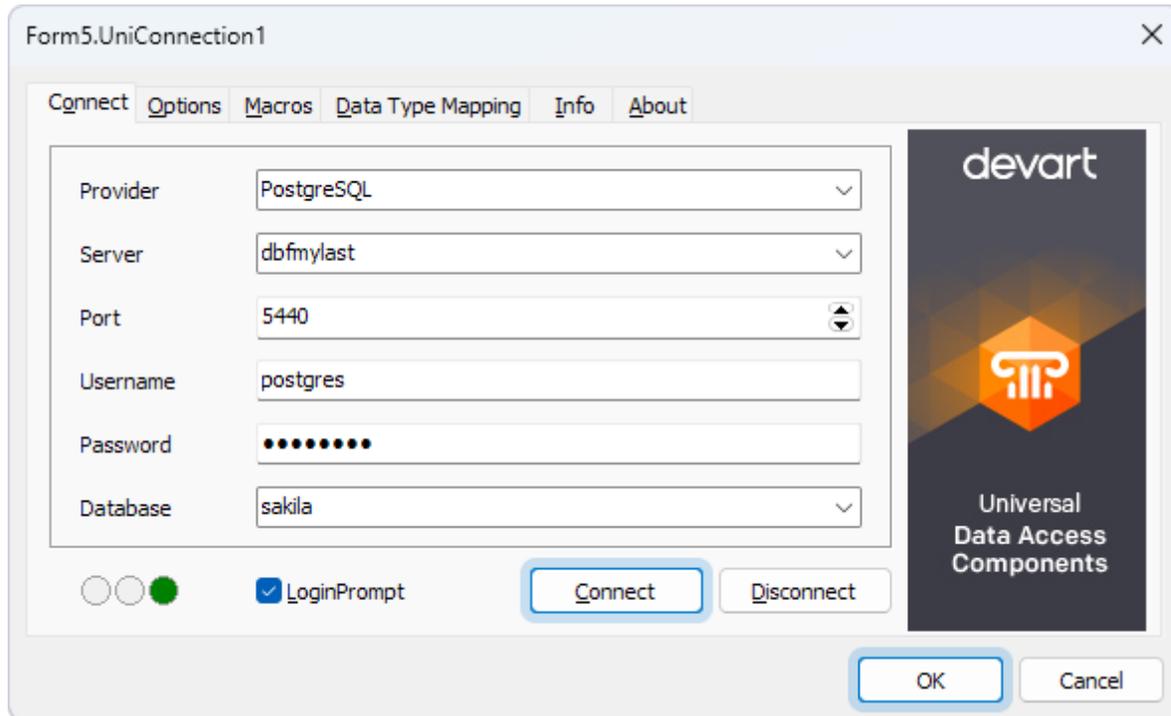
- **Provider** – Select **PostgreSQL**.
- **SSLCACert** – Specify the full path of the Certificate Authority certificate file.
- **SSLCert** – Specify the full path of the client certificate file.
- **SSLKey** – Specify the full path of the client private key file.
- **SSLMode** – Select **smRequire**. This value forces the application to connect only through an SSL connection. If a connection attempt fails, an exception is raised.



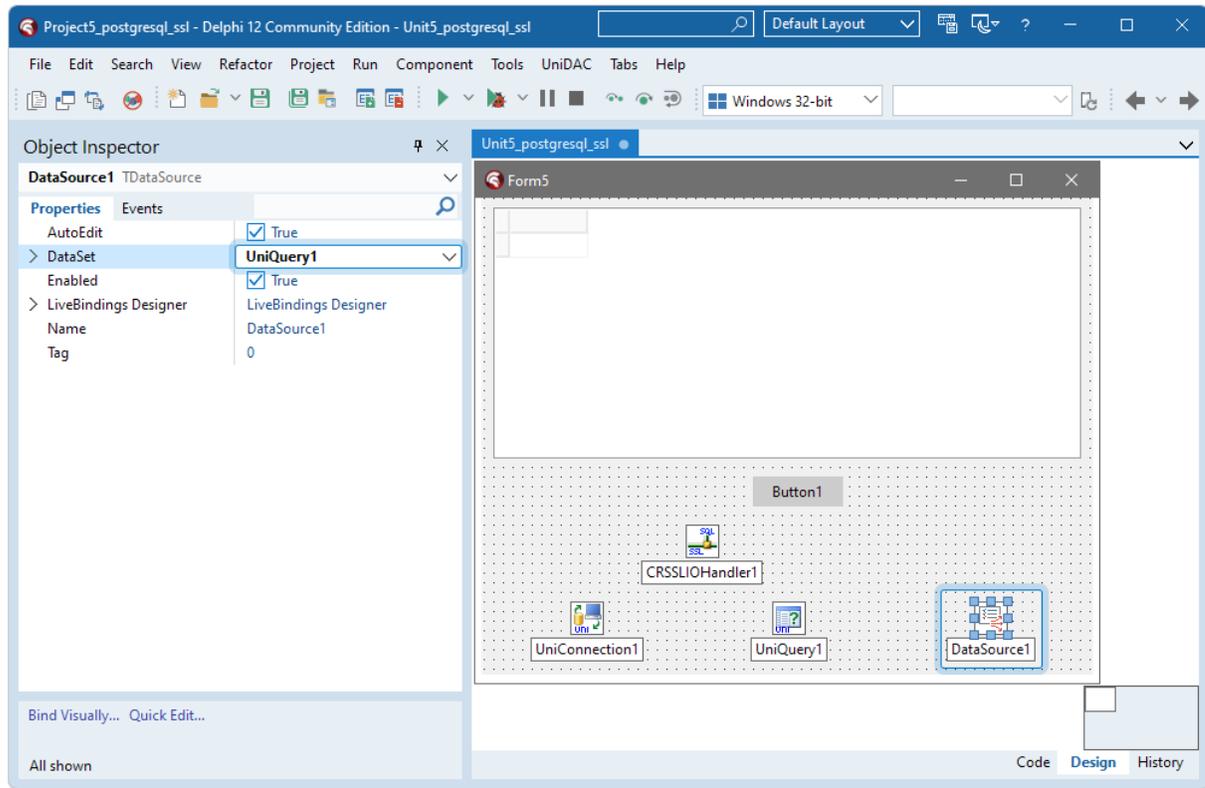
5. On the **Connect** tab, fill out the fields:

- **Provider** – Select **PostgreSQL**.
- **Server** – Enter the host name or IP address of the PostgreSQL server.
- **Port** – Specify the PostgreSQL server port.
- **Username** – Enter the username for the account on the PostgreSQL server.
- **Password** – Enter the password for the account on the PostgreSQL server.
- **Database** – Select the database.

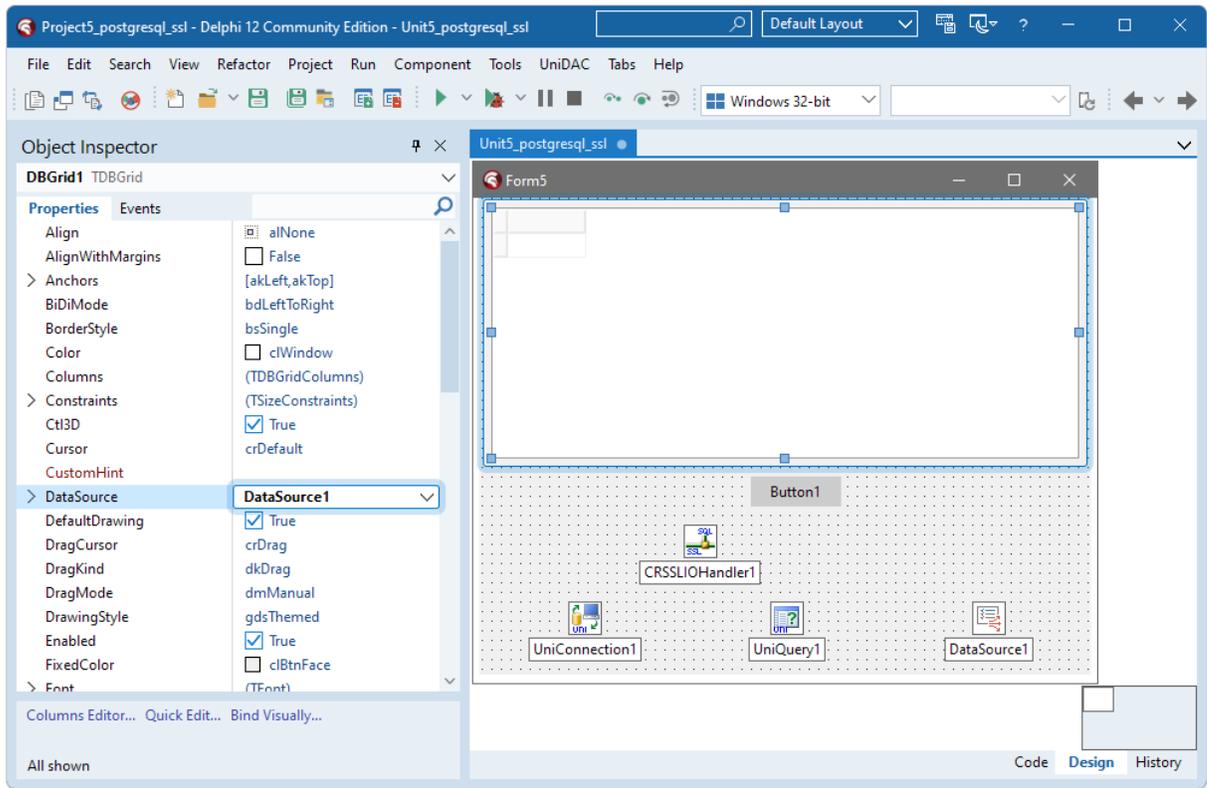
6. Click **Connect** to test the connection to the PostgreSQL server, then click **OK** to close the dialog.



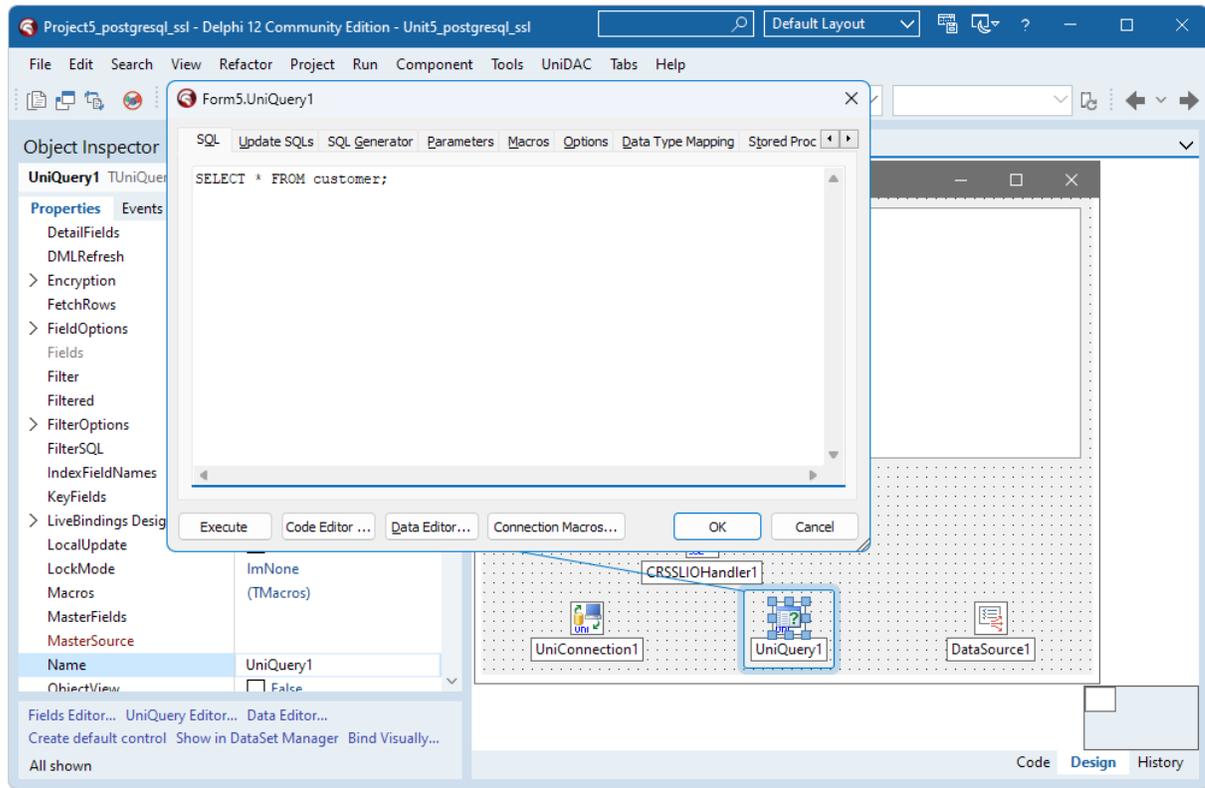
7. Select the **TDataSource** component and set the **DataSet** property to the instance of **TUniQuery (UniQuery1)**.



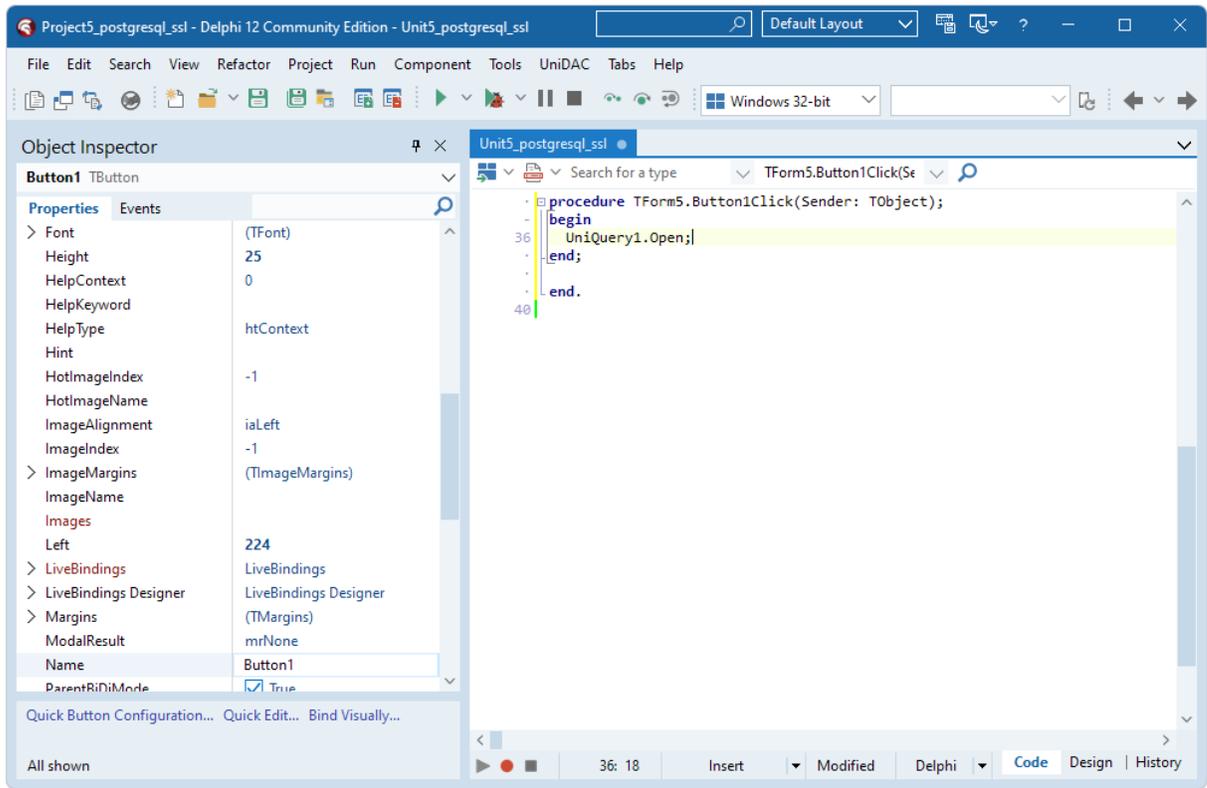
8. Select the **TDBGrid** component and set the **DataSource** property to the instance of **TDataSource (DataSource1)**.



9. Double-click the **TUniQuery** component, enter a SQL query to be run against the PostgreSQL database, and then click **OK**.

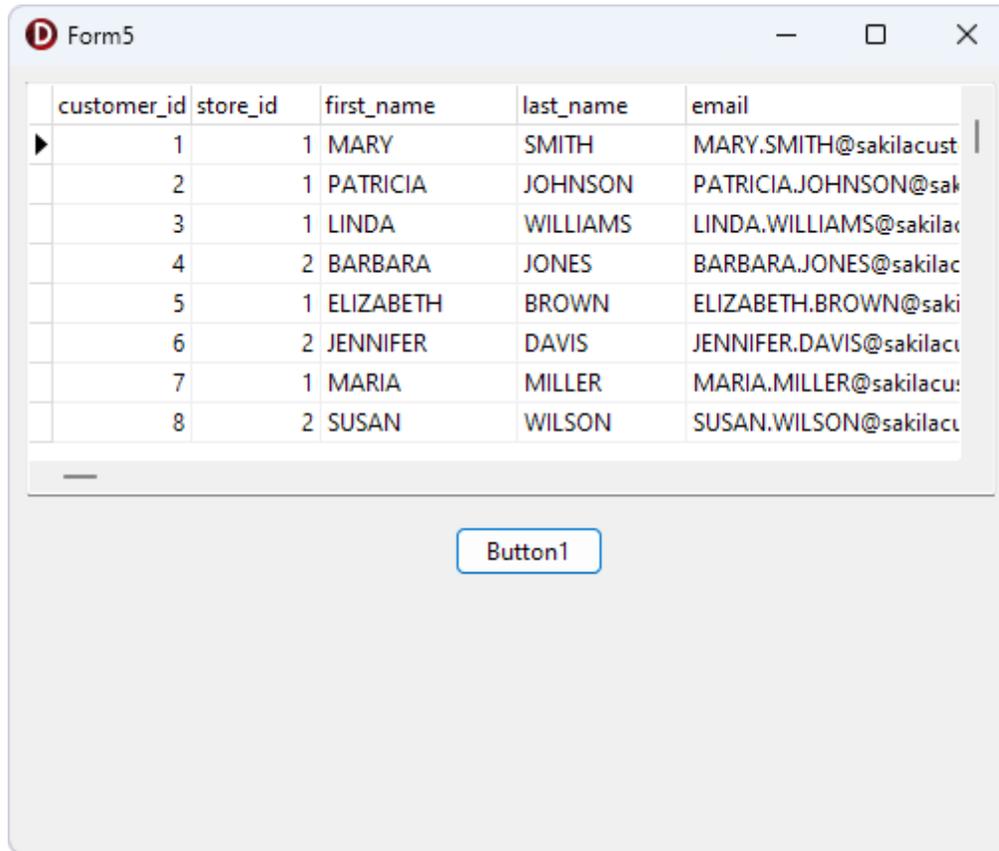


10. Double-click the **TButton** component and add code to call the **Open** method on the **UniQuery1** object to activate the dataset when the button is clicked.



11. Press **F9** to compile and run the application.

12. In the form that appears, click **Button1** to run the query. Data appears in the grid.



Connect Using the OpenSSL library

Another way to embed SSL client functionality into your Delphi app that uses UniDAC components to access PostgreSQL is by using the OpenSSL library. The OpenSSL library implements the SSL protocol and enables servers to communicate with their clients securely.

Important: The authenticity of the CA certificate is not verified when you use the OpenSSL library. UniDAC supports only the basic OpenSSL functionality, which does not include CA certificate verification. To configure advanced SSL features such as CA certificate verification, use [SecureBridge components](#) instead of OpenSSL.

To establish an SSL connection, set the following options:

- `SSLCACert` – The full path of the Certificate Authority certificate file.
- `SSLCert` – The full path of the client certificate file.
- `SSLKey` – The full path of the client private key file.

- `SSLCipherList` – A list of allowed ciphers to use for SSL encryption.

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.15.3 Connecting via SSH

SSH is a protocol that allows users to securely log in 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.

You can use SSH to secure the network connection between a Delphi application and a PostgreSQL server. An SSH connection enables you to run shell commands as if you were physically operating the remote machine.

This page demonstrates how to use UniDAC to connect to a PostgreSQL server securely via an SSH tunnel.

To follow the steps on this page, you need an SSH client and an SSH server. You can create them using [SecureBridge](#), a solution provided by Devart. It enables you to embed an SSH client into your Delphi application and, if needed, configure an SSH server. For more information, see the SecureBridge [tutorial on configuring an SSH server](#). You can also build the SSHServer [demo project](#), which is distributed with SecureBridge.

Your PostgreSQL server must be configured to accept connections from the SSH tunnel.

- [Connect Using SecureBridge](#)
- [Connect Using OpenSSH or Any Other Third-Party SSH Tunneling Tools](#)

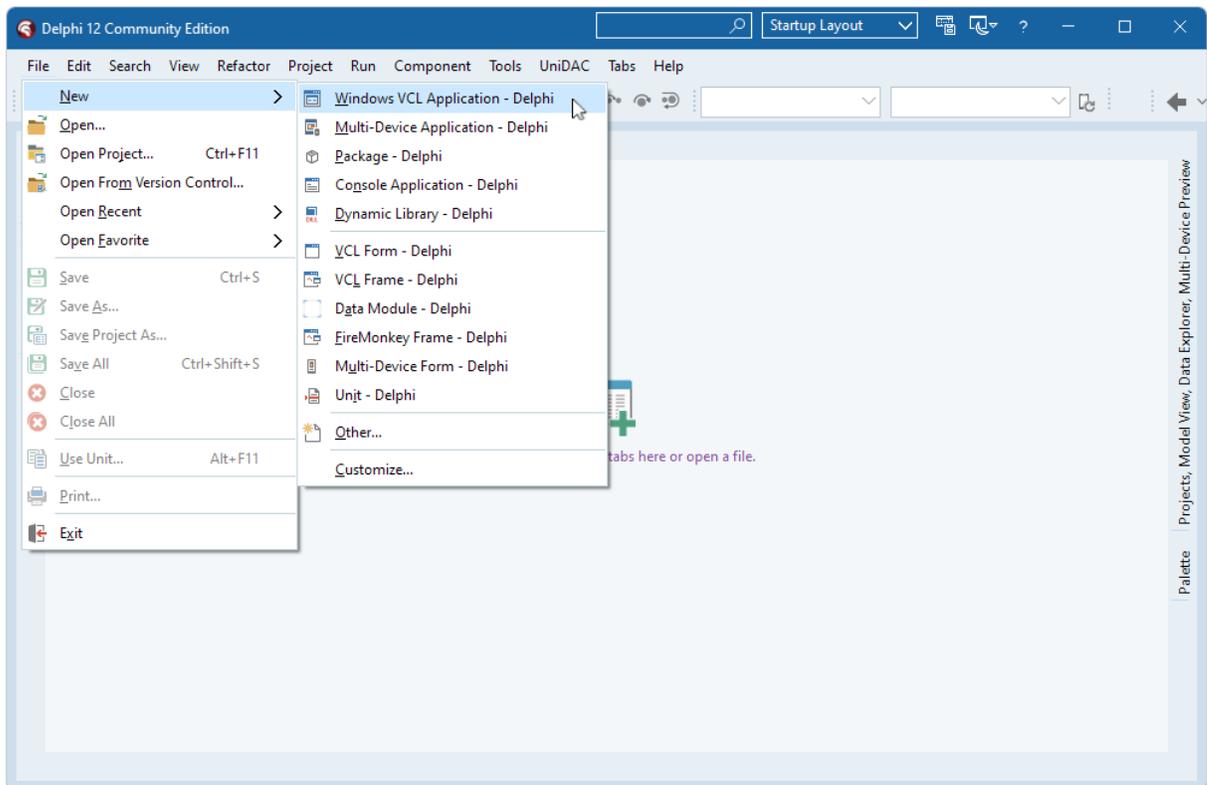
Connect Using SecureBridge

You can establish a secure SSH tunnel using SecureBridge without the need for third-party applications. SecureBridge provides all the necessary components to create an SSH client within your Delphi application.

Prerequisites:

- Install SecureBridge.
- Install the TCRSSHIOHandler component in RAD Studio to bind UniDAC with SecureBridge. For instructions, see `Readme.html`—by default, it is located in `c:\Users\\Documents\Devart\UniDAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge`.

1. In RAD Studio, select **File > New > Windows VCL Application - Delphi**.

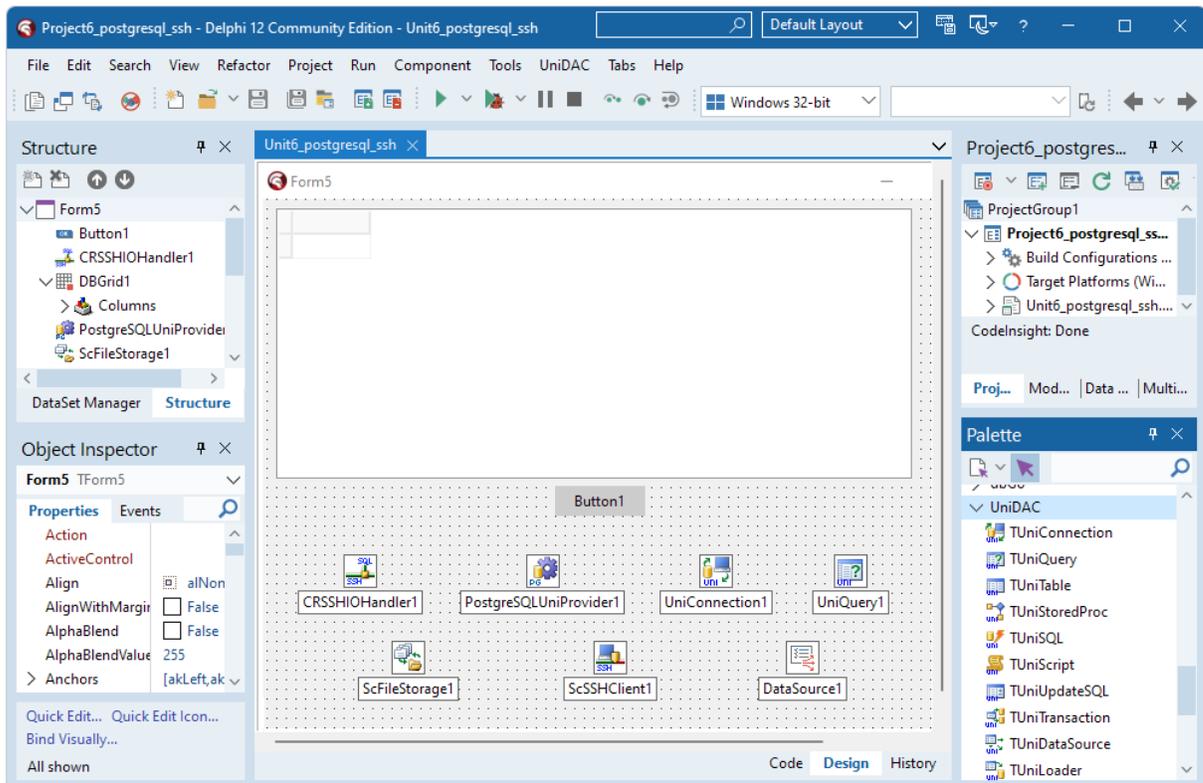


2. Place the following components from **Palette** on the form:

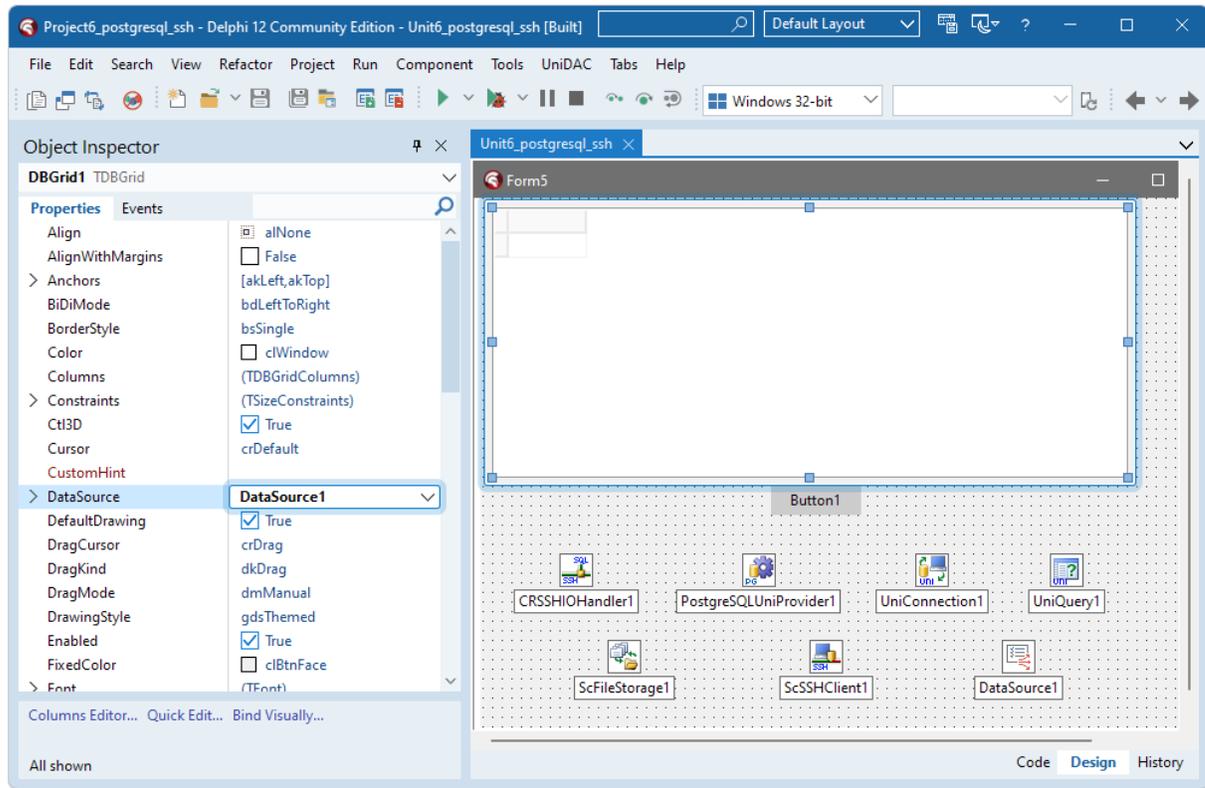
- **TCRSSHIOHandler**
- **TPostgreSQLUniProvider**
- **TUniConnection**
- **TUniQuery**
- **TScFileStorage**

- TScSSHClient
- TDataSource
- TDBGrid
- TButton

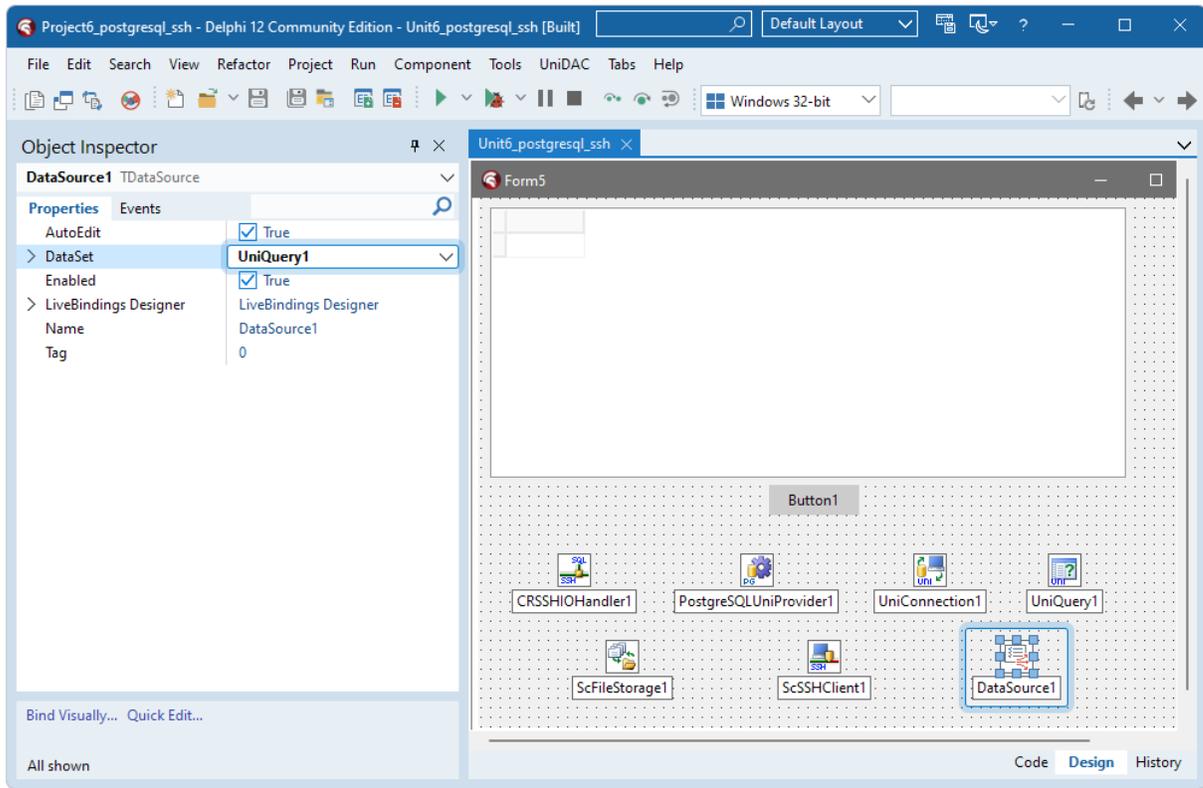
These components are required to establish an SSH connection to a PostgreSQL server using UniDAC, execute a query, and display the retrieved data in a grid.



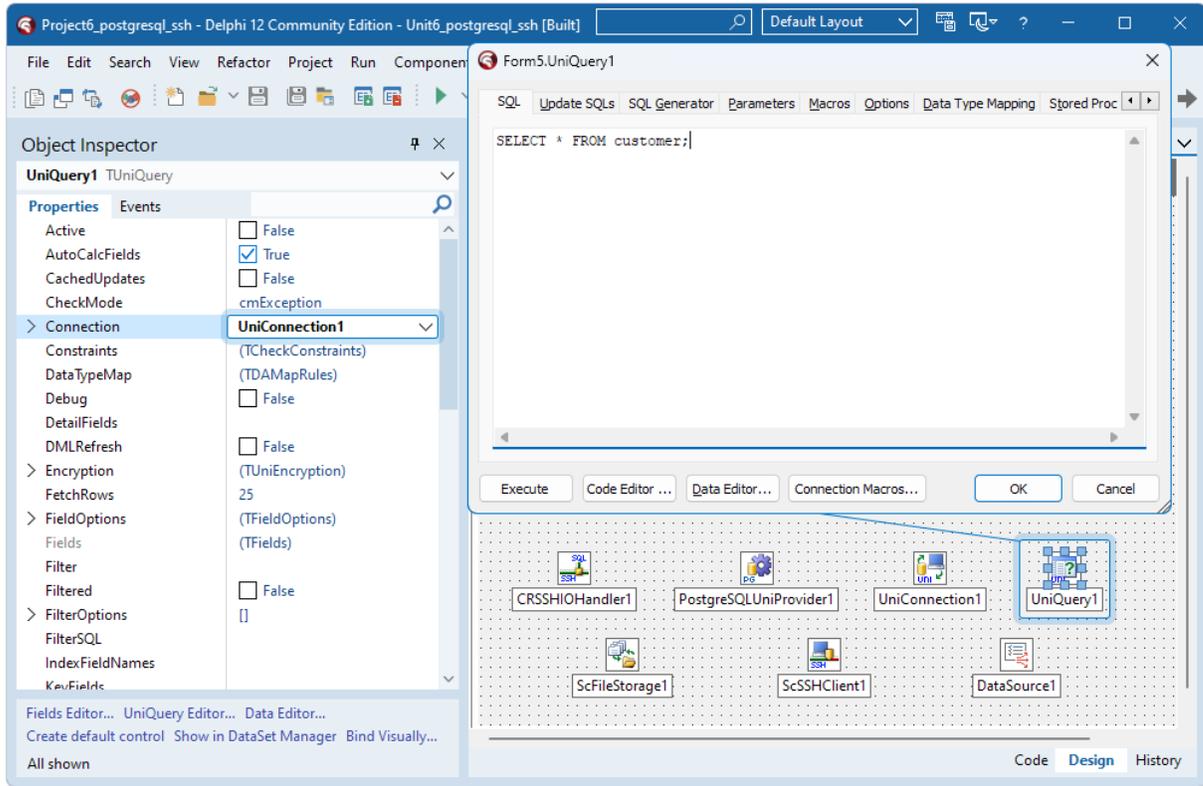
3. Select the **TDBGrid** component and, in **Object Inspector**, set the **DataSource** property to the instance of **TDataSource (DataSource1)**.



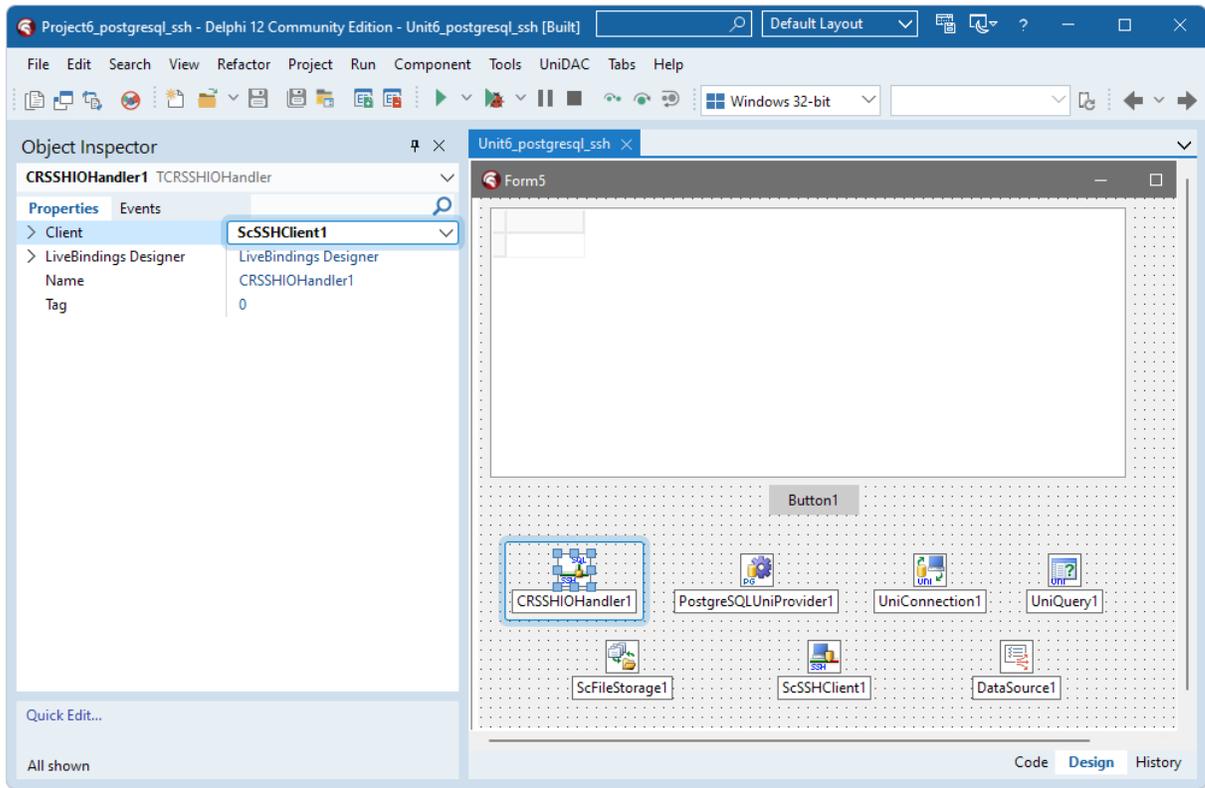
4. Select the **TDataSource** component and set the **DataSet** property to the instance of **TUniQuery (UniQuery1)**.



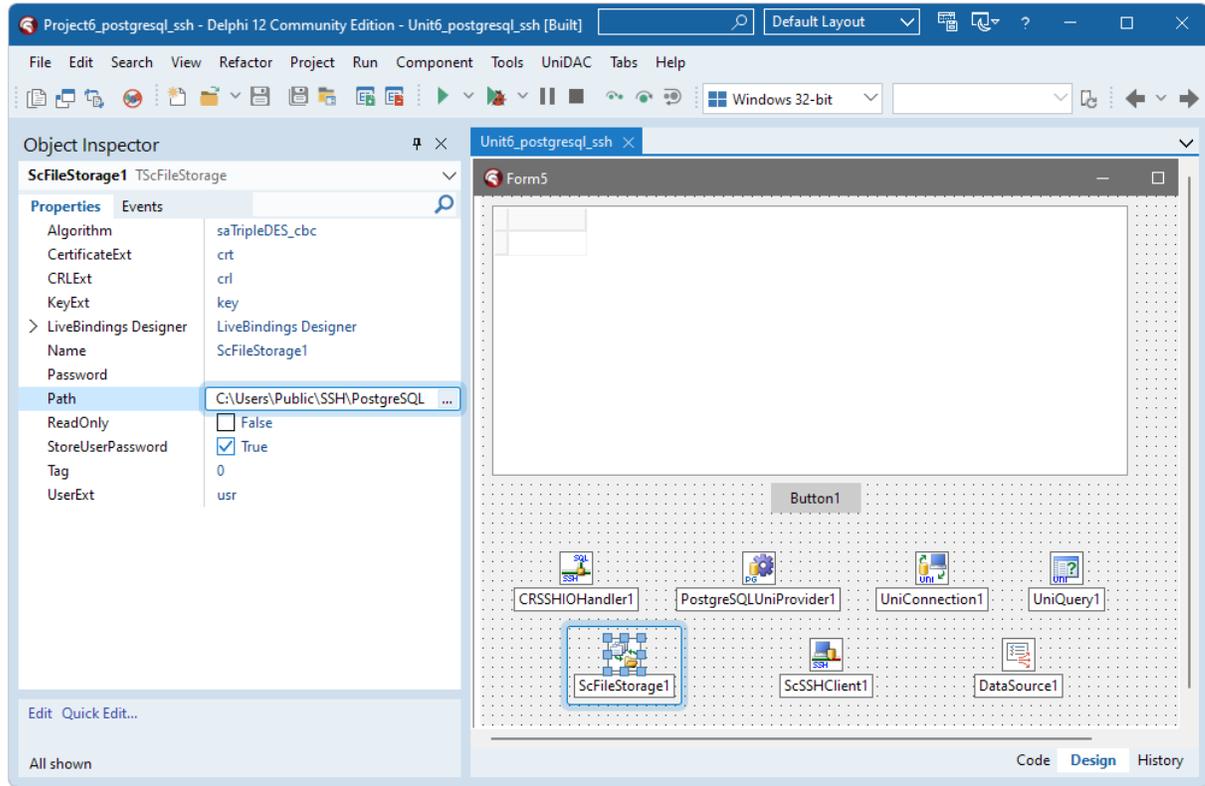
5. Select the **TUniQuery** component and set the **Connection** property to the instance of **TUniConnection (UniConnection1)**.
6. Double-click the **TUniQuery** component, enter a SQL query to be run against the PostgreSQL database, and click **OK**.



7. Select the **TCRSSHIOHandler** component and set the **Client** property to the instance of **TScSSHClient (ScSSHClient1)**.

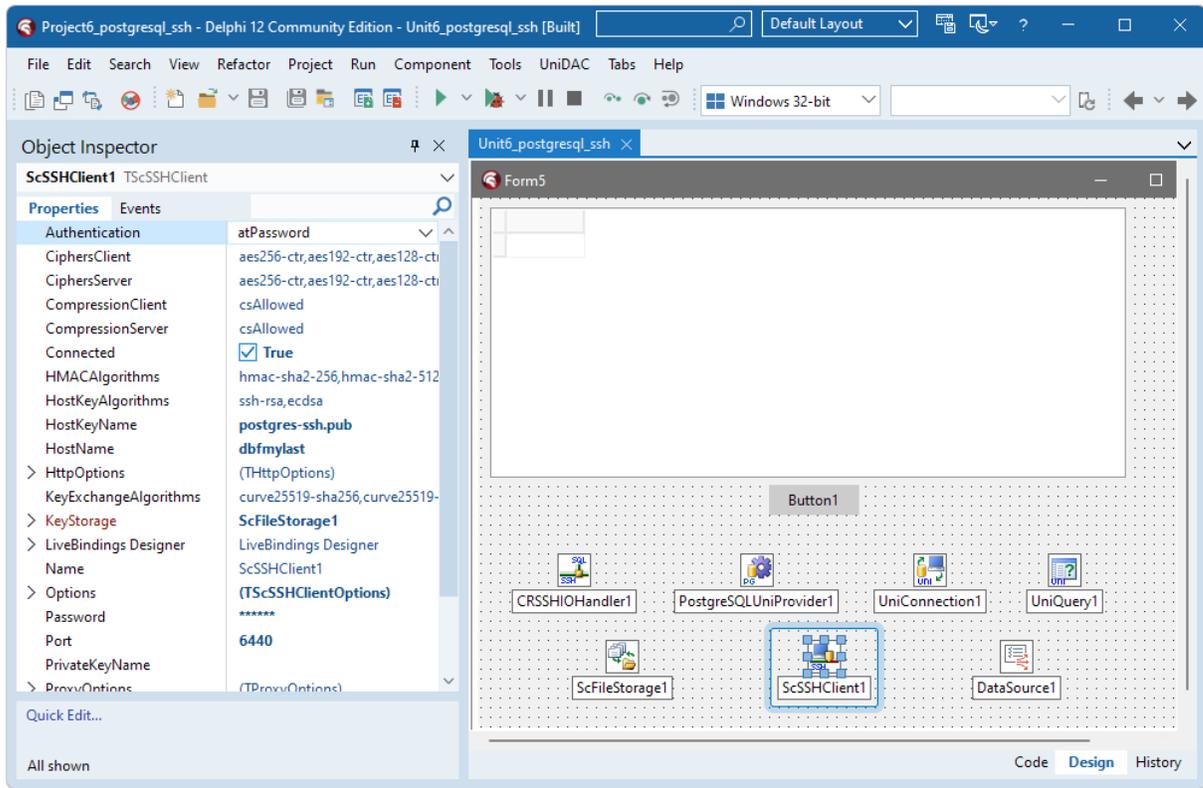


8. Select the **TScFileStorage** component and, in the **Path** property, specify the directory where the keys are stored.



9. Select the **TScSSHClient** component and assign values to the following properties:

- **Authentication** – Select the value depending on the authentication method applicable for your SSH server: **atPassword** or **atPublicKey**.
- **HostKeyName** – Specify the filename of the SSH server public key.
- **Hostname** – Enter the host name or IP address of the SSH server.
- **KeyStorage** – Set the property to the instance of **TScFileStorage** (**ScFileStorage1**).
- **Password** – For password authentication, enter the password for the account on the SSH server.
- **Port** – Specify the SSH port.
- **PrivateKeyName** – For public key authentication, specify the filename of the client private key.
- **User** – Enter the username for the account on the SSH server.

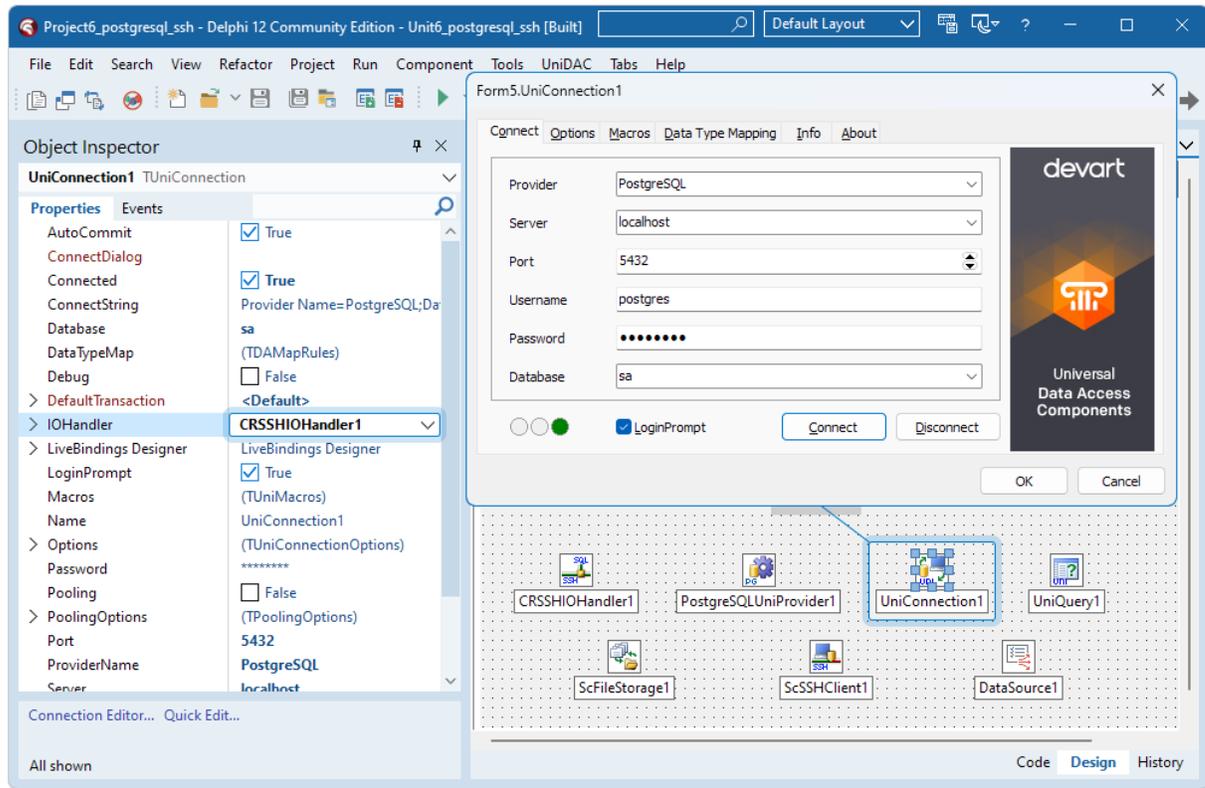


10. Select the **TUniConnection** component and set the **IOHandler** property to the instance of **TCRSSHIOHandler** (**CRSSHIOHandler1**).

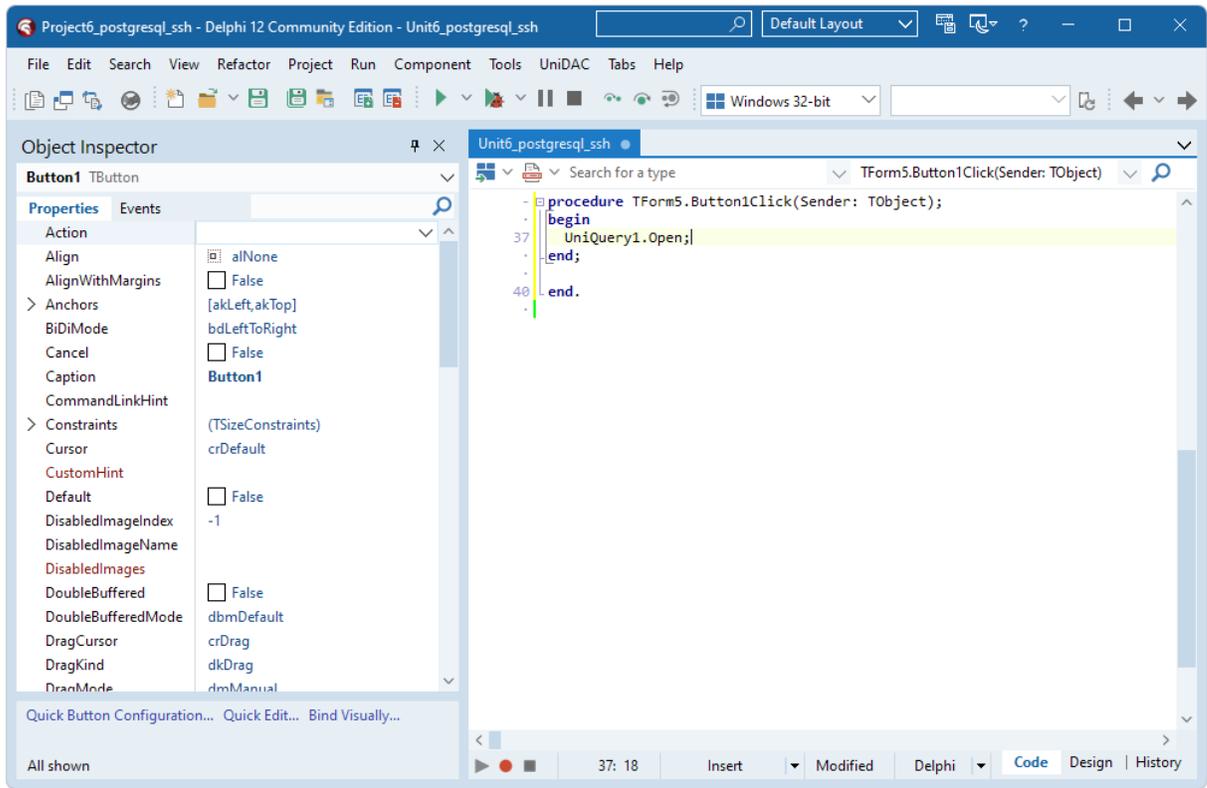
11. Double-click the **TUniConnection** component and fill out the fields on the **Connect** tab:

- **Provider** – Select **PostgreSQL**.
- **Server** – Enter the host name or IP address of the PostgreSQL server. If the SSH server and PostgreSQL server are both running on the same machine, use *localhost*.
- **Port** – Specify the port of the PostgreSQL server.
- **Username** – Enter the username for the account on the PostgreSQL server.
- **Password** – Enter the password for the account on the PostgreSQL server.
- **Database** – Select the database.

12. Click **Connect** to test the connection to the PostgreSQL server, then click **OK** to close the dialog.

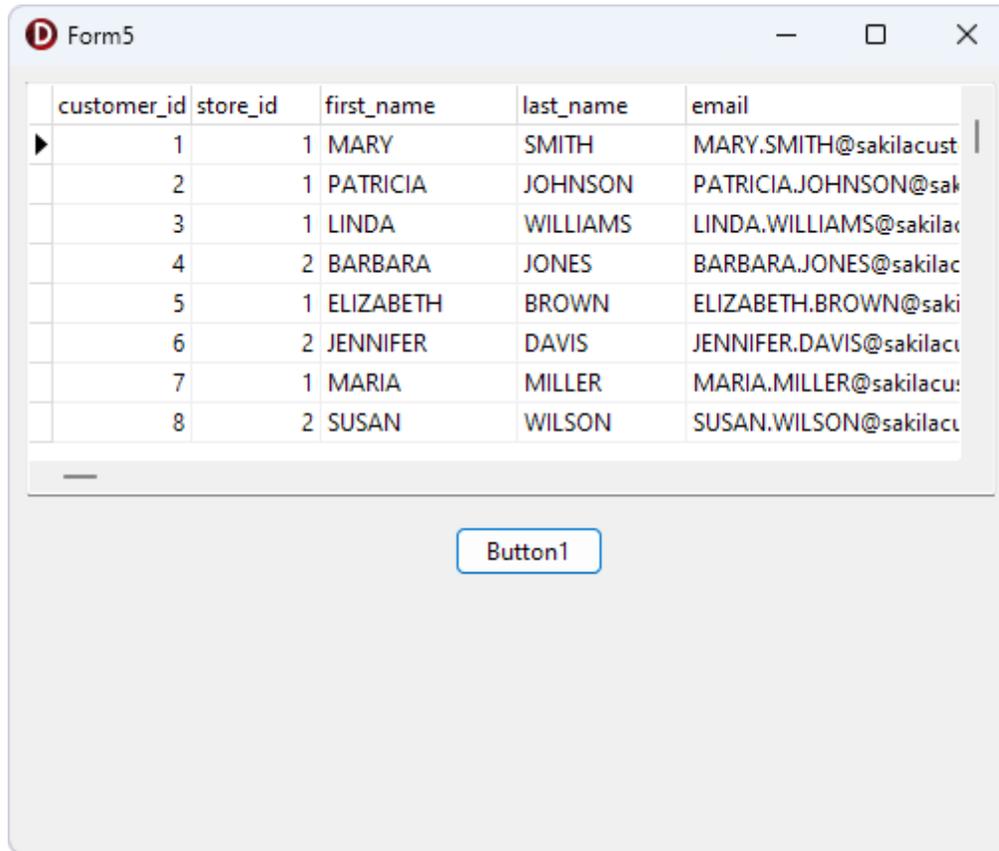


13. Double-click the **TButton** component and add code to call the **Open** method on the **UniQuery1** object to activate the dataset when the button is clicked.



14. Press **F9** to compile and run the application.

15. In the form that appears, click **Button1** to run the query. Data appears in the grid.



Connect Using OpenSSH or Any Other Third-Party SSH Tunneling Tools

Using SecureBridge and its components is not obligatory—you can use any other server that implements the SSH protocol.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.15.4 Network Tunneling

In typical scenarios, a client connects to a server through a direct network path. However, modern security policies or complex network topologies often require the use of a proxy or traversal through a firewall. This article describes different ways to connect to a database server using UniDAC.

- [Direct connection](#)

- [Connection through an HTTP tunnel](#)
- [Connection through a proxy and an HTTP tunnel](#)
- [Additional information](#)

Direct connection

A direct connection to the server means that the server host is accessible from the client without any additional 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 to communicate with the server. Use it whenever possible.

The following example demonstrates how to configure a direct connection:

```
UniConnection := TUniConnection.Create(self);
UniConnection.ProviderName := '<provider_name>';
UniConnection.Server := 'localhost';
UniConnection.Port := 3306;
UniConnection.Username := '<user>';
UniConnection.Password := '<password>';
UniConnection.Connect;
```

Connection through an HTTP tunnel

Sometimes, client computers are shielded by a firewall that does not allow you to connect directly to the server on the specified port. If the firewall allows HTTP connections, you can use UniDAC and HTTP tunneling software to connect to the database server.

UniDAC supports HTTP tunneling based on a PHP script.

Consider a scenario where direct access to the database server port is blocked on a remote host, and only HTTP traffic on port 80 is permitted. In such cases, web script tunneling allows remote access to the database as if it were a direct connection.

You need to deploy the `tunnel.php` script included in the provider package on the web server. It allows access to the database server using HTTP tunneling. The script must be accessible through the HTTP protocol. You can check if it is accessible using a web browser. You can find the script in the HTTP subfolder of the installed provider folder, such as `%Program Files%\Devart\UniDac for Delphi X\HTTP\tunnel.php`. The only requirement is that the server must support PHP 5.

To connect to the database via HTTP tunneling, first configure `TUniConnection` as for a direct connection from the web server. Then, set the `Protocol` option to `mpHttp` and provide the

following HTTP tunneling parameters:

Specific Option	Required	Description
HttpUrl	Yes	The URL of the tunneling PHP script. For example, if the script is in the server root, the URL can be <code>http://localhost/tunnel.php</code> .
HttpUsername, HttpPassword	No	A registered user credentials. Set these properties if access to the website folder with the script is available only for registered users authenticated with a username and password.

Connection through a proxy and an HTTP tunnel

Consider the previous case with one more complication.

The HTTP tunneling server is not directly accessible from the client computer. For example, the client address is 10.0.0.2, the server address is 192.168.0.10, and the database server is listening on port 3307. Since the client and the server are on different networks, the client can only access the server through a proxy located at 10.0.0.1 and listening on port 808. In this case, in addition to the HTTP-specific options, you need to set the proxy-specific options.

```
UniConnection := TUniConnection.Create(self);
UniConnection.ProviderName := '<provider_name>';
UniConnection.Server := '192.168.0.10';
UniConnection.Port := 3307;
UniConnection.Username := '<user>';
UniConnection.Password := '<password>';
UniConnection.SpecificOptions.Values['Protocol'] := 'mpHttp';
UniConnection.SpecificOptions.Values['HttpUrl'] := 'http://server/tunnel.php';
UniConnection.SpecificOptions.Values['ProxyHostname'] := '10.0.0.1';
UniConnection.SpecificOptions.Values['ProxyPort'] := '808';
UniConnection.SpecificOptions.Values['ProxyUsername'] := '<proxy_user>';
UniConnection.SpecificOptions.Values['ProxyPassword'] := '<proxy_password>';
UniConnection.Connect;
```

Note that when proxy-specific options are configured, UniDAC will automatically route the connection through the specified proxy server.

Additional information

Remember that traffic tunneling or encryption increases CPU usage and network load. Direct connections are recommended whenever possible.

Reserved.

5.1.16 UniDAC and SQL Server

5.1.16.1 SQL Server Provider

This article provides a brief overview of the SQL Server data access provider for UniDAC used to establish a connection to SQL Server databases from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [SQL Server-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)
- [SQL Server specific notes](#)
- [Connecting in Direct mode](#)
- [Data Type Mapping](#)



Overview

SQL Server data access provider is based on the SQL Server Data Access Components ([SDAC](#)) library, which is one of the best known Delphi data access solutions for SQL Server. The main features of SQL Server data access provider are:

- Access to the SQL Server through the lowest documented protocol level (OLE DB)
- High performance
- Easy deployment
- Comprehensive support for the latest versions of SQL Server

The full list of SQL Server provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the SQL Server provider. For Express Edition of UniDAC, the SQL Server provider can be installed with SDAC.

Compatibility

To learn about SQL Server compatibility, refer to the [Compatibility](#) section.

Requirements

SQL Server provider requires OLE DB or SQL Native Client installed on workstation.

In the current versions of Microsoft Windows, since Windows 2000, OLE DB is already included as a standard package. But it's highly recommended to download the latest version (higher than 2.5) of Microsoft Data Access Components (MDAC) or SQL Native Provider.

Some features of SQL Server 2005 are available only with SQL Native Provider.

If you are working with SQL Server Compact Edition, you should have it installed. You can download SQL Server Compact Edition from the site of Microsoft.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

SQL Server-specific options

Though UniDAC is components that provide unified interface to work with different database servers, it also lets you tune the behaviour for each server individually. For thin setup of a certain database server, UniDAC provides server-specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a string list. Therefore you can use the following syntax to assign an option value:

```
UniConnection.SpecificOptions.Values['ApplicationName'] := 'My application';
```

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
AccessToken	Specifies the access token used for authentication in Azure Active Directory. Supported only by the prMSOLEDB provider (Microsoft OLE DB Driver for SQL Server).
ApplicationIntent	Specifies the application workload type when connecting to a server.
ApplicationName	The name of a client application. The default value is the name of the executable file of your application.
Authentication	<p>Use the Authentication property to specify authentication service used by the database server to identify a user. The Authentication property accepts one of the following values:</p> <ul style="list-style-type: none"> auWindows Uses Windows NT/2000/XP integrated security, or "SSPI" (Security Support Provider Interface). Username, Password and LoginPrompt properties are ignored. auServer (default) An alternative way of identifying users by database server. To establish a connection valid Username and Password either hardcoded into application or provided in server login prompt fields are required. auADPassword User ID and password authentication with an Azure Active Directory identity. auADIntegrated Integrated authentication with an Azure

	<p>Active Directory identity.</p> <p><code>auADInteractive</code></p> <p>Interactive authentication with an Azure Active Directory identity. This method supports Azure Active Directory Multi-Factor Authentication (MFA).</p> <p><code>auActiveDirectoryMSI</code></p> <p>Managed Identity (MSI) authentication. For a user-assigned identity, the user ID should be set to the object ID of the user identity.</p> <p><code>auSqlPassword</code></p> <p>Authentication using user ID and password.</p> <p>Note: The <code>auADPassword</code>, <code>auADIntegrated</code>, <code>auADInteractive</code>, <code>auActiveDirectoryMS</code>, and <code>auSqlPassword</code> authentication methods are supported only for the <code>prMSOLEDB</code> provider (Microsoft OLE DB Driver for SQL Server).</p>
AutoTranslate	When set to True, character strings sent between the client and server are translated by converting through Unicode to minimize problems in matching extended characters between the code pages on the client and the server.
Charset	Specifies the character set applied when writing string parameter values in the national encoding format. If Charset is not configured, the system locale name is used by default.
CompactAutoShrinkThreshold	Specifies the amount of free space in the database file before automatic shrink will start. Measured in percents. The default value is 60.
CompactDefaultLockEscalation	Specifies how many locks should be performed before trying escalation from row to page or from page to table. The default value is 100.
CompactFlushInterval	Specifies the interval at which committed transactions are flushed to disk. Measured in seconds. The default value is 10.
CompactInitMode	Use this property to specify the file mode that will be used to open the database file. The <code>InitMode</code> property accepts one of the following values: <code>imExclusive</code> Database file is opened for exclusive use. This mode prevents others from opening this database file. <code>imReadOnly</code> Database file is opened for reading. All operations that write to database are unallowable. <code>imReadWrite</code> (<i>default</i>) Both read and write operations are allowed.

	<p><code>imShareRead</code> Opens a database file preventing others from opening the same file in the read mode.</p>
<code>CompactLocaleIdentifier</code>	Specifies the locale ID. The default value is the system default locale on Windows systems and 0 on other systems.
<code>CompactLockEscalation</code>	Specifies how many locks should be performed before trying escalation from row to page or from page to table. Measured in milliseconds. The default value is 100.
<code>CompactLockTimeout</code>	Specifies how much time a transaction will wait for a lock. Measured in milliseconds. The default value is 2000.
<code>CompactMaxBufferSize</code>	Specifies how much memory SQL Server Compact Edition can use before flushing changes to disc. Measured in kilobytes. The default value is 640.
<code>CompactMaxDatabaseSize</code>	Specifies maximum size of the main database file. Measured in megabytes. The default value is 128.
<code>CompactTempFileDirectory</code>	Specifies the temp file directory. If this option is not assigned, the current database is used as a temporary database.
<code>CompactTempFileMaxSize</code>	Specifies maximum size of the temporary database file. Measured in megabytes. The default value is 128.
<code>CompactTransactionCommitMode</code>	<p>Specifies in what way the buffer pool will be flushed on transaction commit. The following two values are allowed:</p> <p><code>cmAsynchCommit</code> Asynchronous commit to disk.</p> <p><code>cmSynchCommit</code> (<i>default</i>) Synchronous commit to disk.</p>
<code>CompactVersion</code>	<p>Specifies which version of SQL Server Compact Edition will be used.</p> <p><code>cvAuto</code> (<i>default</i>) Version of SQL Server Compact Edition will be chosen automatically depending on database version. If database is not provided, the higher available server version will be chosen.</p> <p><code>cv30</code> Uses SQL Server Compact Edition Version 3.0 or 3.1.</p> <p><code>cv35</code> Uses SQL Server Compact Edition Version 3.5.</p>
<code>ConnectionTimeout</code>	Use <code>ConnectionTimeout</code> to specify the amount of time, in seconds, that can expire before an attempt to consider a connection unsuccessful. The default value is 15 seconds.
<code>Encrypt</code>	Specifies if data should be encrypted before sending it over the network. The default value is <code>False</code> .
<code>FailoverPartner</code>	Specifies the SQL Server name to which SQL Native Client will reconnect when a failover of the principal SQL Server occurs. This option is supported only for SQL Server 2005 using SQL Native Client as an OLE DB provider.

ForceCreateDatabase	Used to force TLiteConnection to create a new database before opening a connection, if the database does not exist.
HttpPassword	Use the HttpPassword option to specify the password for HTTP authorization.
HttpTrustServerCertificate	This option specifies whether or not the driver should trust the server certificate when connecting to the server. The default value is False—the driver won't trust the server certificate and will verify validity of the server certificate instead. If set to True, the driver will trust the server certificate.
HttpUrl	Use the HttpUrl option to specify the URL of the PHP tunneling script.
HttpUsername	Use the HttpUsername option to specify the username for HTTP authorization.
IPVersion	Use the IPVersion property to specify Internet Protocol Version. Supported values: ivIPBoth Specifies that either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) will be used. ivIPv4 (<i>default</i>) Specifies that Internet Protocol Version 4 (IPv4) will be used. ivIPv6 Specifies that Internet Protocol Version 6 (IPv6) will be used. Note: When the TIPVersion property is set to ivIPBoth, a connection attempt will be made via IPv6 if it is enabled on the operating system. If the connection attempt fails, a new connection attempt will be made via IPv4.
InitialFileName	Specifies the name of the main database file. This database will be default database for the connection. SQL Server attaches the database to the server if it has not been attached to the server yet. So, this property can be used to connect to the database that has not been attached to the server yet.
Language	A SQL Server language name. Identifies the language used for system message selection and formatting. The language must be installed on the computer running an instance of SQL Server otherwise the connection will fail.
LockTimeout	Specifies the number of milliseconds that a transaction will wait to obtain a lock to avoid global deadlocks. The default value is 2000.
MSOLEDBVersion	Is used to indicate the version of Microsoft OLE DB driver. The option has the TMSOLEDBVersion type with the following supported values: ole18 Microsoft OLE DB Driver for SQL Server 18 is used. ole19 Microsoft OLE DB Driver for SQL Server 19 is used.

	<p>oleAuto UniDAC first searches for Microsoft OLE DB Driver for SQL Server 19 and then for Microsoft OLE DB Driver for SQL Server 18. The first found driver is used.</p>
MultipleActiveResult Sets	<p>Enables support for SQL Server 2005 Multiple Active Result Sets (MARS) technology. It allows applications to have more than one pending request per connection, and in particular, to have more than one active default result set per connection. Current session is not blocked when using FetchAll = False, and it is not necessary for OLE DB to create additional sessions for any query executing. MARS is only supported by SQL Server 2005 with using SQL Native Client as OLE DB provider.</p>
MultipleConnections	<p>Enables or disables the creation of additional connections to support concurrent sessions, commands and rowset objects.</p>
MultiSubnetFailover	<p>Use the MultiSubnetFailover option to configure the prNativeClient or prMSOLEDB provider to faster detect and connect to the currently active server by making simultaneous connection attempts to all IP addresses associated with the group listener of a SQL Server AlwaysOn Availability Group or a SQL Server Failover Cluster Instance. The default value is False.</p>
NativeClientVersion	<p>Specifies which version of SQL Native Client will be used. The default value is ncAuto. NativeClientVersion is applied when the Provider property is set to prNativeClient or prAuto.</p>
NetworkLibrary	<p>The name of the Net-Library (DLL) used to communicate with an instance of SQL Server. The name should not include the path or the .dll file name extension. The default name is provided by the SQL Server Client Network Utility.</p>
PacketSize	<p>Network packet size in bytes. The packet size property value must be between 512 and 32,767. The default network packet size is 4,096.</p>
PersistSecurityInfo	<p>The data source object is allowed to persist sensitive authentication information such as password along with other authentication information.</p>
Provider	<p>This property allows you to specify a provider from the list of supported providers or use the Direct mode. Some features added to SQL Server 2005 require the SQL Native Client (prNativeClient) provider to be used. If chosen provider is not installed, an exception is raised.</p> <p>Supported values: prAuto (<i>default</i>) prAuto is the default value of the Provider property. With default value, UniDAC will use the most recent version of one of the supported providers in the following order:</p> <ol style="list-style-type: none"> 1. prNativeClient

	<p>2. prMSOLEDB</p> <p>3. prSQL</p> <p>First UniDAC checks whether SQL Server Native Client is installed in the system. If SQL Server Native Client is not found, UniDAC looks for Microsoft OLE DB Driver for SQL Server. If neither SQLNCLI nor MSOLEDBSQL is installed in the system, the driver will use Microsoft OLE DB Provider for SQL Server.</p> <p>prSQL Uses the provider preinstalled with Windows that has limited functionality.</p> <p>prMSOLEDB Uses Microsoft OLE DB Driver for SQL Server (MSOLEDBSQL). You need to have the driver installed on your system to use this value for Provider.</p> <p>prNativeClient Uses the SQL Native Client. It should be installed on the computer to use this Provider value. This provider offers the maximum functionality set.</p> <p>prCompact SQL Server Compact Edition provider.</p> <p>prDirect Connect to SQL Server directly via TCP/IP.</p>
QuotedIdentifier	<p>Causes Microsoft® SQL Server™ to follow the SQL-92 rules regarding quotation mark delimiting identifiers and literal strings. Identifiers delimited by double quotation marks can be either Transact-SQL reserved keywords or can contain characters not usually allowed by the Transact-SQL syntax rules for identifiers. QuotedIdentifier must be True when creating or manipulating indexes on computed columns or indexed views. If QuotedIdentifier is False, CREATE, UPDATE, INSERT, and DELETE statements on tables with indexes on computed columns or indexed views will fail.</p> <p>True (<i>default</i>)</p> <p>Identifiers can be delimited by double quotation marks, and literals must be delimited by single quotation marks. All strings delimited by double quotation marks are interpreted as object identifiers. Therefore, quoted identifiers do not have to follow the Transact-SQL rules for identifiers. They can be reserved keywords and can include characters not usually allowed in Transact-SQL identifiers. Double quotation marks cannot be used to delimit literal string expressions; single quotation marks must be used to enclose literal strings. If a single quotation mark (') is a part of the literal string, it can be represented by two single quotation marks ("). QuotedIdentifier</p>

	<p>must be True when reserved keywords are used for object names in the database.</p> <p>False (BDE compatibility)</p> <p>Identifiers cannot be quoted and must follow all Transact-SQL rules for identifiers. Literals can be delimited by either single or double quotation marks. If a literal string is delimited by double quotation marks, the string can contain embedded single quotation marks, such as apostrophes.</p>
UseHttp	The UseHttp option enables the use of HTTP tunneling to connect to the server. The default value is False.
UseWideMemos	Use the option to manage the field type that will be created for the NTEXT data type. If True (default), TWideMemo fields will be created for the NTEXT data type. If False, TMemo fields will be created.
UuidWithBraces	Determines whether curly braces should be added to GUID values of the uniqueidentifier field type. This option is Boolean. The default value is True.
TrustServerCertificate	Lets enabling traffic encryption without validation. The default value is False. This option is only supported by SQL Server 2005 with using SQL Native Client as OLE DB provider.
WorkstationID	A string identifying the workstation. The default value is the name of your machine.

TUniSQL

Option name	Description
CommandTimeout	Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. Is measured in seconds. If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect. The default value is 0 (infinite).
DescribeParams	Specifies whether to query the Name, ParamType, DataType, Size, and TableName properties from the server when preparing a query. The default value is False.
NonBlocking	Used to execute an SQL statement in a separate thread. Set the NonBlocking option to True to fetch rows in a separate thread.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CheckRowVersion	Determines whether the dataset checks for rows modifications made by another user on automatic generation of SQL statement

	<p>for update or delete data. If CheckRowVersion property is False and DataSet has keyfields, the WHERE clause of SQL statement is generated basing on these keyfields. If there is no primary key and no Identity field, then all non-BLOB fields will take part in generating SQL statements. If CheckRowVersion is True and DataSet has TIMESTAMP field, only this field is included into WHERE clause of generated SQL statement. Otherwise, all non BLOB fields are included. All mentioned fields refer to the current UpdatingTable. The default value is False. The CheckRowVersion option requires enabled DMLRefresh.</p>
CommandTimeout	<p>Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. Is measured in seconds. If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect. The default value is 0 (infinite).</p>
CursorType	<p>Allows choosing cursor types supported by SQL Server. The available values are:</p> <p>ctBaseTable</p> <p>Base table cursor. This cursor is used for working with Compact Edition. This cursor is the fastest of the SQL server cursors and the only cursor that interacts directly with the storage engine. This allows to increase the speed of data access several times. Data modifications, deletions, and insertions by other users are visible. If UniDirectional=False, the cursor is used only when fetching data, and Data updates are reflected on database by SQL statements execution. In order to use the cursor also for data modification it is necessary to set the UniDirectional property to True. But in this case the cursor does not support bookmarks and cannot be represented in multiline controls such as DBGrid.</p> <p>ctDefaultResultSet (<i>default</i>)</p> <p>By the old SQL Server terminology is the Firehose cursor. It serves for the fastest data fetch from server to the client side. Allows to run batches. Data updates are reflected in the database only by SQL statements execution. The</p>

	<p>ctDynamic</p> <p>ctKeyset</p> <p>ctStatic</p>	<p>default value.</p> <p>Dynamic cursor. Used when data is not cached at the server and fetch is performed row by row as required. Doesn't support bookmarks and cannot be represented in multiline controls such as DBGrid. Data modifications, deletions, and insertions by other users are visible. Data updates are reflected on database both by SQL statements execution and server cursors means.</p> <p>Allows to cache only keyfields at the server. Fetching is performed row by row when a data-aware component or a program requests it. Records added by other users are not visible, and records deleted by other users are inaccessible. Data updates are reflected in the database both by SQL statements execution and server cursors means.</p> <p>Static copying of records. Query execution results are cached at the server. Fetch is performed row by row when a data-aware component or a program requests it. When a cursor is opened, all newly added updates are invisible. Used mostly for reporting.</p>
CursorUpdate		Specifies what way data updates reflect on database when modifying dataset by using server cursors ctKeySet and ctDynamic. If the CursorUpdate property is True, all dataset modifications pass to database by server cursors. If the CursorUpdate property is False, all dataset updates pass to server by SQL statements generated automatically or specified in SQLUpdate, SQLInsert or SQLDelete. The default value is True.
DescribeParams		Specifies whether to query the Name, ParamType, DataType, Size, and TableName properties from the server when preparing a query. The default value is False.
DisableMultipleResults		Use the option to disable support for the Multiple Active Result Sets (MARS) technology, which allows applications to have multiple pending requests per connection and multiple default result sets per connection. The default value is False.
FetchAll		If True, all records of the query are requested from the database

	server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is True.
HideSystemUniqueFields	Used the option to hide system fields for the prSQL, prNativeClient and prMSOLEDB providers. The default value is True.
LastIdentityValueFunction	<p>Determines which system function to use to obtain an identifier when adding a record. The available values are:</p> <p>vfIdentCurrent</p> <p>The IDENT_CURRENT system function is used. It returns the last identity value generated for a specified table or view. The last identity value generated can be for any session and any scope.</p> <p>vfIdentity</p> <p>The @@IDENTITY system function is used. It returns the last-inserted identity value.</p> <p>vfScopeIdentity</p> <p>The SCOPE_IDENTITY system function is used. It returns the last identity value inserted into an identity column in the same scope. A scope is a module: a stored procedure, trigger, function, or batch.</p>
NonBlocking	Set the NonBlocking option to True to fetch rows in a separate thread. The BeforeFetch event is called in the additional thread context that performs data fetching. This event is called every time on the Fetch method call. The AfterFetch event is called in the main thread context only once after fetching is completely completed. In the NonBlocking mode, as well as if FetchAll = False, an extra connection is created. When setting the NonBlocking option to True, you should keep in mind that execution of such queries blocks the current session. In order to avoid blocking, OLE DB creates an additional session as in FetchAll = False. It causes the same problems when FetchAll = False. This problem can be solved by using MARS (the specific option MultipleActiveResultSets = True). The current session is not blocked and OLE DB is not required to create an additional session to run a query. MARS is supported since SQL Server 2005 if SQL Native Client is used as OLE DB provider.
QueryIdentity	Specifies whether to request Identity field value, if such exists, on execution Insert or Append method. If to refuse of getting Identity you can have an impact on performance of Insert or Append by about 20%. Affects only for ctDefaultResultSet cursor. If you are inserting value into SQL_VARIANT field, and QueryIdentity is True then an error is raised. The default value is True.

UniqueRecords	Use UniqueRecords to specify whether to query additional key fields from the server. If UniqueRecords is False, keyfields are not queried from the server when they are not included in the query explicitly. For example, the result of the query execution "SELECT ShipName FROM Orders" holds the only field ShipName. When used with the ReadOnly property set to True, UniqueRecords option gives insignificant advantage of performance. But in this case SQLRefresh will be generated in simplified way. If UniqueRecord is True, keyfields needed for complete automatic generation of SQLInsert, SQLUpdate, SQLDelete or SQLRefresh statements are queried from the server implicitly. For example, the result of query execution "SELECT ShipName FROM Orders" holds at least two fields ShipName and OrderID. The default value is False. Has effect only for ctDefaultResultSet cursor.
---------------	---

TUniScript

The TUniScript component has no SQL Server-specific options.

TUniLoader

Option name	Description
FireTrigger	Use the option to fire table triggers with TMSLoader on SQL Server during insertion operations. The default value is False.
KeepIdentity	Use the KeepIdentity property to specify in what way IDENTITY column values must be handled. If KeepIdentity is set to False, IDENTITY columns will be initialized by the server. Any value assigned to such column in your application is ignored. If KeepIdentity is set to True, the IDENTITY property will not be available for all IDENTITY fields accepting NULL. So in this case unique values should be generated and assigned by the client application. The default value of the KeepIdentity property is False.
KeepNulls	If this option is set to False, each NULL value inserted into a field with a DEFAULT constraint will be replaced with the default value. If KeepNulls is set to True, NULL values inserted into a field with a DEFAULT constraint will not be replaced with the default values. The default value of the KeepNulls property is False.
RowsPerBatch	Use the RowsPerBatch property to specify the number of rows to load in a single batch. Server optimizes loading according to this value. The default value of this option is Unknown.
KilobytesPerBatch	Use the KilobytesPerBatch option to specify the size of data in kilobytes to load in a single batch. The default value of this option

	is Unknown.
LockTable	Use the LockTable property to specify if the table-level lock is performed while loading is in progress. Setting this option to True should improve the performance greatly. If this option is set to False, the locking behaviour is determined by the table option. The default value of the LockTable option is False.
CheckConstraints	Use the CheckConstraints property to specify if the table constraints are checked during loading. If this option is set to False, the table constraints are not checked. The default value of the CheckConstraints option is False.
QuoteNames	Use the QuoteNames option to quote all database object names in automatically generated SQL statements, such as UPDATE statements. The default value is False.

TUniDump

Option name	Description
IdentityInsert	Use the IdentityInsert property to add SET IDENTITY_INSERT TableName ON at the beginning of the script and SET IDENTITY_INSERT TableName OFF at the end of the script. The first line allows explicit values to be inserted into the identity column of a table and INSERT statements are generated with IDENTITY field values. Otherwise the IDENTITY field will not be included to the INSERT statements. SET IDENTITY_INSERT will not be added while the option is ON if the table does not have a field identified as IDENTITY or there are no records in the table.

SQL Server specific notes

Connecting in Direct mode

By default, the OLE DB interface is used directly through a set of COM-based interfaces to connect to server. Such approach allows using client applications on Windows workstations only.

To overcome these problems, the prDirect value for the Provider property was added for ability to connect to SQL Server directly over the network using the TCP/IP protocol. This is referred to as connecting in the Direct mode. Connection in the Direct mode does not require OLEDB provider or SQL Native Client provider to be installed on target machines. The only requirement for running an UniDAC-based application that uses the Direct mode is that the operating system must support the TCP/IP protocol.

Setting up Direct mode connections

Here is an example that illustrates connecting to SQL Server in the Direct mode. The server's IP address is 205.227.44.44, its port number is 1433 (this is the most commonly used port for SQL Server).

```
var
  UniConnection: TUniConnection;
. . .
UniConnection.ProviderName := 'SQL Server';
UniConnection.SpecificOptions.Values['Provider'] := 'prDirect';
UniConnection.SpecificOptions.Values['Authentication'] := 'auServer';
UniConnection.Username := 'sa';
UniConnection.Password := '';
UniConnection.Server := '205.227.44.44';
UniConnection.Port := 1433;
UniConnection.Connect;
```

All we have to do is to set the `TUniConnection.Options.Provider` property to `prDirect` to enable Direct mode connections in your application. You do not have to rewrite other parts of your code.

Data Type Mapping

The following table lists the constants for mapping SQL Server data types to Delphi data types. See [Data Type Mapping](#) for more information.

Constant	Description
<code>msBigint</code>	Maps <code>bigint</code> to Delphi data types.
<code>msBinary</code>	Maps <code>binary</code> to Delphi data types.
<code>msBit</code>	Maps <code>bit</code> to Delphi data types.
<code>msChar</code>	Maps <code>char</code> to Delphi data types.
<code>msDate</code>	Maps <code>date</code> to Delphi data types.
<code>msDatetime</code>	Maps <code>datetime</code> to Delphi data types.
<code>msDatetime2</code>	Maps <code>datetime2</code> to Delphi data types.
<code>msDatetimeoffset</code>	Maps <code>datetimeoffset</code> to Delphi data types.
<code>msDecimal</code>	Maps <code>decimal</code> to Delphi data types.
<code>msFloat</code>	Maps <code>float</code> to Delphi data types.
<code>msImage</code>	Maps <code>image</code> to Delphi data types.
<code>msInt</code>	Maps <code>int</code> to Delphi data types.
<code>msMoney</code>	Maps <code>money</code> to Delphi data types.
<code>msNChar</code>	Maps <code>nchar</code> to Delphi data types.
<code>msNText</code>	Maps <code>ntext</code> to Delphi data types.

msNumeric	Maps numeric to Delphi data types.
msNVarchar	Maps nvarchar to Delphi data types.
msReal	Maps real to Delphi data types.
msSmalldatetime	Maps smalldatetime to Delphi data types.
msSmallint	Maps smallint to Delphi data types.
msSmallmoney	Maps smallmoney to Delphi data types.
msSqlVariant	Maps sql_variant to Delphi data types.
msText	Maps text to Delphi data types.
msTime	Maps time to Delphi data types.
msTimestamp	Maps timestamp to Delphi data types.
msTinyint	Maps tinyint to Delphi data types.
msUniqueIdentifier	Maps uniqueidentifier to Delphi data types.
msVarbinary	Maps varbinary to Delphi data types.
msVarchar	Maps varchar to Delphi data types.
msXml	Maps xml to Delphi data types.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.16.2 Connecting via SSL

SSL (Secure Sockets Layer) is a standard protocol used to establish secure connections to remote machines over untrusted networks. It runs on top of TCP/IP and ensures secure client-server communication by enabling mutual authentication between SSL-enabled clients and servers.

During server authentication, the client uses public-key cryptography to verify the server's identity. This involves checking that the server's certificate is issued by a trusted Certificate Authority and that it correctly corresponds to the server's public key.

Client authentication, on the other hand, allows the server to verify the client's identity. While mutual authentication using self-signed certificates is possible, it is generally discouraged except in controlled environments such as intranets or development servers.

After the SSL connection is established, the client and the server exchange messages that are symmetrically encrypted with a shared secret key. Compared to SSH, SSL is often preferred for connecting to SQL Server due to its simpler configuration and better performance.

See the SQL Server documentation for more information on [how to enable SSL encryption for](#)

[SQL Server.](#)

To establish an SSL connection to SQL Server using UniDAC, enable the `Encrypt` option.

```
UniConnection1.SpecificOptions.Values['Encrypt'] := 'True';  
UniConnection1.Connect;
```

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.16.3 Connecting via SSH

SSH is a protocol that allows users to securely log in 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.

You can use SSH to secure the network connection between a Delphi application and SQL Server. An SSH connection enables you to run shell commands as if you were physically operating the remote machine.

This page demonstrates how to use UniDAC to connect to SQL Server securely via an SSH tunnel.

To follow the steps on this page, you need an SSH client and an SSH server. You can create them using [SecureBridge](#), a solution provided by Devart. It enables you to embed an SSH client into your Delphi application and, if needed, configure an SSH server. For more information, see the SecureBridge [tutorial on configuring an SSH server](#). You can also build the SSHServer [demo project](#), which is distributed with SecureBridge.

Your SQL Server instance must be configured to accept connections from the SSH tunnel.

Connecting to SQL Server via SSH is only supported in Direct mode. Direct mode—`prDirect`—is our proprietary implementation of the Tabular Data Stream (TDS) protocol. In this mode, UniDAC communicates directly with SQL Server without relying on any additional libraries.

```
UniConnection1.SpecificOptions.Values['Provider'] := 'prDirect';
```

- [Connect Using SecureBridge](#)
- [Connect Using Any Third-Party SSH Tunnel](#)

Connect Using SecureBridge

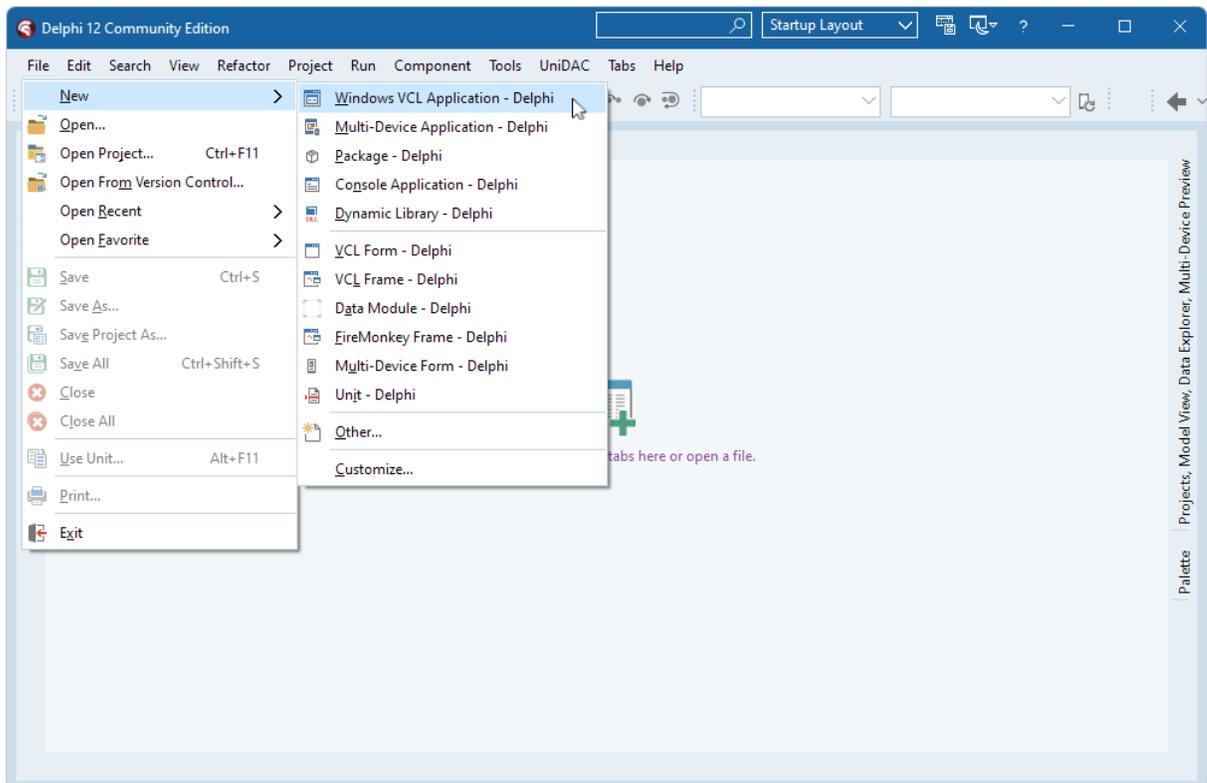
You can establish a secure SSH tunnel using SecureBridge without the need for third-party applications. SecureBridge provides all the necessary components to create an SSH client within your Delphi application.

Let's create a sample Delphi app that connects to SQL Server using SSH.

Prerequisites:

- Install SecureBridge.
- Install the TCRSSHIOHandler component in RAD Studio to bind UniDAC with SecureBridge. For instructions, see `Readme.html`—by default, it is located in `C:\Users\\Documents\Devart\UniDAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge`.

1. In RAD Studio, select **File > New > Windows VCL Application - Delphi**.

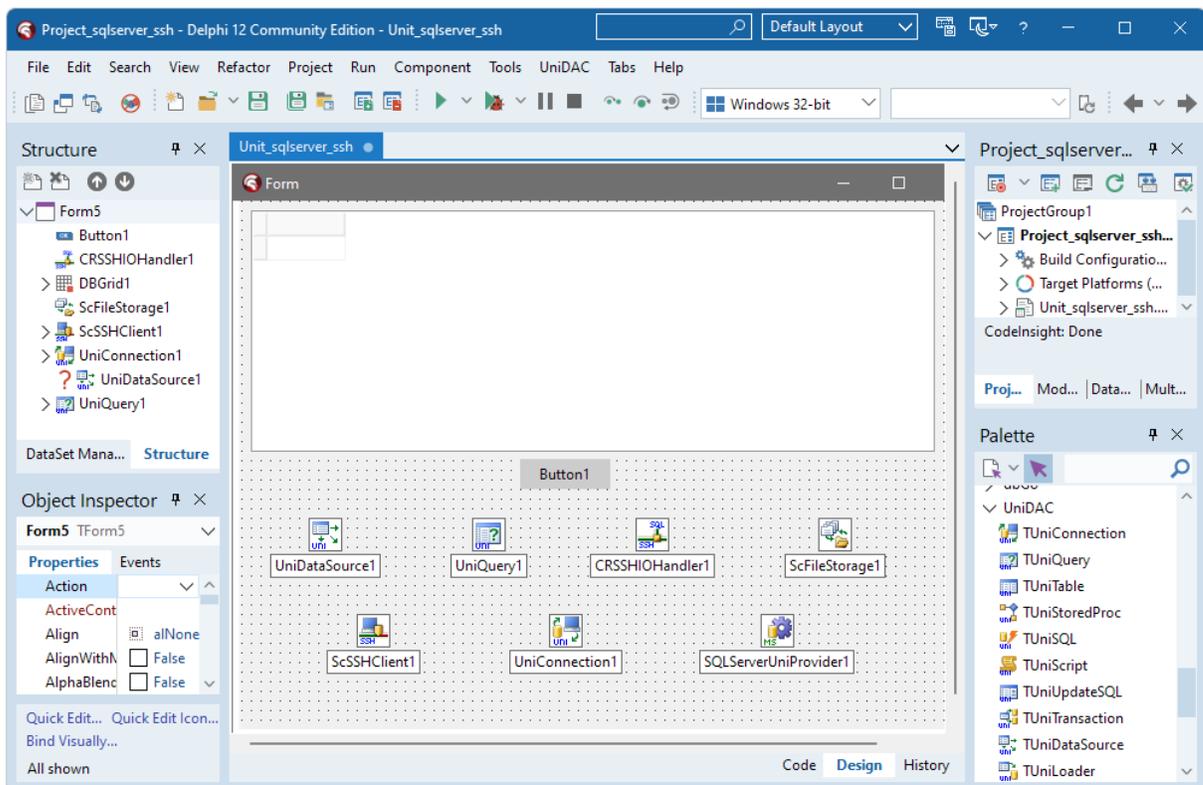


2. Place the following components from **Palette** on the form:

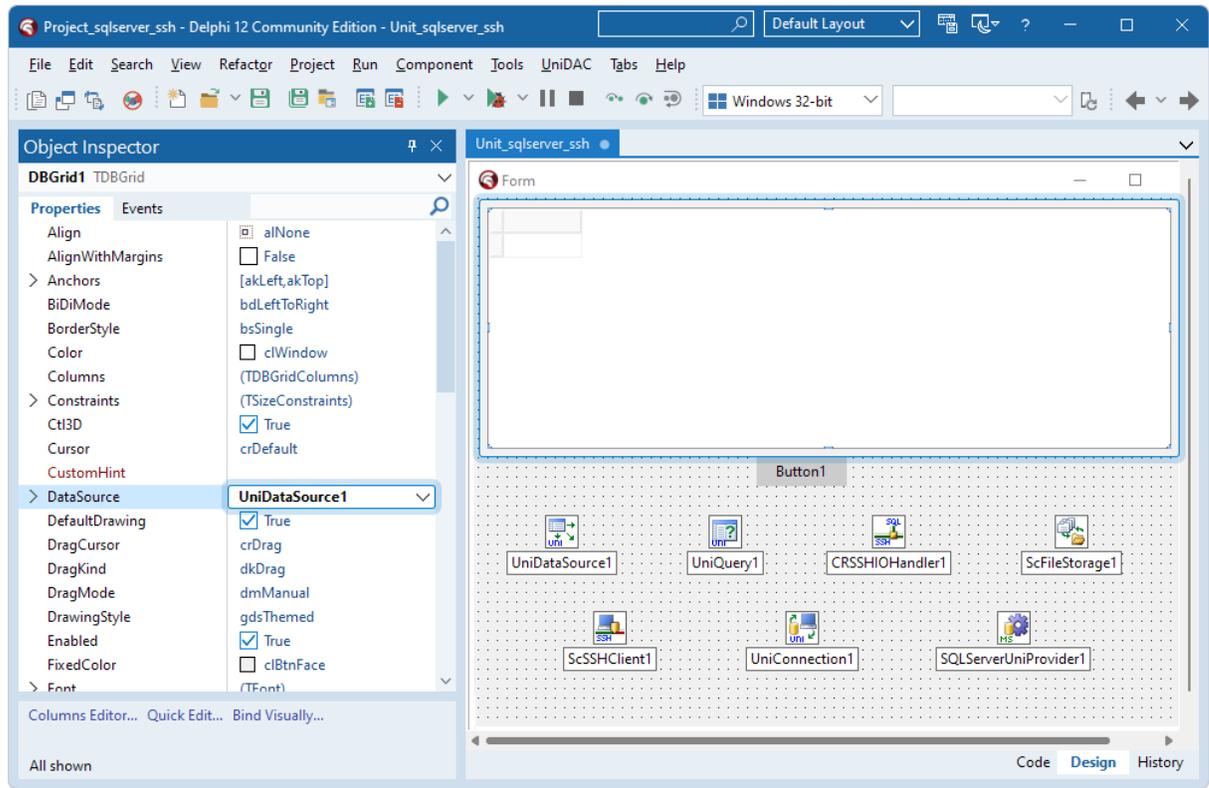
- **TDBGrid**
- **TButton**

- TUniDataSource
- TUniQuery
- TCRSSHIOHandler
- TScFileStorage
- TScSSHClient
- TUniConnection
- TSQLServerUniProvider

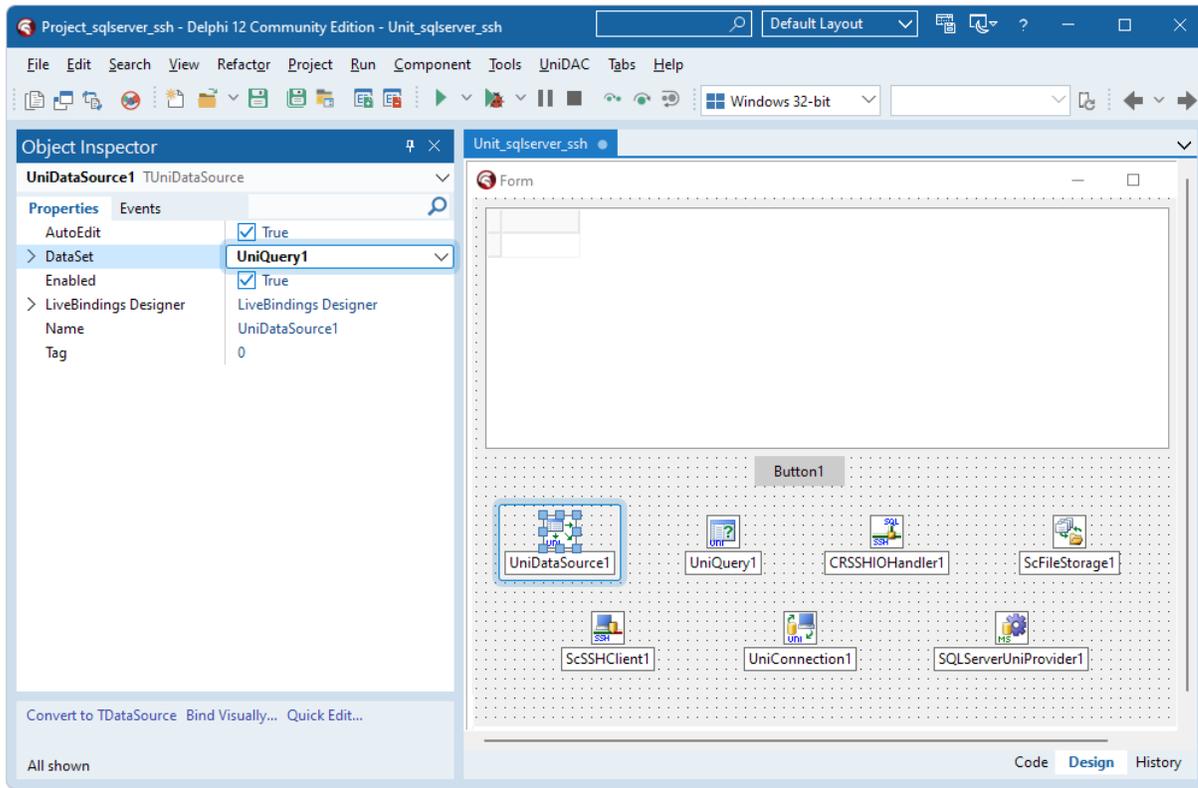
With these components, the sample application will connect to SQL Server via SSH, execute a query, and display the retrieved data in a grid.



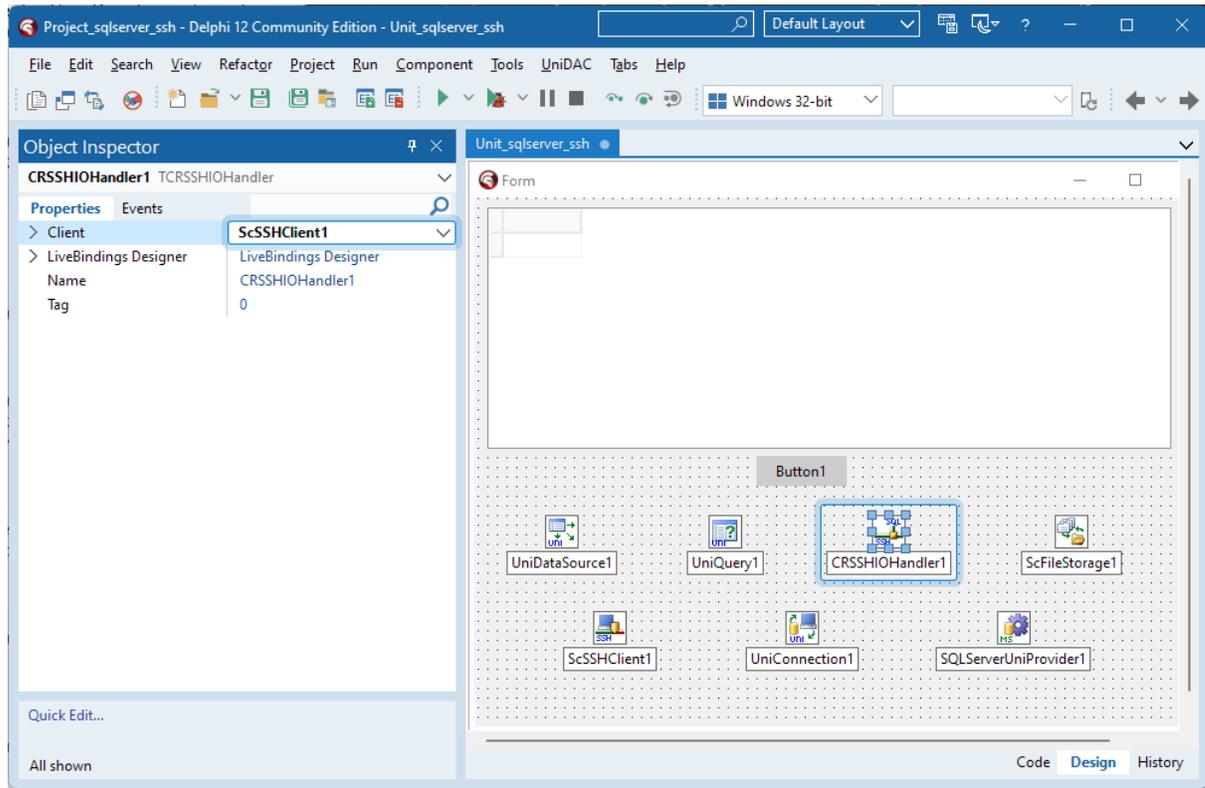
3. Select the **TDBGrid** component and, in **Object Inspector**, set the **DataSource** property to the instance of **TUniDataSource (UniDataSource1)**.



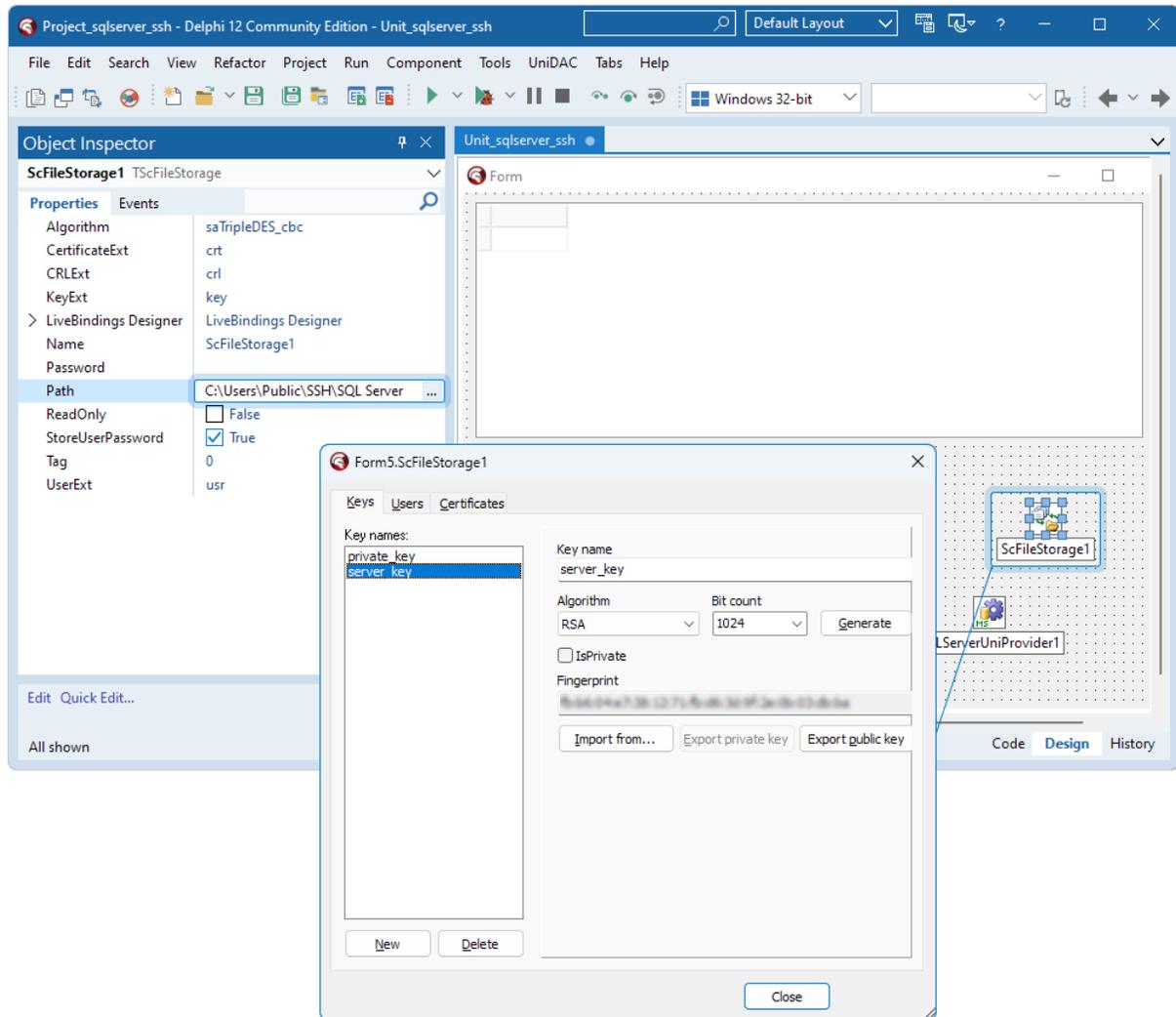
4. Select the **TUniDataSource** component and set the **DataSet** property to the instance of **TUniQuery (UniQuery1)**.



5. Select the **TCRSSHIOHandler** component and set the **Client** property to the instance of **TScSSHClient (ScSSHClient1)**.



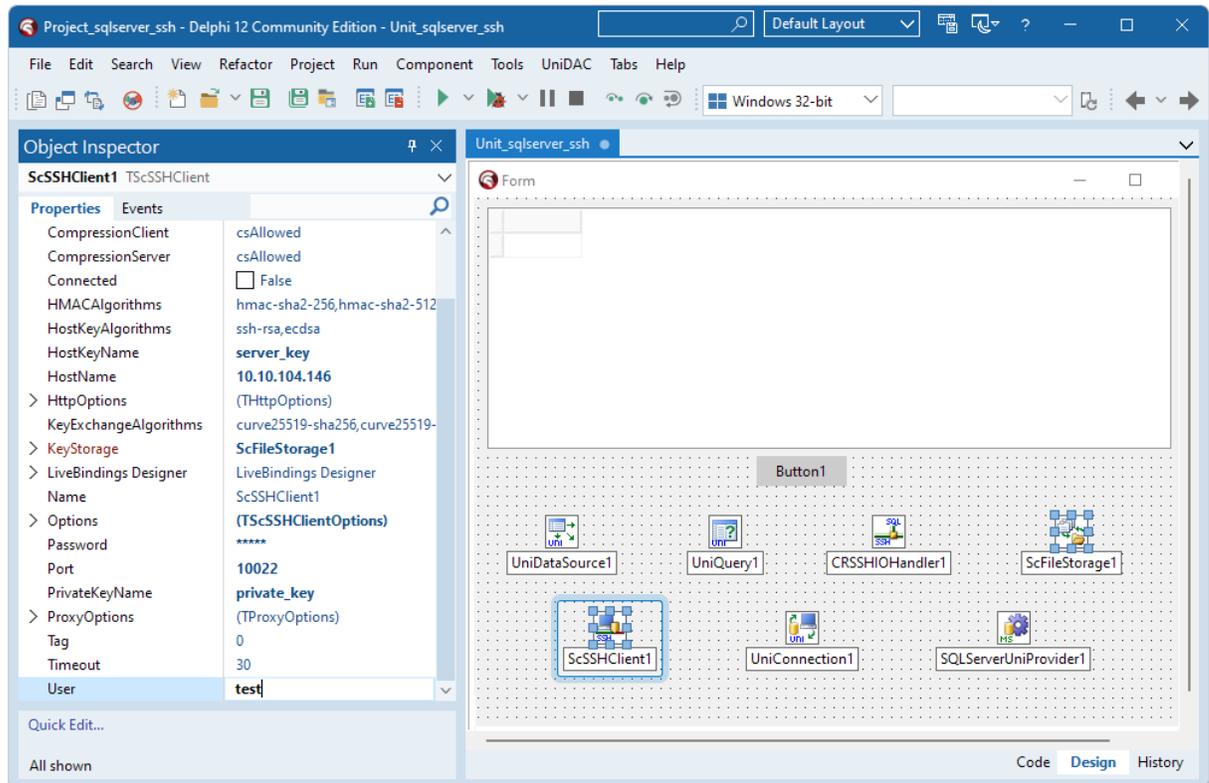
6. Select the **TScFileStorage** component and, in the **Path** property, specify the directory to store information about keys.
7. Double-click the **TScFileStorage** component and generate a pair of keys for authenticating the server by the client.



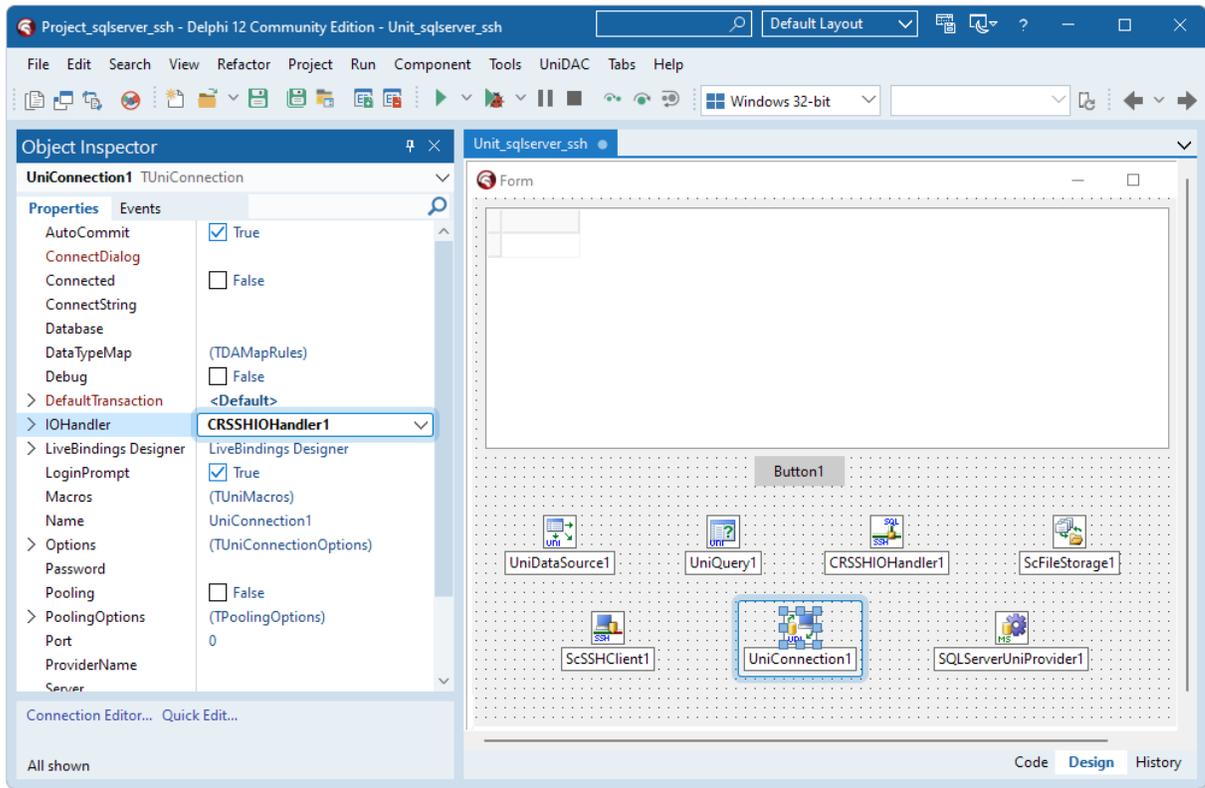
8. Select the **TScSSHClient** component and assign values to the following properties:

- **Authentication** – Select the value depending on the authentication method applicable for your SSH server: **atPassword** or **atPublicKey**.
- **HostKeyName** – Specify the filename of the SSH server public key.
- **Hostname** – Enter the host name or IP address of the SSH server.
- **KeyStorage** – Set the property to the instance of **TScFileStorage** (**ScFileStorage1**).
- **Password** – For password authentication, enter the password for the account on the SSH server.
- **Port** – Specify the SSH port.

- **PrivateKeyName** – For public key authentication, specify the filename of the client private key.
- **User** – Enter the username for the account on the SSH server.



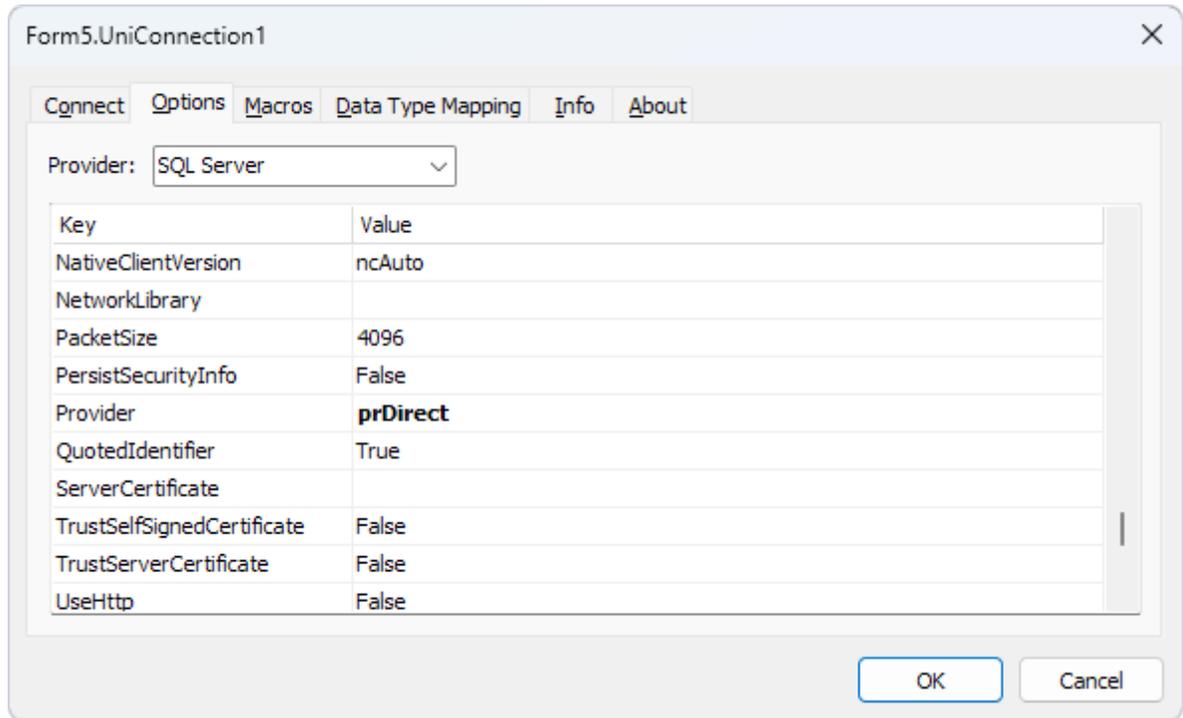
9. Select the **TUniConnection** component and set the **IOHandler** property to the instance of **TCRSSHIOHandler** (**CRSSHIOHandler1**).



10. Double-click the **TUniConnection** component.

11. On the **Options** tab, fill out the fields:

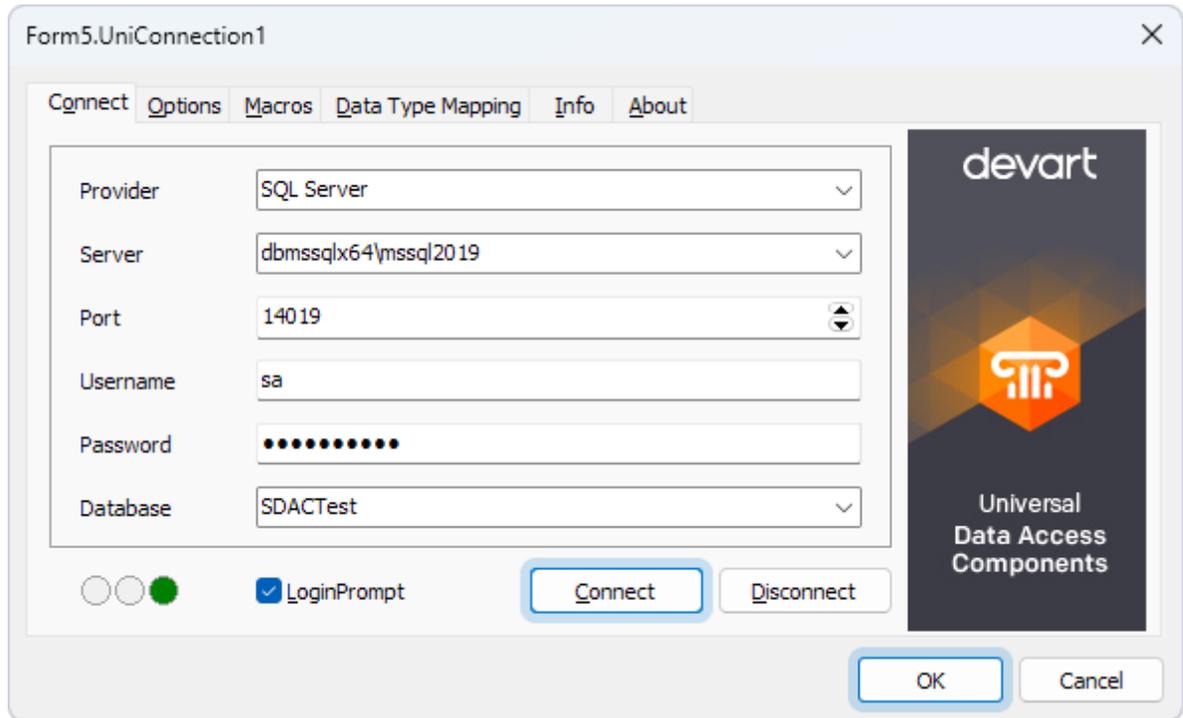
- **Provider** – Select **SQL Server**.
- **Provider (key)** – Select **prDirect**.



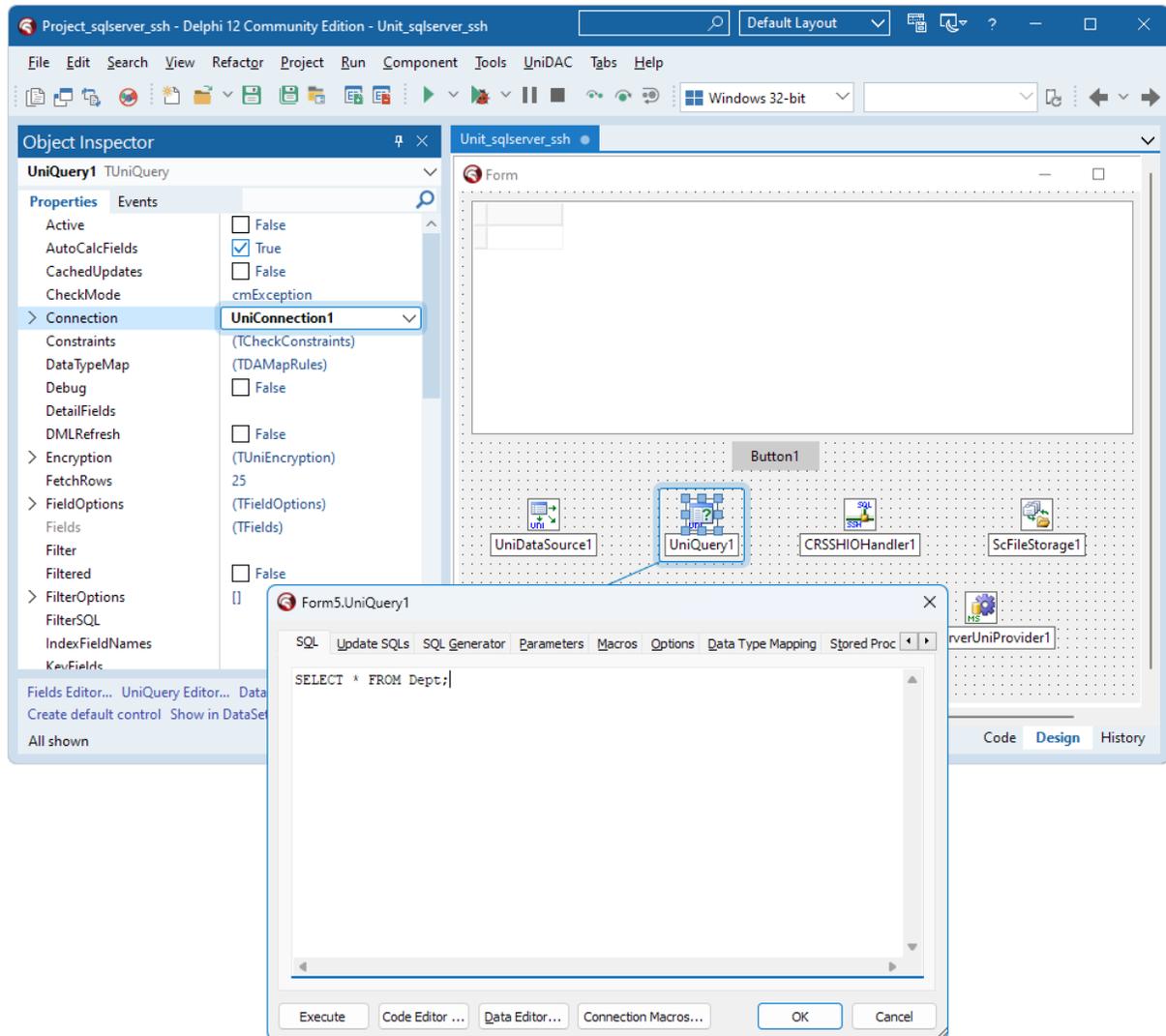
12. On the **Connect** tab, fill out the fields:

- **Provider** – Select **SQL Server**.
- **Server** – Enter the host name or IP address of the SQL Server instance.
- **Port** – Specify the port number used by the SQL Server instance.
- **Username** – Enter the username for the SQL Server account.
- **Password** – Enter the password for the SQL Server account.
- **Database** – Select the database.

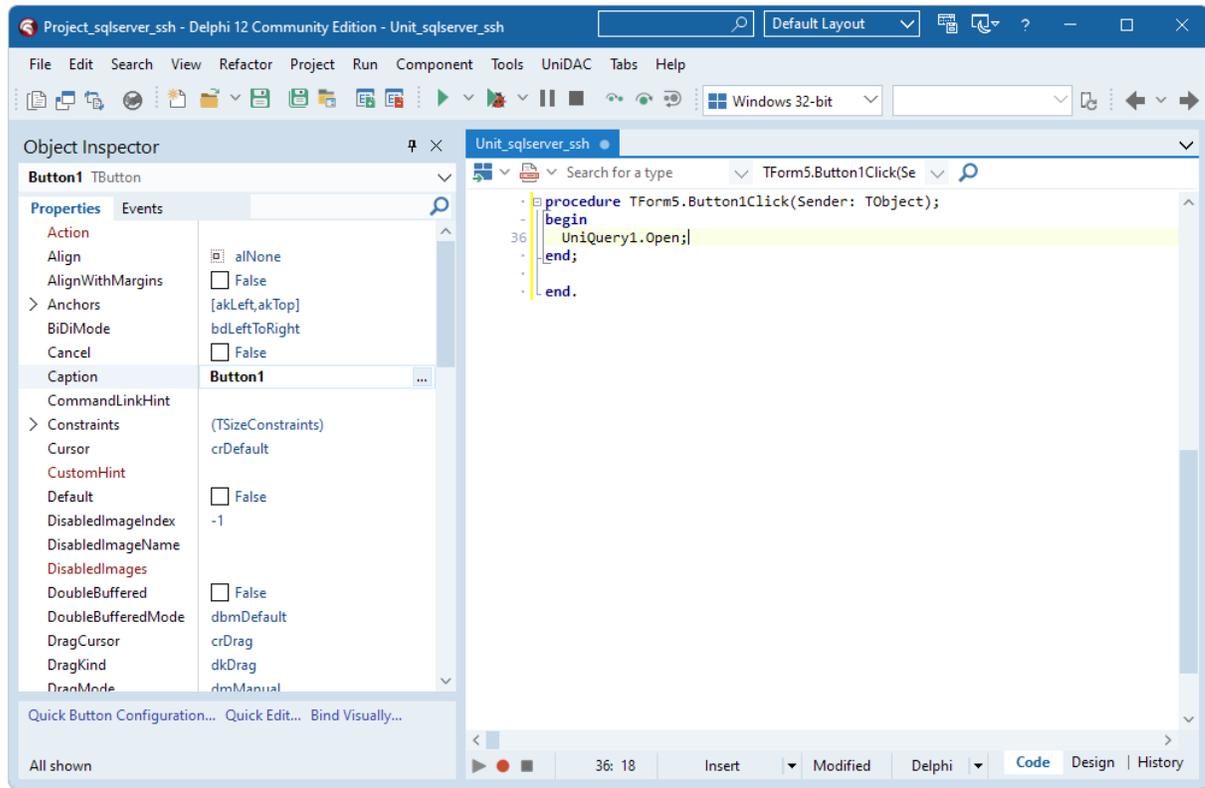
13. Click **Connect** to test the connection to SQL Server, then click **OK** to close the dialog.



14. Select the **TUniQuery** component and set the **Connection** property to the instance of **TUniConnection (UniConnection1)**.
15. Double-click the **TUniQuery** component, enter a SQL query to be run against the SQL Server database, and click **OK**.

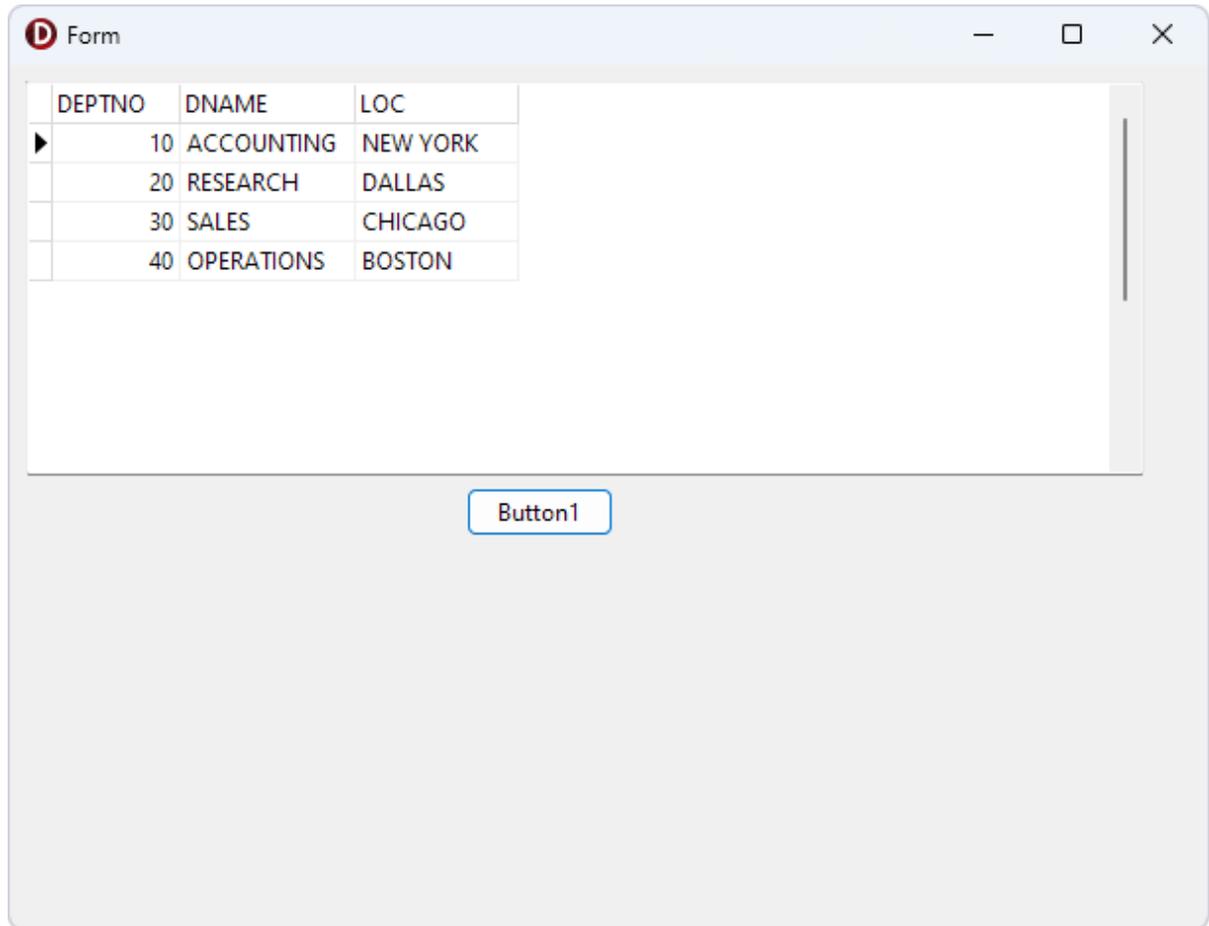


16. Double-click the **TButton** component and add code to call the **Open** method on the **TUniQuery** component instance to activate the dataset when the button is clicked.



17. Press **F9** to compile and run the application.

18. In the form that appears, click **Button1** to run the query. Data appears in the grid.



The screenshot shows a window titled "Form" with a table and a button. The table has three columns: DEPTNO, DNAME, and LOC. The data rows are:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Below the table is a button labeled "Button1".

Connect Using Any Third-Party SSH Tunnel

It is not mandatory to use SecureBridge and its components—you can use any other server that implements the SSH protocol.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.1.16.4 Network Tunneling

In typical scenarios, a client connects to a server through a direct network path. However, modern security policies or complex network topologies often require the use of a proxy or traversal through a firewall. This article describes different ways to connect to a database server using UniDAC.

- [Direct connection](#)

- [Connection through an HTTP tunnel](#)
- [Connection through a proxy and an HTTP tunnel](#)
- [Additional information](#)

Direct connection

A direct connection to the server means that the server host is accessible from the client without any additional 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 to communicate with the server. Use it whenever possible.

The following example demonstrates how to configure a direct connection:

```
UniConnection := TUniConnection.Create(self);
UniConnection.ProviderName := '<provider_name>';
UniConnection.Server := 'localhost';
UniConnection.Port := 3306;
UniConnection.Username := '<user>';
UniConnection.Password := '<password>';
UniConnection.Connect;
```

Connection through an HTTP tunnel

Sometimes, client computers are shielded by a firewall that does not allow you to connect directly to the server on the specified port. If the firewall allows HTTP connections, you can use UniDAC and HTTP tunneling software to connect to the database server.

UniDAC supports HTTP tunneling based on a PHP script.

Consider a scenario where direct access to the database server port is blocked on a remote host, and only HTTP traffic on port 80 is permitted. In such cases, web script tunneling allows remote access to the database as if it were a direct connection.

You need to deploy the `tunnel.php` script included in the provider package on the web server. It allows access to the database server using HTTP tunneling. The script must be accessible through the HTTP protocol. You can check if it is accessible using a web browser. You can find the script in the HTTP subfolder of the installed provider folder, such as `%Program Files%\Devart\UniDac for Delphi X\HTTP\tunnel.php`. The only requirement is that the server must support PHP 5.

To connect to the database via HTTP tunneling, first configure `TUniConnection` as for a direct connection from the web server. Then, set the `Protocol` option to `mpHttp` and provide the

following HTTP tunneling parameters:

Specific Option	Required	Description
HttpUrl	Yes	The URL of the tunneling PHP script. For example, if the script is in the server root, the URL can be <code>http://localhost/tunnel.php</code> .
HttpUsername, HttpPassword	No	A registered user credentials. Set these properties if access to the website folder with the script is available only for registered users authenticated with a username and password.

Connection through a proxy and an HTTP tunnel

Consider the previous case with one more complication.

The HTTP tunneling server is not directly accessible from the client computer. For example, the client address is 10.0.0.2, the server address is 192.168.0.10, and the database server is listening on port 3307. Since the client and the server are on different networks, the client can only access the server through a proxy located at 10.0.0.1 and listening on port 808. In this case, in addition to the HTTP-specific options, you need to set the proxy-specific options.

```
UniConnection := TUniConnection.Create(self);
UniConnection.ProviderName := '<provider_name>';
UniConnection.Server := '192.168.0.10';
UniConnection.Port := 3307;
UniConnection.Username := '<user>';
UniConnection.Password := '<password>';
UniConnection.SpecificOptions.Values['Protocol'] := 'mpHttp';
UniConnection.SpecificOptions.Values['HttpUrl'] := 'http://server/tunnel.php';
UniConnection.SpecificOptions.Values['ProxyHostname'] := '10.0.0.1';
UniConnection.SpecificOptions.Values['ProxyPort'] := '808';
UniConnection.SpecificOptions.Values['ProxyUsername'] := '<proxy_user>';
UniConnection.SpecificOptions.Values['ProxyPassword'] := '<proxy_password>';
UniConnection.Connect;
```

Note that when proxy-specific options are configured, UniDAC will automatically route the connection through the specified proxy server.

Additional information

Remember that traffic tunneling or encryption increases CPU usage and network load. Direct connections are recommended whenever possible.

Reserved.

5.2 Cloud Providers

5.2.1 UniDAC and BigCommerce

This article provides a brief overview of the BigCommerce cloud provider for UniDAC used to access BigCommerce from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting to BigCommerce](#)
- [Bigcommerce-specific options](#)
 - [TUniConnection](#)
 - [Proxy connection options](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

Overview

Main features of BigCommerce cloud provider are:

- Direct access to BigCommerce cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the BigCommerce cloud provider.

Compatibility

BigCommerce provider supports BigCommerce data types and API.

Requirements

Applications that use the BigCommerce cloud provider require [Devart ODBC Driver for BigCommerce](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to BigCommerce

To connect to BigCommerce using **legacy authentication** and Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Server
- Username
- AuthenticationToken

For more information on how to obtain BigCommerce AuthenticationToken, see the [article](#).

To connect to BigCommerce using the **OAuth authentication** and Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Authentication
- StoreId
- ClientId
- AccessToken

For more information on how to obtain BigCommerce AccessToken, ClientId and StoreId, see the [article](#).

BigCommerce-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
AccessToken	Used to supply a unique Access Token for your app.
Authentication	Used to specify the required BigCommerce authentication. The available values are: Basic OAuth The default value is Basic.
AuthenticationToken	Used to supply an API key to login to BigCommerce.
ClientId	Used to supply a unique Client ID for your app.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
StoreId	Used to identify the store you are logging into.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same

	way as described above for the host.
ProxyServer	If you are using Proxy for connection to your network, specify the Proxy server address in this option. To learn your Proxy server address, open Control Panel > Internet Options > Connections > LAN settings.
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all BigCommerce data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniScript component has no BigCommerce-specific options.

TUniLoader

The TUniLoader component has no BigCommerce-specific options.

TUniDump

The TUniDump component has no BigCommerce-specific options.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.2 UniDAC and Dynamics 365

This article provides a brief overview of the Dynamics 365 cloud provider for UniDAC used to access Dynamics 365 from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting to Dynamics 365](#)
- [Dynamics 365-specific options](#)
 - [TUniConnection](#)
 - [Proxy connection options](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

Overview

Main features of Dynamics 365 cloud provider are:

- Direct access to Dynamics 365 cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Dynamics 365 cloud provider.

Compatibility

Dynamics 365 provider supports Dynamics 365 Field data types and API.

Requirements

Applications that use the Dynamics 365 cloud provider require [Devart ODBC Driver for Dynamics 365](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to Dynamics 365

To connect to Dynamics 365 using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Server
- Username
- Password

If using OAuth2.0 authentication, you need to specify the **Refresh Token**. It is available when the [OAuth 2.0](#) authentication type is selected.

In the TUniConnection component, specify the following parameters:

- Server
- RefreshToken

Dynamics 365-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
Authentication	The authentication type to use when connecting to Dynamics 365. atLogin The basic user/password authentication. atOAuth The OAuth 2.0 authentication. The default value is atOAuth.
CacheMetadata	Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month. False The metadata caching is disabled. Hour Cached metadata is reset once per hour. Day Cached metadata is reset once a day (i.e. every 24 hours). Month Cached metadata is reset once per month. True Metadata caching won't reset until the driver is unloaded.
ClientId	Custom Client Id for the Dynamics 365 OAuth 2.0.
ClientSecret	Custom Client Secret for the Dynamics 365 OAuth 2.0.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
RefreshToken	The Dynamics 365 OAuth 2.0 token. Available when the OAuth 2.0 authentication type is selected.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideStrings, and TStringField is replaced by TWideStringField.

UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.
----------	---

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	If you are using Proxy for connection to your network, specify the Proxy server address in this option. To learn your Proxy server address, open Control Panel > Internet Options > Connections > LAN settings.
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all Dynamics 365 data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are

	mapped to TMemoField. The TEXT data type is always mapped to TMemoField regardless of the value of this option.
--	---

TUniScript

The TUniScript component has no Dynamics 365-specific options.

TUniLoader

The TUniLoader component has no Dynamics 365-specific options.

TUniDump

The TUniDump component has no Dynamics 365-specific options.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.3 UniDAC and FreshBooks

This article provides a brief overview of the FreshBooks cloud provider for UniDAC used to access FreshBooks from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting to FreshBooks](#)
- [FreshBooks-specific options](#)
 - [TUniConnection](#)
 - [FreshBooks Classic](#)
 - [FreshBooks New](#)
 - [Proxy connection options](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)

- [TUniScript](#)
- [TUniLoader](#)
- [TUniDump](#)

Overview

Main features of FreshBooks cloud provider are:

- Direct access to FreshBooks cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the FreshBooks provider.

Compatibility

FreshBooks provider supports supports FreshBooks data types and API.

Requirements

Applications that use the FreshBooks cloud provider require [Devart ODBC Driver for FreshBooks](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to FreshBooks

To connect to **FreshBooks Classic** using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- ApiVersion

- Server
- AuthenticationToken

For more information on how to obtain FreshBooks AuthenticationToken, see the [article](#).

To connect to **FreshBooks New** using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- ApiVersion
- CompanyName
- AccessToken

For more information on how to request FreshBooks AccessToken, see the [article](#).

FreshBooks-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
ApiVersion	Used to specify the required FreshBooks version. The available values are: Classic New The default value is Classic.
CacheMetadata	Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month. False The metadata caching is disabled. Hour Cached metadata is reset once per hour.

	Day Cached metadata is reset once a day (i.e. every 24 hours). Month Cached metadata is reset once per month. True Metadata caching won't reset until the driver is unloaded.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.

FreshBooks Classic

Option name	Description
AuthenticationToken	Authentication token is used to securely connect to your FreshBooks account.

FreshBooks New

Option name	Description
AccessToken	Access token is used to securely connect to your FreshBooks account.
CompanyName	The company name used when creating a FreshBooks account.
RefreshToken	A refresh token allows an application to obtain a new access token without prompting the user. Only one FreshBooks refresh token can be valid at any time per user per application.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	If you are using Proxy for connection to your network, specify the Proxy server address in this option.

	To learn your Proxy server address, open Control Panel->Internet Options->Connections->LAN settings.
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all FreshBooks data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniScript component has no FreshBooks-specific options.

TUniLoader

The TUniLoader component has no FreshBooks-specific options.

TUniDump

The TUniDump component has no FreshBooks-specific options.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.4 UniDAC and Google BigQuery

This article provides a brief overview of the Google BigQuery database provider for UniDAC used to access Google BigQuery from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Google Bigquery-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

Overview

Main features of Google BigQuery data access provider are:

- Direct access to Google BigQuery without client library and additional tools
- Extended SQL Syntax

The full list of Database provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Google BigQuery provider.

Compatibility

Google BigQuery provider supports Google BigQuery data types and API.

Requirements

Applications that use the Google BigQuery provider require [Devart ODBC Driver for Google BigQuery](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to Google BigQuery

To connect to Google BigQuery using **legacy authentication** and Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Project ID
- Dataset ID
- Refresh Token

For more information on how to obtain Google BigQuery Refresh Token, see the [article](#).

Google BigQuery-specific options

Though UniDAC is components that provide a unified interface to work with various database services, it also lets you tune behaviour for each server individually. For thin setup of a certain database service, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
DataSetId	Allows referring the dataset by its name.
PrimaryKeyColumns	Contains unique keys that are used to identify all existing rows in the table.
ProjectID	Used to identify the projects you have created from the existing ones.
RefreshToken	Used to supply an API key to log in to Google BigQuery.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideStrings, and TStringField is replaced by TWideStringField.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	If you are using Proxy for connection to your network, specify the Proxy server address in this option. To learn your Proxy server address, open Control Panel->Internet Options->Connections->LAN settings.
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is

	False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all Google BigQuery data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniDump component has no Google BigQuery-specific options.

TUniLoader

The TUniLoader component has no Google BigQuery-specific options.

TUniDump

The TUniDump component has no Google BigQuery-specific options.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.5 UniDAC and HubSpot

This article provides a brief overview of the HubSpot cloud provider for UniDAC used to access HubSpot from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting to HubSpot](#)
- [HubSpot-specific options](#)

- [TUniConnection](#)
- [Proxy connection options](#)
- [TUniSQL](#)
- [TUniQuery, TUniTable, TUniStoredProc](#)
- [TUniScript](#)
- [TUniLoader](#)
- [TUniDump](#)

Overview

Main features of HubSpot cloud provider are:

- Direct access to HubSpot cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the HubSpot provider.

Compatibility

HubSpot provider supports supports HubSpot data types and API.

Requirements

Applications that use the HubSpot cloud provider require [Devart ODBC Driver for HubSpot](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to HubSpot

To connect to **HubSpot** via OAuth using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- RefreshToken

For more information on how to obtain HubSpot RefreshToken, see the [article](#).

To connect to **HubSpot** via Private App API Key using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- API Key

For more information on how to get HubSpot Private App API Key, see the [article](#).

HubSpot-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
API Key	The API key authentication.
Authentication	The authentication type to use when connecting to HubSpot. Private App The Private App authentication. OAuth The OAuth 2.0 authentication. The default value is OAuth.
CacheMetadata	Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month. False The metadata caching is disabled.

	<p>Hour Cached metadata is reset once per hour.</p> <p>Day Cached metadata is reset once a day (i.e. every 24 hours).</p> <p>Month Cached metadata is reset once per month.</p> <p>True Metadata caching won't reset until the driver is unloaded.</p>
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
CustomObjects	Enables support for custom objects so that data can be loaded from and to them.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.
RefreshToken	A refresh token allows an application to obtain a new access token without prompting the user. Only one HubSpot refresh token can be valid at any time per user per application.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	<p>If you are using Proxy for connection to your network, specify the Proxy server address in this option.</p> <p>To learn your Proxy server address, open Control Panel->Internet Options->Connections->LAN settings.</p>
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all HubSpot data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniScript component has no HubSpot-specific options.

TUniLoader

The TUniLoader component has no HubSpot-specific options.

TUniDump

The TUniDump component has no HubSpot-specific options.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.6 UniDAC and Magento

This article provides a brief overview of the Magento cloud provider for UniDAC used to access Magento from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)

- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting to Magento](#)
- [Magento-specific options](#)
 - [TUniConnection](#)
 - [Proxy connection options](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

Overview

Main features of Magento cloud provider are:

- Direct access to Magento cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Magento cloud provider.

Compatibility

Magento provider supports Magento data types and API.

Requirements

Applications that use the Magento cloud provider require [Devart ODBC Driver for Magento](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to

False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to Magento

To connect to **Magento 1.x** using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- ApiVersion
- Server
- Username
- ApiKey

For more information on how to obtain an API Key while creating Magento Api User, see the [article](#).

To connect to **Magento 2.x** using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- ApiVersion
- Server
- Username
- Password

For more information on how to obtain a password while creating Magento Api User, see the [article](#).

Magento-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
ApiKey	API Key is used for secure authorization of API users to Magento store.
ApiVersion	API Version is used to specify the required Magento version. The available values are: Ver1 Ver2 The default value is Ver1.
CacheMetadata	Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month. False The metadata caching is disabled. Hour Cached metadata is reset once per hour. Day Cached metadata is reset once a day (i.e. every 24 hours). Month Cached metadata is reset once per month. True Metadata caching won't reset until the driver is unloaded.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideStrings, and TStringField is replaced by TWideStringField.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	If you are using Proxy for connection to your network, specify the

	Proxy server address in this option. To learn your Proxy server address, open Control Panel->Internet Options->Connections->LAN settings.
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all Magento data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniScript component has no Magento-specific options.

TUniLoader

The TUniLoader component has no Magento-specific options.

TUniDump

The TUniDump component has no Magento-specific options.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.7 UniDAC and Mailchimp

This article provides a brief overview of the Mailchimp cloud provider for UniDAC used to access Mailchimp from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting to Mailchimp](#)
- [Mailchimp-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

Overview

Main features of Mailchimp cloud provider are:

- Direct access to Mailchimp cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the MailChimp cloud provider.

Compatibility

Mailchimp cloud provider supports Mailchimp data types and API.

Requirements

Applications that use the Mailchimp cloud provider require [Devart ODBC Driver for Mailchimp](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to Mailchimp

To connect to Mailchimp using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- ApiKey

For more information on how to obtain an API Key, see the [article](#).

Mailchimp-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
ApiKey	Your Mailchimp API Key (token).

ApiVersion	<p>Used to specify the Mailchimp API version. The choice of the API version will impact the scope of Mailchimp objects and fields available to you.</p> <p>Supported values:</p> <p>apiVer2 Mailchimp API 2.0. will be used.</p> <p>apiVer3 (<i>default</i>) Mailchimp API 3.0. will be used.</p>
CacheMetadata	<p>Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month.</p> <p>False The metadata caching is disabled.</p> <p>Hour Cached metadata is reset once per hour.</p> <p>Day Cached metadata is reset once a day (i.e. every 24 hours).</p> <p>Month Cached metadata is reset once per month.</p> <p>True Metadata caching won't reset until the driver is unloaded.</p>
ConnectionTimeout	<p>The time to wait for a connection to open before raising an exception.</p>
MergeCustomFields	<p>Use the option to specify how custom fields(merge tags) are handled when working with the ListMembers table.</p> <p>Supported values:</p> <p>mcfNone Merge tags are not read from Mailchimp.</p> <p>mcfJoinCommon (<i>default</i>) Merge tags are read from Mailchimp, but only the tags that are defined for all the Lists are joined to other ListMembers table columns. Other tags are ignored.</p> <p>mcfJoinAll All the merge tags are joined to other ListMembers table columns. If a merge tag is not defined for the list that a list member belongs to, NULL value is returned for the corresponding column of the list member.</p>
UseMergeTagAsFieldName	<p>Determines whether to use merge tag values as column names for Mailchimp merge tags instead of merge tag labels. The default value if False.</p>
UseUnicode	<p>Enables or disables Unicode support. Affects character data</p>

	<p> fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.</p>
UTCDates	<p>Use the UTCDates option to return the datetime values from the data source as UTC values.</p>

Proxy connection options

Option name	Description
ProxyPassword	<p>If Proxy User authorization is used, specify Proxy user password in this option.</p>
ProxyPort	<p>Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.</p>
ProxyServer	<p>If you are using Proxy for connection to your network, specify the Proxy server address in this option.</p> <p>To learn your Proxy server address, open Control Panel > Internet Options > Connections > LAN settings.</p>
ProxyUser	<p>If Proxy User authorization is used, specify Proxy user name (ID) in this option.</p>

TUniSQL

Option name	Description
CommandTimeout	<p>The time to wait for a statement to be executed.</p>

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	<p>The time to wait for a statement to be executed.</p>
ExtendedFieldsInfo	<p>If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.</p>
FetchAll	<p>If True, all records of a query are requested from database server when the dataset is being opened.</p> <p>If False, records are retrieved when a data-aware component or a program requests it. The default value is False.</p>
FieldsAsString	<p>If set to True, all non-BLOB fields are handled as strings. The default value is False.</p>

UnknownAsString	If set to True, all Mailchimp data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemoField. The TEXT data type is always mapped to TMemoField regardless of the value of this option.
-----------------	--

TUniScript

The TUniScript component has no Mailchimp-specific options.

TUniLoader

The TUniLoader component has no Mailchimp-specific options.

TUniDump

The TUniDump component has no Mailchimp-specific options.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.8 UniDAC and NetSuite

This article provides a brief overview of the NetSuite cloud provider for UniDAC used to access NetSuite from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting to NetSuite](#)
- [NetSuite-specific options](#)
 - [TUniConnection](#)
 - [Proxy connection options](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)

- [TUniScript](#)
- [TUniLoader](#)
- [TUniDump](#)

Overview

Main features of NetSuite cloud provider are:

- Direct access to NetSuite cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the NetSuite cloud provider.

Compatibility

NetSuite cloud provider supports NetSuite data types and API.

Requirements

Applications that use the NetSuite cloud provider require [Devart ODBC Driver for NetSuite](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to NetSuite

To connect to NetSuite using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Username
- Password

- AccountID
- ApplicationID

For more information on how to obtain an AccountID and ApplicationID, see the [article](#).

If using token-based authentication, you need to specify the following parameters in the TUniConnection component:

- AccountID
- RoleID
- ConsumerKey
- ConsumerSecret
- Token
- TokenSecret

For more information on how to set up the token-based authentication, see the [article](#).

NetSuite-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
AccountID	Used together with User ID and Password fields to authenticate to NetSuite.
ApplicationID	Used for authentication as well as User ID, Password and Account ID.
AuthenticationType	The authentication type to use when connecting to NetSuite. atTokenBased The token-based authentication.

	<p>atBasic Regular authentication.</p> <p>OAuth The OAuth 2.0 authentication. The default value is atTokenBased.</p>
CacheMetadata	<p>Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month.</p> <p>False The metadata caching is disabled.</p> <p>Hour Cached metadata is reset once per hour.</p> <p>Day Cached metadata is reset once a day (i.e. every 24 hours).</p> <p>Month Cached metadata is reset once per month.</p> <p>True Metadata caching won't reset until the driver is unloaded.</p>
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
ConsumerKey	Consumer Key and Consumer Secret keys are private keys generated when you create an app in your NetSuite account in the <i>Setup > Integration > Integration Management > Manage Integrations > Newtab</i> . Check the Token-based Authentication box to get the Consumer Key and Consumer Secret.
ConsumerKeySecret	
CustomFields	Allows accessing custom table fields.
CustomTables	Allows accessing a custom table.
RoleID	Use the option to specify your NetSuite Role ID.
Sandbox	Allows connecting to Sandbox instead of Production data.
TokenId	Use the option to specify your NetSuite Token ID.
TokenSecret	Use the option to specify your NetSuite Token Secret.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideStrings, and TStringField is replaced by TWideStringField.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.

ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	If you are using Proxy for connection to your network, specify the Proxy server address in this option. To learn your Proxy server address, open Control Panel->Internet Options->Connections->LAN settings.
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all NetSuite data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniScript component has no NetSuite-specific options.

TUniLoader

The TUniLoader component has no NetSuite-specific options.

TUniDump

The TUniDump component has no NetSuite-specific options.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.9 UniDAC and QuickBooks

This article provides a brief overview of the QuickBooks cloud provider for UniDAC used to access QuickBooks from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [QuickBooks-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

Overview

Main features of QuickBooks cloud provider are:

- Direct access to QuickBooks cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the QuickBooks provider.

Compatibility

QuickBooks cloud provider supports QuickBooks data types and API.

Requirements

Applications that use the QuickBooks cloud provider require [Devart ODBC Driver for QuickBooks](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to QuickBooks

To connect to QuickBooks using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- CompanyId
- RefreshToken

QuickBooks-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
AccessToken	Deprecated now by QuickBooks.
AccessTokenSecret	Deprecated now by QuickBooks.
CompanyId	Used to supply your QuickBooks registered Company ID. This field is filled in automatically after you authorize to QuickBooks. To learn your Company ID, sign in to QuickBooks at intuit.com, go to Your Account settings and select Company Info tab - you Company ID is provided at the top of the appeared window.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
ConsumerKey	Deprecated now by QuickBooks.
ConsumerKeySecret	
RefreshToken	Used to specify the refresh token to generate access tokens when they expire. QuickBooks access tokens are valid for an hour.
Sandbox	Helps to build and test intergration with QuickBooks.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	If you are using Proxy for connection to your network, specify the Proxy server address in this option. To learn your Proxy server address, open Control Panel->Internet Options->Connections->LAN settings.
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all QuickBooks data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniDump component has no QuickBooks-specific options.

TUniLoader

The TUniLoader component has no QuickBooks-specific options.

TUniDump

The TUniDump component has no QuickBooks-specific options.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.10 UniDAC and Salesforce

This article provides a brief overview of the Salesforce cloud provider for UniDAC used to access Salesforce from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)

- [Requirements](#)
- [Deployment](#)
- [Connecting to Salesforce](#)
- [Salesforce-specific options](#)
 - [TUniConnection](#)
 - [Proxy connection options](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

Overview

Main features of Salesforce cloud provider are:

- Direct access to Salesforce cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Salesforce cloud provider.

Compatibility

Salesforce provider supports Salesforce data types and API.

Requirements

Applications that use the Salesforce cloud provider require [Devart ODBC Driver for Salesforce](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to

False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to Salesforce

To connect to Salesforce using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Username
- Password
- SecurityToken

If using OAuth2.0 authentication, you need to specify the **Refresh Token**. It is available when the [OAuth 2.0](#) authentication type is selected.

For more information on how to obtain Salesforce SecurityToken, see the [article](#).

In the TUniConnection component, specify the following parameters:

- Server
- RefreshToken

Salesforce-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
-------------	-------------

Authentication	<p>The authentication type to use when connecting to Salesforce.</p> <p>OAuth The OAuth 2.0 authentication.</p> <p>User ID and Password The basic user/password authentication.</p> <p>The default value is OAuth.</p>
CacheMetadata	<p>Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month.</p> <p>False The metadata caching is disabled.</p> <p>Hour Cached metadata is reset once per hour.</p> <p>Day Cached metadata is reset once a day (i.e. every 24 hours).</p> <p>Month Cached metadata is reset once per month.</p> <p>True Metadata caching won't reset until the driver is unloaded.</p>
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
IncludeDeleted	If set to True, the result set of a query will contain deleted records that are visible in the recycle bin. The default value is False.
RefreshToken	This option is available when the OAuth 2.0 authentication type is selected.
SecurityToken	Enter the security token of your Salesforce account in this field.
Server	<p>This option specifies the URL of the Salesforce server. Supported domains include:</p> <ul style="list-style-type: none"> • salesforce.com • force.com • database.com <p>By default, the login.salesforce.com domain is selected.</p>
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	If you are using Proxy for connection to your network, specify the Proxy server address in this option. To learn your Proxy server address, open Control Panel > Internet Options > Connections > LAN settings.
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.
Refresh Token	The Salesforce OAuth 2.0 token. Available when the OAuth 2.0 authentication type is selected.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all Salesforce data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniScript component has no Salesforce-specific options.

TUniLoader

The TUniLoader component has no Salesforce-specific options.

TUniDump

The TUniDump component has no Salesforce-specific options.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.11 UniDAC and Salesforce MC

This article provides a brief overview of the Salesforce MC cloud provider for UniDAC used to access Salesforce MC from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting to Salesforce MC](#)
- [Salesforce MC-specific options](#)
 - [TUniConnection](#)
 - [Proxy connection options](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

Overview

Main features of Salesforce MC cloud provider are:

- Direct access to Salesforce MC cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Salesforce MC cloud provider.

Compatibility

Salesforce MC cloud provider supports Salesforce MC data types and API.

Requirements

Applications that use the Salesforce MC cloud provider require [Devart ODBC Driver for Salesforce MC](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting to Salesforce MC

To connect to Salesforce MC using Devart ODBC Driver and **User/Password Authentication**, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Server
- Username
- Password

To connect to Salesforce MC using Devart ODBC Driver and **App Center Client**

Authentication, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Authentication
- ClientID
- ClientSecret

To connect to Salesforce MC using Devart ODBC Driver and **Server to Server**

Authentication, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Authentication
- ClientID
- ClientSecret

Salesforce MC-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
AppClientID	Used to supply Application Client ID for App Center Client authentication.
AppClientSecret	Used to supply Application center client secret for App Center Client authentication.
AppSandbox	Allows using a production or sandbox account for App Center Client authentication.
Authentication	Specifies the authentication type. atUserAndPassword Authentication with a user ID and a

	<p>password. Specify your Username and Password.</p> <p>AtAppCenterClient Authentication as an App Center Client. Specify your AppClientID and AppClientSecret.</p> <p>atServerToServer Authentication using the Server to Server option. Specify your ClientID and ClientSecret.</p> <p>The default option is atUserAndPassword.</p>
CacheMetadata	<p>Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month.</p> <p>False The metadata caching is disabled.</p> <p>Hour Cached metadata is reset once per hour.</p> <p>Day Cached metadata is reset once a day (i.e. every 24 hours).</p> <p>Month Cached metadata is reset once per month.</p> <p>True Metadata caching won't reset until the driver is unloaded.</p>
ClientID	Used to supply application center client ID for Server to Server authentication.
ClientSecret	Used to supply application center client secret for Server to Server authentication.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
ExtensionObjects	<p>Allows creating custom objects called Data Extensions. If set to True, the data extension information will be read and processed in the same way as the predefined Salesforce Marketing Cloud objects are.</p> <p>Note: Reading data extension information might take additional time.</p>
PartnerIDs	The list of specific partner accounts or business units for retrieve requests.
Subdomain	Is used to provide a subdomain (being a 28-character string starting with the letters "mc") for Server to Server authentication.
UseUnicode	Enables or disables Unicode support. Affects character data

	<p> fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.</p>
UTCDates	<p>Use the UTCDates option to return the datetime values from the data source as UTC values.</p>

Proxy connection options

Option name	Description
ProxyPassword	<p>If Proxy User authorization is used, specify Proxy user password in this option.</p>
ProxyPort	<p>Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.</p>
ProxyServer	<p>If you are using Proxy for connection to your network, specify the Proxy server address in this option.</p> <p>To learn your Proxy server address, open Control Panel->Internet Options->Connections->LAN settings.</p>
ProxyUser	<p>If Proxy User authorization is used, specify Proxy user name (ID) in this option.</p>

TUniSQL

Option name	Description
CommandTimeout	<p>The time to wait for a statement to be executed.</p>

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	<p>The time to wait for a statement to be executed.</p>
ExtendedFieldsInfo	<p>If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.</p>
FetchAll	<p>If True, all records of a query are requested from database server when the dataset is being opened.</p> <p>If False, records are retrieved when a data-aware component or a program requests it. The default value is False.</p>
FieldsAsString	<p>If set to True, all non-BLOB fields are handled as strings. The default value is False.</p>

UnknownAsString	If set to True, all Salesforce Marketing Cloud data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemoField. The TEXT data type is always mapped to TMemoField regardless of the value of this option.
-----------------	---

TUniScript

The TUniScript component has no Salesforce MC-specific options.

TUniLoader

The TUniLoader component has no Salesforce MC-specific options.

TUniDump

The TUniDump component has no Salesforce MC-specific options.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.12 UniDAC and SugarCRM

This article provides a brief overview of the SugarCRM cloud provider for UniDAC used to access SugarCRM from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting](#)
- [SugarCRM-specific options](#)
 - [TUniConnection](#)
 - [Proxy connection options](#)
 - [TUniSQL](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)

- [TUniScript](#)
- [TUniLoader](#)
- [TUniDump](#)

Overview

Main features of SugarCRM cloud provider are:

- Direct access to SugarCRM cloud databases via HTTPS
- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the SugarCRM cloud provider.

Compatibility

SugarCRM cloud provider supports SugarCRM data types and API.

Requirements

Applications that use the SugarCRM cloud provider require [Devart ODBC Driver for SugarCRM](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting

To connect to SugarCRM using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- Server
- Username

- Password

SugarCRM-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
CacheMetadata	Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month. False The metadata caching is disabled. Hour Cached metadata is reset once per hour. Day Cached metadata is reset once a day (i.e. every 24 hours). Month Cached metadata is reset once per month. True Metadata caching won't reset until the driver is unloaded.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideStrings, and TStringField is replaced by TWideStringFiled.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password

	in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	If you are using Proxy for connection to your network, specify the Proxy server address in this option. To learn your Proxy server address, open Control Panel > Internet Options > Connections > LAN settings.
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all SugarCRM data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniScript component has no SugarCRM-specific options.

TUniLoader

The TUniLoader component has no SugarCRM-specific options.

TUniDump

The TUniDump component has no SugarCRM-specific options.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.2.13 UniDAC and Zoho CRM

This article provides a brief overview of the Zoho CRM cloud provider for UniDAC used to access Zoho CRM from Delphi and Lazarus. You will find the description of some useful features and how to get started quickly.

- [Overview](#)
- [Compatibility](#)
- [Requirements](#)
- [Deployment](#)
- [Connecting](#)
- [Zoho CRM-specific options](#)
 - [TUniConnection](#)
 - [TUniSQL](#)
 - [Proxy connection options](#)
 - [TUniQuery, TUniTable, TUniStoredProc](#)
 - [TUniScript](#)
 - [TUniLoader](#)
 - [TUniDump](#)

Overview

Main features of Zoho CRM cloud provider are:

- Direct access to Zoho CRM cloud databases via HTTPS

- Extended SQL Syntax

The full list of Cloud provider features can be found on the UniDAC [features](#) page.

Both [Professional and Standard Editions](#) of UniDAC include the Zoho CRM cloud provider.

Compatibility

Zoho CRM cloud provider supports Zoho CRM data types and API.

Requirements

Applications that use the Zoho CRM cloud provider require [Devart ODBC Driver for Zoho CRM](#) to be installed on the client computer. The driver is sold and distributed separately from UniDAC.

Deployment

When an application was built without runtime packages (Link with runtime packages set to False in Project Options), you do not need to deploy any BPL files with it. For more information, see [Deployment](#).

Note that UniDAC Trial requires deployment of additional BPL files regardless of Link with runtime packages.

Connecting

To connect to Zoho CRM using Devart ODBC Driver, you should configure the driver and set up a [DSN](#). In the TUniConnection component, specify the following parameters:

- RefreshToken

For more information on how to obtain Zoho CRM Refresh token, see the [article](#).

Zoho CRM-specific options

Though UniDAC is components that provide a unified interface to work with various cloud services, it also lets you tune behaviour for each cloud individually. For thin setup of a certain cloud, UniDAC provides specific options. These options can be applied to such components as TUniConnection, TUniQuery, TUniTable, TUniStoredProc, TUniSQL, TUniScript via their SpecificOptions property. SpecificOptions is a sting list.

Below you will find the description of allowed options grouped by components.

TUniConnection

Option name	Description
AccessToken	Access Token is now deprecated by Zoho CRM.
ApiVersion	Used to specify the Zoho CRM API version. The choice of the API version will impact the syntax, output and methods available to you. apiVer2 Zoho CRM API 2.0. will be used. apiVer1 The default value. Zoho CRM API 1.0. is now deprecated by Zoho CRM.
AuthenticationToken	Authentication Token is deprecated now by Zoho CRM.
CacheMetadata	Allows caching and storing metadata in a temporary database. The option settings specify the frequency of resetting cached metadata ranging from 1 hour to 1 month. False The metadata caching is disabled. Hour Cached metadata is reset once per hour. Day Cached metadata is reset once a day (i.e. every 24 hours). Month Cached metadata is reset once per month. True Metadata caching won't reset until the driver is unloaded.
ConnectionTimeout	The time to wait for a connection to open before raising an exception.
NonApprovedRecords	Enables retrieval of non-approved records from Zoho CRM. The default value is False.
NormalizeDbNames	Replaces the . character in column names with the _ character. Enable the option for third-party tools that don't support the . character in column names.
QueryAPI	Allows querying Zoho CRM data through COQL (CRM Object Query Language). Enabling the option increases the performance of queries and operations that utilize filters. Note: Not all fields are natively supported in SELECT and WHERE. Other COQL limitations also apply.
RefreshToken	The option is available when apiVer2 is chosen. Zoho CRM API

	access tokens are valid for only an hour. To generate a new access token, use the refresh token. Enter your refresh token in the field to refresh access tokens when they expire.
Server	<p>Specifies the URL to the Zoho CRM. Now the available domains are:</p> <ul style="list-style-type: none"> • crm.zoho.com • crm.zoho.eu • crm.zoho.in • crm.zoho.com.cn <p>If the new ones appear, you can specify them by yourself in the connection string or in the Connection Editor dialog box.</p>
UseUnicode	Enables or disables Unicode support. Affects character data fetched from the server. When set to True, all character data is stored as WideString, and TStringField is replaced by TWideStringField.
UTCDates	Use the UTCDates option to return the datetime values from the data source as UTC values.

Proxy connection options

Option name	Description
ProxyPassword	If Proxy User authorization is used, specify Proxy user password in this option.
ProxyPort	Specify the Proxy port here. You can learn Proxy Port in the same way as described above for the host.
ProxyServer	<p>If you are using Proxy for connection to your network, specify the Proxy server address in this option.</p> <p>To learn your Proxy server address, open Control Panel->Internet Options->Connections->LAN settings.</p>
ProxyUser	If Proxy User authorization is used, specify Proxy user name (ID) in this option.

TUniSQL

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.

TUniQuery, TUniTable, TUniStoredProc

Option name	Description
CommandTimeout	The time to wait for a statement to be executed.
ExtendedFieldsInfo	If True, an additional query is performed to get information about the returned fields and tables they belong to. The default value is False.
FetchAll	If True, all records of a query are requested from database server when the dataset is being opened. If False, records are retrieved when a data-aware component or a program requests it. The default value is False.
FieldsAsString	If set to True, all non-BLOB fields are handled as strings. The default value is False.
UnknownAsString	If set to True, all Zoho CRM data types that are fetched as text and don't have the size limit, are mapped to TStringField with the default size 8192 bytes. If False (default value), such types are mapped to TMemofield. The TEXT data type is always mapped to TMemofield regardless of the value of this option.

TUniScript

The TUniScript component has no Zoho CRM-specific options.

TUniLoader

The TUniLoader component has no Zoho CRM-specific options.

TUniDump

The TUniDump component has no Zoho CRM-specific options.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

5.3 Database Specific Aspects of 64-bit Development

Oracle Connectivity Aspects

OCI mode:

Since at design-time Rad Studio XE 2 works only with x32 libraries and if a connection to the

server is needed at design-time, you need to install Oracle Client (x32) regardless of the intended platform. (If the x32 client is needed only for development, you can use only Oracle Instant Client). By default, UniDAC use DEFAULT of Oracle Client, that is why, if a x64 client is the default client at design-time, you need to specify a x32 client. To prevent conflicts between different versions of Oracle Client on the end-user side, you can leave the Home property empty, in this case, the default client will be used.

DIRECT mode:

Since there is no need to install Oracle Client for the DIRECT mode, the development of applications for the x64 platform does not differ from the development of application for Windows x86.

SQL Server Connectivity Aspects

If you are working in the Direct mode or developing a 32-bit application only, then the development process will not be different for you, except some peculiarities of each particular platform. But if you are developing a 64-bit application, you have to be aware of specifics of working with client libraries at design-time and run-time. To connect to a SQL Server database at design-time, you must have its 32-bit client library. You have to place it to the C:\Windows\SysWOW64 directory. This requirement flows out from the fact that RAD Studio XE2 is a 32-bit application and it cannot load 64-bit libraries at design-time. To work with a SQL Server database at run-time (64-bit application), you must have the 64-bit client library placed to the C:\Windows\System32 directory.

MySQL Connectivity Aspects

Client mode:

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

DIRECT mode:

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

InterBase and FireBird Connectivity Aspects

To work with InterBase and Firebird, UniDAC uses their client libraries (gds32.dll and fbclient.dll correspondingly). If you are developing a 32-bit application, then the development process will not be different for you, except some peculiarities of each particular platform. But if you are developing a 64-bit application, you have to be aware of specifics of working with client libraries at design-time and run-time. To connect to an InterBase or Firebird database at design-time, you must have its 32-bit client library. You have to place it to the C:\Windows\System32 directory. This requirement flows out from the fact that RAD Studio XE2 is a 32-bit application and it cannot load 64-bit libraries in design-time. To work with an InterBase or Firebird database at run-time (64-bit application), you must have the 64-bit client library placed to the C:\Windows\System32 directory.

PostgreSQL Connectivity Aspects

Since UniDAC 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.

SQLite Connectivity Aspects

Presently, developers of SQLite do not provide a ready driver for x64 platforms, that is why, for x64 applications you need to manually compile the sqlite library (for example, in MS VisualStudio). By default, the sqlite libraries must be placed to the following directories: for Win32 you need only the x32 library placed into C:\Windows\System32, and for windows x64, the x64 library should be placed to C:\Windows\System32 and the x32 library to C:\Windows\System32. >If the libraries are located as described above, you don't have to make additional settings for different target platforms when developing applications to work with the SQLite database; the required libraries will be correctly located both at design-time and run-time. Besides, when delivering your application to its end-users, you can supply the required library (x32 or x64) together with the application by placing it to the folder that contains the executable file. (If at design-time you don't need to connect to the database, then the x32 library is not needed either.)

If the libraries are located in different directories, then at design-time you will have to specify the path to the x32 library in the ClientLibrary option, and when building the final application for the x64 platform, you will have to specify the path to the x64 library.

MS Access Connectivity Aspects

When developing cross-platform application to work with the MS Access database, you should remember that it is impossible to install two (32- and 64-bit) drivers on the same system (Microsoft limitation). That is why, if you need to connect to the database at design-time, the 32-bit driver must be installed on the development computer, since Rad Studio XE 2 uses x32 libraries at design-time. If no such connection is needed, you can install the x64 MS Access driver. All the other aspects of x64 and x32 development are identical.

Other ODBC Connectivity Aspects

As regards all other providers using ODBC, for information on drivers for different platforms and specifics contact their developers.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6 Reference

This page shortly describes units that exist in UniDAC.

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 establishing HTTP

	connections.
CRXml	Description is not available at the moment.
DAAlerter	This unit contains the base class for the TUniAlerter component.
DADump	This unit contains the base class for the TUniDump component.
DALoader	This unit contains the base class for the TUniLoader component.
DAScript	This unit contains the base class for the TUniScript component.
DASQLMonitor	This unit contains the base class for the TUniSQLMonitor component.
DBAccess	This unit contains base classes for most of the components.
LiteCollation	This unit contains types for registering user-defined collations.
LiteFunction	This unit contains types for registering user-defined functions.
MemData	This unit contains classes for storing data in memory.
MemDS	This unit contains implementation of the TMemDataSet class.
OracleUniProvider	This unit contains the TOraUtils class that allows you to use features of Oracle database.
SQLiteUniProvider	This unit contains the TLiteUtils class that allows you to use features of SQLite database.
SQLServerUniProvider	This unit contains the TMSSqlUtils class that allows you to use features of

	SQL Server database.
Uni	This unit contains main components of UniDAC.
UniAlerter	This unit contains the implementation of the TUniAlerter component.
UniDacVcl	This unit contains the visual constituent of UniDAC.
UniDump	This unit contains the implementation of the TUniDump component.
UniLoader	This unit contains the implementation of the TUniLoader component.
UniProvider	This unit contains the TUniProvider class for linking the server-specific providers to application.
UniScript	This unit contains the implementation of the TUniScript component.
UniSQLMonitor	This unit contains the implementation of the TUniSQLMonitor component.
VirtualDataSet	Description is not available at the moment.
VirtualQuery	Description is not available at the moment.
VirtualTable	Description is not available at the moment.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.1 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.1.1 Classes

Classes in the **CRAccess** unit.

Classes

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.1.1.1 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.1.1.1.1 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.
---	---------------------------------

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.1.2 Types

Types in the **CRAccess** unit.

Types

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.1.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.1.3 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.1.3.1 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.
iiReadCommitted	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.
iiReadUnCommitted	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.
iiRepeatableRead	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).
iiSnapshot	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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.1.3.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.1.3.3 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

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1 Classes

Classes in the **CRBatchMove** unit.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1 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 UniDAC page.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.5 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.7 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.8 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
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.9 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.10 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.11 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.12 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.2.13 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.3 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.3.1 Execute Method

Performs the batch operation.

Class

[TCRBatchMove](#)

Syntax

```
procedure Execute;
```

Remarks

Call the Execute method to perform the batch operation.

© 1997-2025

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.2.1.1.4 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.1.1.4.1 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.

© 1997-2025

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.2.2 Types

Types in the **CRBatchMove** unit.

Types

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

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.2.1 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.2.3.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.2.3.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3 CREncryption

This unit contains base classes for data encryption.

Classes

Name	Description
TCREncryptor	The class that performs data encryption and decryption. For the list of all members of this type, see CREncryption members.

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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.1 Classes

Classes in the **CREncryption** unit.

Classes

Name	Description
TCREncryptor	The class that performs data encryption and decryption. For the list of all members of this type, see CREncryption members.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.1.1 TCREncryptor Class

The class that performs data encryption and decryption. For the list of all members of this type, see CREncryption members.

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

Unit

[CREncryption](#)

Syntax

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

© 1997-2025

Devart. All Rights Reserved.

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

6.3.1.1.1 Members

[TCREncryptor](#) class overview.

Properties

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

Methods

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

© 1997-2025

Devart. All Rights Reserved.

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

6.3.1.1.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.1.1.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.1.1.2.2 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 caBlowfish.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.1.1.2.3 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.1.1.2.4 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 in the following way: 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.1.1.2.5 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.

See Also

- [SetKey](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.1.1.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.1.1.3.1 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**Offset*

Sets a key with an offset, using which data is encrypted.

Count

Sets a key, using which data is encrypted.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)**6.3.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)**6.3.2.1 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.

See Also

- [Data Encryption](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.3.2.4 TCRInvalidHashAction Enumeration

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

Unit

[CREncryption](#)

Syntax

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

Values

Value	Meaning
ihFail	An exception is raised.
ihIgnoreError	Hash checking is not performed. No exception is raised.
ihSkipData	If hash is invalid the field value is set to Null. No exception is raised.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4 CRGrid

This unit contains the TCRDBGrid component.

Classes

Name	Description
TCRDBGrid	Extends the functionality of the TDBGrid component.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1 Classes

Classes in the **CRGrid** unit.

Classes

Name	Description
TCRDBGrid	Extends the functionality of the TDBGrid component.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1 TCRDBGrid Class

Extends the functionality of the TDBGrid component.

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

Unit

[CRGrid](#)

Syntax

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

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.4.1.1.1 Members

[TCRDBGrid](#) class overview.

Properties

Name	Description
Columns	To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.
Filtered	Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.
LevelDelimiterChar	Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.
OnMemoClick	Write an OnMemoClick event handler to provide custom processing of Memo fields instead of built-in Memo field editor.
OptionsEx	OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.
OptionsMenu	TCRDBGrid has local menu is situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User

can change content of this menu using OptionsMenu property.

Methods

Name	Description
ActivateFilterEdit	Use ActivateFilterEdit at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.
ActivateSearchEdit	Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.
AdjustColumns	Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.
ApplyFilter	Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.
CalcTitleLevel	Call CalcTitleLevel method to set Top and Bottom values of aRect parameter record depending on Level parameter value.
ClearFilters	Call CalcFilters method to clear all filter statements for grid columns.
ClearSorting	Call ClearSorting method to discard sorting previously

	applied to the grid data.
DataChanged	Call DataChanged method to update displayed data and status information for the grid.
GetGridSize	Call GetGridSize method to obtain width of a client area for the grid component.
GetTitleLevel	GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.

Events

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.2 Properties

Properties of the **TCRDBGrid** class.

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

Public

Name	Description
OptionsMenu	TCRDBGrid has local menu is situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User can change content of this menu using OptionsMenu property.

Published

Name	Description
Columns	To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.
Filtered	Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.
LevelDelimiterChar	Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.
OnMemoClick	Write an OnMemoClick event handler to provide custom processing of Memo fields instead of built-in Memo field editor.
OptionsEx	OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

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

6.4.1.1.2.1 Columns Property

To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.

Class

[TCRDBGrid](#)

Syntax

```
property columns: TCRDBGridColumns stored False;
```

Remarks

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.2.2 Filtered Property

Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.

Class

[TCRDBGrid](#)

Syntax

```
property Filtered: boolean default True;
```

Remarks

Conditional statements of all columns are combined to form a single filter.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.2.3 LevelDelimiterChar Property

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

Class

[TCRDBGrid](#)

Syntax

```
property LevelDelimiterChar: char default '|';
```

© 1997-2025

Devart. All Rights Reserved.

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

6.4.1.1.2.4 OnMemoClick Property

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

Class

[TCRDBGrid](#)

Syntax

```
property OnMemoClick: TOnMemoClick;
```

© 1997-2025

Devart. All Rights Reserved.

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

6.4.1.1.2.5 OptionsEx Property

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

Class

[TCRDBGrid](#)

Syntax

```
property OptionsEx: TCRDBGridOptionsEx default [dgeEnableSort,
```

```
dgeLocalFilter, dgeLocalSorting, dgeRecordCount];
```

Remarks

Following values are supported for the OptionsEx set:

dgeEnableSort - enables sorting of records by the specified column clicking on the column title.

dgeFilterBar - shows filter bar below grid caption where filter expressions for every column are entered manually.

dgeLocalFilter - records are filtered using associated dataset component facilities. Otherwise database server processes all filter operations.

dgeLocalSorting - records are sorted using associated dataset component facilities. Otherwise database server processes all sorting operations.

dgeRecordCount - displays status bar with current record number in respect to the total number of records. This option is mutually exclusive with *dgeSummary* option.

dgeSearchBar - shows search bar below grid caption where incremental search expression is entered for every column.

dgeStretch - makes all columns adjust their widths so that they fit onto entire grid area.

dgeSummary - displays status bar with summary information for every column. See *TCRColumn.SummaryMode* property on description of available options for individual columns. *dgeSummary* is mutually exclusive with *dgeRecordCount* option.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.2.6 OptionsMenu Property

TCRDBGrid has local menu is situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User can change content of this menu using OptionsMenu property.

Class

[TCRDBGrid](#)

Syntax

```
property OptionsMenu: TPopupMenu;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.3 Methods

Methods of the **TCRDBGrid** class.

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

Public

Name	Description
ActivateFilterEdit	Use ActivateFilterEdit at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.
ActivateSearchEdit	Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.
AdjustColumns	Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.
ApplyFilter	Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.
CalcTitleLevel	Call CalcTitleLevel method

	to set Top and Bottom values of aRect parameter record depending on Level parameter value.
ClearFilters	Call CalcFilters method to clear all filter statements for grid columns.
ClearSorting	Call ClearSorting method to discard sorting previously applied to the grid data.
DataChanged	Call DataChanged method to update displayed data and status information for the grid.
GetGridSize	Call GetGridSize method to obtain width of a client area for the grid component.
GetTitleLevel	GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.3.1 ActivateFilterEdit Method

Use ActivateFilterEdit at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.

Class

[TCRDBGrid](#)

Syntax

```
procedure ActivateFilterEdit(Column: TColumn);
```

Parameters

Column

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.3.2 ActivateSearchEdit Method

Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.

Class

[TCRDBGrid](#)

Syntax

```
procedure ActivateSearchEdit(Column: TColumn);
```

Parameters

Column

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.3.3 AdjustColumns Method

Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.

Class

[TCRDBGrid](#)

Syntax

```
procedure AdjustColumns;
```

Remarks

Visual feedback is immediate if OptionsEx property includes dgeLocalFilter option.

Note: Since filtering is performed at session level then other data-aware controls may also be updated.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.3.4 ApplyFilter Method

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

Class

[TCRDBGrid](#)

Syntax

```
procedure ApplyFilter;
```

Remarks

Visual feedback is immediate if OptionsEx property includes dgeLocalFilter option.

Note: Since filtering is performed at session level then other data-aware controls may also be updated.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.3.5 CalcTitleLevel Method

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

Class

[TCRDBGrid](#)

Syntax

```
procedure CalcTitleLevel(Level: integer; var aRect: TRect);
```

Parameters

Level

aRect

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.4.1.1.3.6 ClearFilters Method

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

Class

[TCRDBGrid](#)

Syntax

```
procedure ClearFilters;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.3.7 ClearSorting Method

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

Class

[TCRDBGrid](#)

Syntax

```
procedure ClearSorting;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.3.8 DataChanged Method

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

Class

[TCRDBGrid](#)

Syntax

```
procedure DataChanged;
```

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.4.1.1.3.9 GetGridSize Method

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

Class

[TCRDBGrid](#)

Syntax

```
function GetGridSize: integer;
```

Remarks

Client area accommodates cell values only without column grid lines.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.3.10 GetTitleLevel Method

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

Class

[TCRDBGrid](#)

Syntax

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

Parameters

Level

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.4 Events

Events of the **TCRDBGrid** class.

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

Published

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

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.4.1.1.4.1 OnGetCellParams Event

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

Class

[TCRDBGrid](#)

Syntax

```
property OnGetCellParams: TGetCellParamsEvent;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5 CRVio

This unit contains classes for establishing 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.1 THttpOptions Class

This class is used to establish an HTTP connection.

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

Unit

[CRVio](#)

Syntax

```
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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.1.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.1.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.1.2.1 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.1.2.2 Password Property

Holds the password for HTTP authorization.

Class

[THttpOptions](#)

Syntax

```
property Password: string;
```

Remarks

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

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.1.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.1.2.4 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 UniDAC bypasses walking the certificate chain to verify the certificate. The default value is False.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.1.2.5 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.1.2.6 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.2 TProxyOptions Class

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

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

Unit

[CRVio](#)

Syntax

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

Remarks

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

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.2.1 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.2.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.2.2.1 Hostname Property

Holds the hostname or IP address of the proxy server.

Class

[TProxyOptions](#)

Syntax

```
property Hostname: string;
```

Remarks

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.2.2.2 Password Property

Holds the proxy password.

Class

[TProxyOptions](#)

Syntax

```
property Password: string;
```

Remarks

The Password property holds the proxy password.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.2.2.3 Port Property

Holds the port number of the proxy server.

Class

[TProxyOptions](#)

Syntax

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

Remarks

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.1.2.2.4 Username Property

Holds the proxy username.

Class

[TProxyOptions](#)

Syntax

```
property Username: string;
```

Remarks

The Username property holds the proxy username.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.2 Enumerations

Enumerations in the **CRVio** unit.

Enumerations

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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.5.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.6 CRXml

6.6.1 Structs

Structs in the **CRXml** unit.

Structs

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

[TAttribute](#)

TAttribute is not used in UniDAC.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.6.1.1 TAttribute Record

TAttribute is not used in UniDAC.

Unit

CRXm1

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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7 DAAlerter

This unit contains the base class for the TUniAlerter 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.
TAlerterEventEvent	This type is used for the E:Devart.UniDac.TUniAlerter.OnEvent event.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1 Classes

Classes in the **DAAlerter** unit.

Classes

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.1 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.2 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.2.1 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.2.3 Connection Property

Used to specify the connection for `TDAALerter`.

Class

[TDAALerter](#)

Syntax

```
property Connection: TCustomDACConnection;
```

Remarks

Use the `Connection` property to specify the connection for `TDAALerter`.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.3.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.3.3 Stop Method

Stops waiting process.

Class

[TDAAlerter](#)

Syntax

```
procedure Stop;
```

Remarks

Call Stop method to end waiting process.

See Also

- [Start](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.4 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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.1.1.4.1 OnError Event

Occurs if an exception occurs in waiting process

Class

[TDAAlert](#)

Syntax

```
property OnError: TAlertErrorEvent;
```

Remarks

The OnError event occurs if an exception occurs in waiting process. Alert stops in this case. The exception can be accessed using the E parameter.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.2 Types

Types in the **DAlert** unit.

Types

Name	Description
TAlertErrorEvent	This type is used for the TDAAlert.OnError event.
TAlertEventEvent	This type is used for the E:Devart.UniDac.TUniAlert.r.OnEvent event.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.2.1 TAlerterErrorEvent Procedure Reference

This type is used for the TDAAlerter.OnError event.

Unit

[DAAlerter](#)

Syntax

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

Parameters

Sender

An object that raised the event.

E

Exception object.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.7.2.2 TAlerterEventEvent Procedure Reference

This type is used for the E:Devart.UniDac.TUniAlerter.OnEvent event.

Unit

[DAAlerter](#)

Syntax

```
TAlerterEventEvent = procedure (Sender: TDAAlerter; const
EventName: string; const Message: string) of object;
```

Parameters

Sender

An object that raised the event.

EventName

A name of event (alert or pipe).

Message

The content of message waiting process receives.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8 DADump

This unit contains the base class for the TUniDump 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.1 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 , M:Devart.Dac.TDADump.BackupToFile(System.String)

	or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.
OnError	Occurs when server raises some error on TDADump.Restore .
OnRestoreProgress	Occurs to indicate the TDADump.Restore , TDADump.RestoreFromFile , or TDADump.RestoreFromStream method execution progress.

© 1997-2025

Devart. All Rights Reserved.

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

6.8.1.1.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.2.1 Connection Property

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

Class

[TDADump](#)

Syntax

```
property Connection: TCustomDACConnection;
```

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.

See Also

- [TCustomDACConnection](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.2.2 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 UniDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.2.3 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 CREATE 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.2.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.2.5 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.3.3 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](#)

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.3.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.3.5 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.3.6 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](#)

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.3.7 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.4 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 server 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.4.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.8.1.1.4.2 OnError Event

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

Class

[TDADump](#)

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

The OnError event occurs when server 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.1.4.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.2 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);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.2.1 Members

[TDADumpOptions](#) class overview.

Properties

Name	Description
AddDrop	Used to add DROP statements to a script before CREATE 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.2.2 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 CREATE 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.2.2.1 AddDrop Property

Used to add DROP statements to a script before CREATE statements.

Class

[TDADumpOptions](#)

Syntax

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

Remarks

Use the AddDrop property to add DROP statements to a script before CREATE statements.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.2.2.2 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');
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.2.2.3 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.1.2.2.4 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.8.2.1 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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.8.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9 DALoader

This unit contains the base class for the TUniLoader 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.1.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.1.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.1.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.1.2.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.2 TDAColumns Class

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

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

Unit

[DALoader](#)

Syntax

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

Remarks

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

See Also

- [TDALoader](#)
- [TDAColumn](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.2.1 Members

[TDAColumns](#) class overview.

Properties

Name	Description
Items	Used to access individual columns.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.2.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.2.2.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.1 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 TCustomDAConnection 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.2 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 TCustomDAConnection in which TDALoader will be executed.
TableName	Used to specify the name of the table to which data will be loaded.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.2.1 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.2.2 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.2.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.3 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](#)

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.9.1.3.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.3.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.3.4 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.

© 1997-2025

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](#)

© 1997-2025

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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.4.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.4.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.3.4.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.4 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);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.4.1 Members

[TDALoaderOptions](#) class overview.

Properties

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.4.2 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 UniDAC to fill the buffer with null values after

loading a row to the database.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.1.4.2.1 UseBlankValues Property

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.2 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.
--------------------------------------	---

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.2.1 TDAPutDataEvent Procedure Reference

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

Unit

[DALoader](#)

Syntax

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

Parameters

Sender

An object that raised the event.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.2.2 TGetColumnDataEvent Procedure Reference

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

Unit

[DALoader](#)

Syntax

```
TGetColumnDataEvent = procedure (Sender: Tobject; Column: TDAColumn; Row: integer; var Value: variant; var IsEOF: boolean) of object;
```

Parameters

Sender

An object that raised the event.

Column

Points to [TDAColumn](#) object that corresponds to the current loading column.

Row

Indicates the current loading record.

Value

Holds column values.

IsEOF

True, if data loading needs to be stopped.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.9.2.3 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10 DAScript

This unit contains the base class for the TUniScript component.

Classes

Name	Description
TDAScript	Makes it possible to execute several SQL statements one by one.

TDASStatement	This class has attributes and methods for controlling single SQL statement of a script.
TDASStatements	Holds a collection of TDASStatement objects.

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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1 Classes

Classes in the **DAScript** unit.

Classes

Name	Description
TDAScript	Makes it possible to execute several SQL statements one by one.
TDASStatement	This class has attributes and methods for controlling single SQL statement of a

	script.
TDAStatements	Holds a collection of TDAStatements objects.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.1 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 server raises an error.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.1 Connection Property

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

Class

[TDAScript](#)

Syntax

```
property Connection: TCustomDACConnection;
```

Remarks

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

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

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

See Also

- [TCustomDACConnection](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.3 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 UniDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.5 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.7 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.8 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.9 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.10 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.11 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.12 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.2.13 Statements Property

Contains a list of statements obtained from the SQL property.

Class

[TDAScript](#)

Syntax

```
property Statements: TDASentences;
```

Remarks

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

For example, consider the following script:

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

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

Example

You can use the Statements property in the following way:

```

procedure TForm1.Button1Click(Sender: TObject);
var
    i: integer;
begin
    with Script do
        begin
            for i := 0 to Statements.Count - 1 do
                if Copy(Statements[i].SQL, 1, 6) <> 'CREATE' then
                    Statements[i].Execute;
            end;
        end;
end;

```

See Also

- [TDAStatements](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.3.1 BreakExec Method

Stops script execution.

Class

[TDAScript](#)

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to stop script execution.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.3.3 Execute Method

Executes a script.

Class

[TDA Script](#)

Syntax

```
procedure Execute; virtual;
```

Remarks

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

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.3.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.3.5 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 server raises an error, the OnError event occurs.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.3.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.3.7 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.3.8 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.4 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 server raises an error.

See Also

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

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.4.1 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](#)

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.1.4.2 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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.10.1.1.4.3 OnError Event

Occurs when server raises an error.

Class

[TDAScript](#)

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

Occurs when server 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2 TDASentence Class

This class has attributes and methods for controlling single SQL statement of a script.

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

Unit

[DAScript](#)

Syntax

```
TDASentence = class(TCollectionItem);
```

Remarks

TDAScript contains SQL statements, represented as TDASentence objects. The

TDAScript class has attributes and methods for controlling single SQL statement of a script.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.1 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 TDAScript 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2 Properties

Properties of the **TDASStatement** class.

For a complete list of the **TDASStatement** class members, see the [TDASStatement 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 TDAScript 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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2.1 EndLine Property

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

Class

[TDAStructure](#)

Syntax

```
property EndLine: integer;
```

Remarks

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2.2 EndOffset Property

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

Class

[TDAStructure](#)

Syntax

```
property EndOffset: integer;
```

Remarks

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.10.1.2.2.3 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2.4 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2.5 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2.7 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2.8 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2.9 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.2.10 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.2.3.1 Execute Method

Executes a statement.

Class

[TDASStatement](#)

Syntax

```
procedure Execute;
```

Remarks

Use the Execute method to execute a statement.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.3 TDASStatements Class

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

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

Unit

[DAScript](#)

Syntax

```
TDASStatements = class(TCollection);
```

Remarks

Each TDASStatements holds a collection of [TDASStatement](#) objects. TDASStatements maintains

an index of the statements in its `Items` array. The `Count` property contains the number of statements in the collection. Use `TDAStatements` class to manipulate script SQL statements.

See Also

- [TDAScript](#)
- [TDAStatement](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.3.1 Members

[TDAStatements](#) class overview.

Properties

Name	Description
Items	Used to access separate script statements.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.1.3.2.1 Items Property(Indexer)

Used to access separate script statements.

Class

[TDAStatements](#)

Syntax

```
property Items[Index: Integer]: TDASatement; default;
```

Parameters

Index

Holds the index value.

Remarks

Use the Items property to access individual script statements. The value of the Index parameter corresponds to the Index property of [TDASatement](#).

See Also

- [TDASatement](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.2.2 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.2.3 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.3 Enumerations

Enumerations in the **DAScript** unit.

Enumerations

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

exits.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.10.3.1 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11 DASQLMonitor

This unit contains the base class for the TUniSQLMonitor 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.

© 1997-2025

Devart. All Rights Reserved.

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

6.11.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.1.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.1.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.1.2.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.1.2.2 DBMonitorOptions Property

Used to set options for dbMonitor.

Class

[TCustomDASQLMonitor](#)

Syntax

```
property DBMonitorOptions: TDBMonitorOptions;
```

Remarks

Use DBMonitorOptions to set options for dbMonitor.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.11.1.1.2.3 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](#)

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.1.2.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.1.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.1.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.2 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);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.2.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.2.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.2.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.2.2.3 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.1.2.2.4 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.2.1 TDATraceFlags Set

Represents the set of [TDATraceFlag](#).

Unit

[DASQLMonitor](#)

Syntax

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.2.2 TMonitorOptions Set

Represents the set of [TMonitorOption](#).

Unit

[DASQLMonitor](#)

Syntax

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.3 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 displayed.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.11.3.1 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.

© 1997-2025
Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.11.3.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12 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.
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.

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.On

	Error 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
ChangeCursor	When set to True allows data access components to change screen cursor for the execution time.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.1.1 Members

[EDAEError](#) class overview.

Properties

Name	Description
Component	Contains the component that caused the error.
ErrorCode	Determines the error code returned by the server.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.1.2 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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.1.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.1.2.2 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 server. This value is always positive.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.2.1 Members

[TCRDataSource](#) class overview.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.12.1.3.2.2 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.3 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.4 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 UniDAC installation directory.

See Also

- [GetServerList](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.5 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.7 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.8 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.9 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.10 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.2.11 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.3.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.3.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4 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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.4.1 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.
PerformConnect	Establishes a connection to the server regardless of the LoginPrompt property.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2 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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.1 ConnectDialog Property

Allows to link a [TCustomConnectDialog](#) component.

Class

[TCustomDAConnection](#)

Syntax

```
property ConnectDialog: TCustomConnectDialog;
```

Remarks

Use the ConnectDialog property to assign to connection a [TCustomConnectDialog](#) component.

See Also

- [TCustomConnectDialog](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.2 ConnectString Property

Used to specify the connection information, such as: UserName, Password, Server, etc.

Class

[TCustomDACConnection](#)

Syntax

```
property ConnectString: string stored False;
```

Remarks

UniDAC 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
```

See Also

- [Password](#)
- [Username](#)
- [Server](#)
- [Connect](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.3 ConvertEOL Property

Allows customizing line breaks in string fields and parameters.

Class

[TCustomDACConnection](#)

Syntax

```
property ConvertEOL: boolean default False;
```

Remarks

Affects the line break behavior in string fields and parameters. When fetching strings (including the TEXT fields) with ConvertEOL = True, dataset converts their line breaks from the LF to CRLF form. And when posting strings to server with ConvertEOL turned on, their line breaks are converted from CRLF to LF form. By default, strings are not converted.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.4 InTransaction Property

Indicates whether the transaction is active.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.5 LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Class

[TCustomDAConnection](#)

Syntax

```
property LoginPrompt default DefValLoginPrompt;
```

Remarks

Specifies whether a login dialog appears immediately before opening a new connection. If [ConnectDialog](#) is not specified, the default connect dialog will be shown. The connect dialog will appear only if the UniDacVcl unit appears to the uses clause.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.6 Options Property

Specifies the connection behavior.

Class

[TCustomDAConnection](#)

Syntax

```
property options: TDACConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of the connection.

Descriptions of all options are in the table below.

Option Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.
DefaultSortType	Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode	Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.

KeepDesignConnected	Prevents 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 the connection is lost.

See Also

- [Disconnected Mode](#)
- [Working in an Unstable Network](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.7 Password Property

Serves to supply a password for login.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.8 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](#)
- [Using Connection Pooling](#)

Devart. All Rights Reserved.

6.12.1.4.2.9 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.10 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.2.11 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, UniDAC 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3 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.
PerformConnect	Establishes a connection to the server regardless of the LoginPrompt property.
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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.1 ApplyUpdates Method

Applies changes in datasets.

Class

[TCustomDAConnection](#)

Overload List

Name	Description
ApplyUpdates	Applies changes from all active datasets.
ApplyUpdates(const DataSets: array of	Applies changes from the specified

[TCustomDADataSet\)](#)

datasets.

© 1997-2025

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](#)

© 1997-2025

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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.2 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](#)

- [TCustomUniDataSet.SpecificOptions](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.3 Connect Method

Establishes a connection to the server.

Class

[TCustomDACConnection](#)

Syntax

```
procedure Connect; overload; procedure Connect(const
ConnectionString: string); overload;
```

Remarks

Call the Connect method to establish a connection to the server. Connect sets the Connected property to True. If LoginPrompt is True, Connect prompts user for login information as required by the server, or otherwise tries to establish a connection using values provided in the [Username](#), [Password](#), and [Server](#) properties.

See Also

- [Disconnect](#)
- [Username](#)
- [Password](#)
- [Server](#)
- [ConnectDialog](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.4 CreateSQL Method

Creates a component for queries execution.

Class

[TCustomDAConnection](#)

Syntax

```
function CreateSQL: TCustomDASQL; virtual;
```

Return Value

A new instance of the class.

Remarks

Call the CreateSQL to return a new instance of the [TCustomDASQL](#) class and associates it with this connection object. In the descendant classes this method should be overridden to create an appropriate descendant of the TCustomDASQL component.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.5 Disconnect Method

Performs disconnect.

Class

[TCustomDAConnection](#)

Syntax

```
procedure Disconnect;
```

Remarks

Call the Disconnect method to drop a connection to database. Before the connection component is deactivated, all associated datasets are closed. Calling Disconnect is similar to setting the Connected property to False.

In most cases, closing a connection frees system resources allocated to the connection.

If user transaction is active, e.g. the [InTransaction](#) flag is set, calling to Disconnect the current user transaction.

Note: If a previously active connection is closed and then reopened, any associated datasets must be individually reopened; reopening the connection does not automatically reopen

associated datasets.

See Also

- [Connect](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.6 ExecProc Method

Allows to execute stored procedure or function providing its name and parameters.

Class

[TCustomDACConnection](#)

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:= MyUniConnection1.ExecProc('My_Sum', [10, 20]);
Label2.Caption:= MyUniConnection1.ParamByName('Result').AsString;
```

See Also

- [ExecProcEx](#)
- [ExecSQL](#)
- [ExecSQLEx](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.7 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.8 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.9 ExecSQLEx Method

Executes any SQL statement outside the TQuery or TSQL components.

Class

[TCustomDACConnection](#)

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

```
UniConnection.ExecSQLEx('begin :A:= :B + :C; end;',  
    ['A', 0, 'B', 5, 'C', 3]);  
A:= UniConnection.ParamByName('A').AsInteger;
```

See Also

- [ExecSQL](#)

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.10 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.11 GetKeyFieldNames Method

Provides a list of available key field names.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.12 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.13 GetTableNames Method

Provides a list of available tables names.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.14 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.15 PerformConnect Method

Establishes a connection to the server regardless of the LoginPrompt property.

Class

[TCustomDAConnection](#)

Syntax

```
procedure PerformConnect(Retry: boolean = False);
```

Parameters

Retry

When the Retry argument is set to False (default), the BeforeConnect event handler is called before establishing the connection.

If `Retry = True`, `BeforeConnect` is not invoked.

Remarks

Unlike the `Connect` method, the `PerformConnect` method establishes a connection regardless of the `LoginPrompt` property. When `PerformConnect` is called with `LoginPrompt = True`, the Connection Dialog is not displayed, and the `OnLogin` event handler is not invoked. After establishing the connection, the `AfterConnect` event handler is triggered.

Note: `PerformConnect` does not handle exceptions. Exceptions should be handled as usual in the calling code.

See Also

- [Connect](#)
- [LoginPrompt](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.16 Ping Method

Used to check state of connection to the server.

Class

[TCustomDACConnection](#)

Syntax

```
procedure Ping;
```

Remarks

The method is used for checking server connection state.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.17 RemoveFromPool Method

Marks the connection that should not be returned to the pool after disconnect.

Class

[TCustomDACConnection](#)

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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.18 Rollback Method

Discards all current data changes and ends transaction.

Class

[TCustomDACConnection](#)

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](#)
- [TCustomUniDataSet.SpecificOptions](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.3.19 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.4.1 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.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.4.4.2 OnError Event

This event occurs when an error has arisen in the connection.

Class

[TCustomDAConnection](#)

Syntax

```
property OnError: TDAConnectionErrorEvent;
```

Remarks

Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.1 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.
CloneCursor	Shares data from another dataset.
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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.2 Conditions Property

Used to add WHERE conditions to a query

Class

[TCustomDADataset](#)

Syntax

```
property Conditions : TDAConditions stored False;
```

See Also

- [TDAConditions](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.4 DataTypeMap Property

Used to set data type mapping rules

Class

[TCustomDADataset](#)

Syntax

```
property DataTypeMap: TDAMapRules stored IsMapRulesStored;
```

See Also

- [TDAMapRules](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.5 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 UniDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

See Also

- [TCustomDASQL.Debug](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.6 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.7 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.8 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.9 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.10 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.11 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.12 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 metadata from a server, database or dataset depending on the provider.

Note: For InterBase provider, though keys may be created across a number of table fields, sequence is generated only for the first field found in the KeyFields property.

See Also

- [SQLDelete](#)
- [SQLInsert](#)
- [SQLRefresh](#)
- [SQLUpdate](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.13 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.14 Macros Property

Makes it possible to change SQL queries easily.

Class

[TCustomDADataSet](#)

Syntax

```
property Macros: TMacro 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

```
UniQuery.SQL.Text := 'SELECT * FROM Dept ORDER BY &Order';  
UniQuery.MacroByName('Order').Value:= 'DeptNo';  
UniQuery.Open;
```

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.15 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.16 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.17 Options Property

Used to specify the behaviour of TCustomDADataset object.

Class

[TCustomDADataset](#)

Syntax

```
property options: TDADatasetOptions;
```

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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.18 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.19 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.20 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.21 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`.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.22 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.23 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.24 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](#)

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.25 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](#)

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.26 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.27 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.28 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.29 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.30 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.2.31 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

- [TCustomUniDataSet.SpecificOptions](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3 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.
CloneCursor	Shares data from another dataset.
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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.3 CloneCursor Method

Shares data from another dataset.

Class

[TCustomDADataset](#)

Syntax

```
procedure CloneCursor(Source: TMemDataSet; Reset: boolean = False;  
keepSettings: boolean = False); override;
```

Parameters

Source

Specifies the dataset whose data will be shared.

Reset

Defines how to set values for such events and properties: Filter, Filtered, FilterOptions, and OnFilterRecord; IndexName; MasterSource and MasterFields; ReadOnly; RemoteServer and ProviderName. If Reset is True, the properties are all set to their default values.

KeepSettings

Outlines how to assign values for the same events and properties as those of the Reset parameter.

Remarks

Call CloneCursor to provide a dataset with data from another dataset. After calling the method, the internal data storage is identical to both datasets.

If Reset and KeepSettings are False, the values of the properties are configured to match the source dataset. If Reset is False and KeepSettings is True, the properties remain unmodified. In this scenario, the application needs to verify that the current indexes, filters, and related settings are consistent with the cloned data.

The CloneCursor method has the following restrictions:

- * When the source dataset is closed, all cloned datasets will be closed too.
- * The cloned dataset will have the read-only status.
- * The source dataset must not be UniDirectional.
- * It's not allowed to clone a dataset from another thread as the behavior will be undefined.
- * Filter and IndexFieldNames must not be used in the cloned dataset.
- * Applying Filter and/or IndexFieldNames in the source dataset will affect the contents of the cloned datasets.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.4 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.5 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.6 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 .

© 1997-2025

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](#)

© 1997-2025

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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.7 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.8 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.12.1.5.3.9 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.10 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.11 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.12 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.13 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.14 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.15 GetDataType Method

Returns internal field types defined in the MemData and accompanying modules.

Class

[TCustomDADataset](#)

Syntax

```
function GetDataType(const FieldName: string): integer; virtual;
```

Parameters

FieldName

Holds the name of the field.

Return Value

internal field types defined in MemData and accompanying modules.

Remarks

Call the GetDataType method to return internal field types defined in the MemData and accompanying modules. Internal field data types extend the TFieldType type of VCL by specific database server data types. For example, ftString, ftFile, ftObject.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.16 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.17 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.18 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.19 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

- [TCustomDACConnection.GetTableNames](#)
- [TCustomDACConnection.GetStoredProcNames](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.20 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.21 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.22 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.23 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

```
UniQuery.SQL:= 'SELECT * FROM Scott.Dept ORDER BY &Order';  
UniQuery.MacroByName('Order').Value:= 'DeptNo';  
UniQuery.Open;
```

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.24 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.25 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.26 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.27 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.28 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](#)

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.5.3.29 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.30 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.3.31 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.4.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.4.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.4.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.4.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.5.4.5 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.1 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.
--	-----------

© 1997-2025

Devart. All Rights

Reserved.

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

6.12.1.6.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.2 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

[TCustomDASQL](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

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

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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.3 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 UniDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

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

See Also

- [TCustomDADDataSet.Debug](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.5 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.6 Macros Property

Makes it possible to change SQL queries easily.

Class

[TCustomDASQL](#)

Syntax

```
property Macros: TMacro 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.7 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.8 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.9 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 UniSQL 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';  
  end  
end
```

```
Execute;  
end;  
end;
```

See Also

- [TDAParam](#)
- [FindParam](#)
- [Macros](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.10 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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.6.2.11 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.12 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.2.13 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.1 BreakExec Method

Breaks execution of an SQL statement on the server.

Class

[TCustomDASQL](#)

Syntax

```
procedure BreakExec;
```

Remarks

Call the BreakExec method to break execution of an SQL statement on the server. It makes sense to call BreakExec only from another thread. Useful when NonBlocking is True.

See Also

- [TCustomDASQL.Execute](#)
- [TCustomDADataset.BreakExec](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.2 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 .

© 1997-2025

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.

© 1997-2025

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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.5 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.6 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.7 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

```
UniSQL.Execute;  
Edit1.Text := UniSQL.ParamsByName('Contact').AsString;
```

See Also

- [FindParam](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.8 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.9 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.3.10 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.6.4.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7 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

- [TCustomUniDataSet.UpdateObject](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2 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](#)

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.1 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.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.2 DeleteObject Property

Provides ability to perform advanced adjustment of the delete operations.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property DeleteObject: TComponent;
```

Remarks

Assign SQL component or a TCustomUniDataSet 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.4 InsertObject Property

Provides ability to perform advanced adjustment of insert operations.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property InsertObject: TComponent;
```

Remarks

Assign SQL component or TCustomUniDataSet 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.5 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.6 LockObject Property

Provides ability to perform advanced adjustment of lock operations.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property LockObject: TComponent;
```

Remarks

Assign a SQL component or TCustomUniDataSet 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.7 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.8 ModifyObject Property

Provides ability to perform advanced adjustment of modify operations.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property ModifyObject: TComponent;
```

Remarks

Assign a SQL component or TCustomUniDataSet 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.9 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.10 RefreshObject Property

Provides ability to perform advanced adjustment of refresh operations.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property RefreshObject: TComponent;
```

Remarks

Assign a SQL component or TCustomUniDataSet 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.2.11 RefreshSQL Property

Used to specify an SQL statement that will be used for refreshing the current record by [TCustomDADataset.RefreshRecord](#) procedure.

Class

[TCustomDAUpdatesQL](#)

Syntax

```
property RefreshSQL: TStrings;
```

Remarks

Use the RefreshSQL property to specify a SQL statement that will be used for refreshing the current record by the [TCustomDADataset.RefreshRecord](#) procedure.

You can assign to SQLRefresh a WHERE clause only. In such a case it is added to SELECT defined by the SQL property by [TCustomDADataset.AddWhere](#).

To create a RefreshSQL statement at design time, use the query statements editor.

See Also

- [TCustomDADataset.RefreshRecord](#)

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.7.2.12 SQL Property(Indexer)

Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Class

[TCustomDAUpdateSQL](#)

Syntax

```
property SQL[UpdateKind: TUpdateKind]: TStrings;
```

Parameters

UpdateKind

Specifies which of update SQL statements to return.

Remarks

Returns a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties, depending on the value of the UpdateKind index.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.7.3.2 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.8 TDACondition Class

Represents a condition from the [TDAConditions](#) list.

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

Unit

[DBAccess](#)

Syntax

```
TDACondition = class(TCollectionItem);
```

Remarks

Manipulate conditions using [TDAConditions](#).

See Also

- [TDAConditions](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.8.1 Members

[TDACondition](#) class overview.

Properties

Name	Description
Enabled	Indicates whether the condition is enabled or not
Name	The name of the condition
Value	The value of the condition

Methods

Name	Description
Disable	Disables the condition
Enable	Enables the condition

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.8.2 Properties

Properties of the **TDACondition** class.

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

Published

Name	Description
Enabled	Indicates whether the condition is enabled or not
Name	The name of the condition
Value	The value of the condition

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.8.2.1 Enabled Property

Indicates whether the condition is enabled or not

Class

[TDACondition](#)

Syntax

```
property Enabled: Boolean default True;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.8.2.2 Name Property

The name of the condition

Class

[TDACondition](#)

Syntax

```
property Name: string;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.8.2.3 Value Property

The value of the condition

Class

[TDACondition](#)

Syntax

```
property value: string;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.8.3 Methods

Methods of the **TDACondition** class.

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

Public

Name	Description
Disable	Disables the condition
Enable	Enables the condition

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.8.3.1 Disable Method

Disables the condition

Class

[TDACondition](#)

Syntax

```
procedure Disable;
```

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.8.3.2 Enable Method

Enables the condition

Class

[TDACondition](#)

Syntax

```
procedure Enable;
```

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9 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)
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.1 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 TDACCondition (the condition) by its name. If found, the TDACCondition object is returned, otherwise - nil.
Get	Retrieving a TDACCondition object by its name. If found, the TDACCondition object is returned, otherwise - an exception is raised.
IndexOf	Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.
Remove	Removes the condition

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.2 Properties

Properties of the **TDACConditions** class.

For a complete list of the **TDACConditions** class members, see the [TDACConditions 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=COND

	ITION
WhereSQL	Returns the SQL WHERE condition added in the Conditions property.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.2.1 Condition Property(Indexer)

Used to iterate through all the conditions.

Class

[TDAConditions](#)

Syntax

```
property Condition[Index: Integer]: TDACondition;
```

Parameters

Index

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.2.2 Enabled Property

Indicates whether the condition is enabled

Class

[TDAConditions](#)

Syntax

```
property Enabled: Boolean;
```

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.9.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.2.4 Text Property

The property returns condition names and values as CONDITION_NAME=CONDITION

Class

[TDAConditions](#)

Syntax

```
property Text: string;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.2.5 WhereSQL Property

Returns the SQL WHERE condition added in the Conditions property.

Class

[TDAConditions](#)

Syntax

```
property whereSQL: string;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.3.1 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.

© 1997-2025

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)
```

© 1997-2025

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)
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.3.2 Delete Method

Deletes the condition

Class

[TDAConditions](#)

Syntax

```
procedure Delete(Index: integer);
```

Parameters

Index

Index of the condition

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.3.3 Disable Method

Disables the condition

Class

[TDAConditions](#)

Syntax

```
procedure Disable;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.3.4 Enable Method

Enables the condition

Class

[TDAConditions](#)

Syntax

```
procedure Enable;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.3.5 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.3.6 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

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.3.7 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

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.9.3.8 Remove Method

Removes the condition

Class

[TDAConditions](#)

Syntax

```
procedure Remove(const Name: string);
```

Parameters

Name

Specifies the name of the removed condition

© 1997-2025

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.12.1.10 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);
```

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.10.1 Members

[TDACConnectionOptions](#) class overview.

Properties

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.
DefaultSortType	Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode	Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.
KeepDesignConnected	Prevents 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 the connection is lost.
-------------------------------	---

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.10.2 Properties

Properties of the **TDACconnectionOptions** class.

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

Public

Name	Description
DefaultSortType	Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode	Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.
KeepDesignConnected	Prevents 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 the connection is lost.

Published

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.10.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.10.2.2 DefaultSortType Property

Specifies the default type of local sorting for string fields. It is used when a sort type is not explicitly set after the field name in the [TMemDataSet.IndexFieldNames](#) property of a dataset.

Class

[TDACConnectionOptions](#)

Syntax

```
property DefaultSortType: TSortType default stCaseSensitive;
```

Remarks

Use the DefaultSortType property to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the [TMemDataSet.IndexFieldNames](#) property of a dataset.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.10.2.3 DisconnectedMode Property

Opens a connection only when needed to perform a server call, and closes that connection after the operation is complete.

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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.10.2.4 KeepDesignConnected Property

Prevents 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.10.2.5 LocalFailover Property

If True, the [TCustomDACConnection.OnConnectionLost](#) event occurs, and a failover operation can be performed after the connection is lost.

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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.11 TDAConnectionSSLOptions Class

This class is used to set up the SSL options.

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

Unit

[DBAccess](#)

Syntax

```
TDAConnectionSSLOptions = class(TPersistent);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.11.1 Members

[TDAConnectionSSLOptions](#) 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.11.2 Properties

Properties of the **TDAConnectionSSLOptions** class.

For a complete list of the **TDAConnectionSSLOptions** class members, see the

[TDAConnectionSSLOptions 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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.11.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.11.2.2 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.11.2.3 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.11.2.4 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12 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);
```

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.1 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.
---------------------------------	---

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.5 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.7 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.8 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.9 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.10 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.11 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;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.12 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.13 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.14 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.15 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.16 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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.12.2.17 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.18 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.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.19 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.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.20 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.21 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.12.2.22 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.13 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.13.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.12.1.13.2 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.13.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.13.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14 TDAMapRule Class

Class that formes 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.1 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.
------------------------------	--

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.5 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.6 FieldLength Property

The resultant field length in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldLength: Integer default r1Any;
```

Remarks

Setting the Delphi field length after conversion.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.7 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.8 FieldScale Property

The resultant field Scale in Delphi.

Class

[TDAMapRule](#)

Syntax

```
property FieldScale: Integer default r1Any;
```

Remarks

Setting the Delphi field Scale after conversion.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.9 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.14.2.10 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.15 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.15.1 Members

[TDAMapRules](#) class overview.

Properties

Name	Description
IgnoreInvalidRules	Used to avoid raising exception on mapping rules that can't be applied.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.15.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.12.1.15.2.1 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.16 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.16.1 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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.16.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.16.2.1 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

[TDAMetaData](#)

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object to use to connect to a data store.

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

At runtime, set the Connection property to reference an instantiated TCustomDAConnection

object.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.16.2.2 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 MetadataKind property is an empty string
ProcedureParameters	show metainformation about parameters of existing procedures
Procedures	show metainformation about existing procedures
Restrictions	generates a dataset that describes which restrictions are applicable to each MetaDataKind
Tables	show metainformation about existing tables
Databases	show metainformation about existing databases

If you provide a value that equals neither of the values described in the table, an error will be raised.

See Also

- [Restrictions](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.16.2.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.16.3 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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.16.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.16.3.2 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](#)

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17 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

- [TCustomDADDataSet](#)
- [TCustomDASQL](#)
- [TDAParams](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.5 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.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.6 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.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.7 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.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.8 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.9 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.10 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.11 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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.17.2.12 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.13 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.14 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.2.15 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.3.1 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.3.2 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.3.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.3.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.17.3.5 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.

© 1997-2025

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.

© 1997-2025

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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.18 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.18.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.18.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.18.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.18.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.18.3.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.18.3.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.2.2 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.3 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](#)

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.3.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.4 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 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.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.4.1 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 [TUniTransaction.CommitRetaining](#) method execution, the [OnCommitRetaining](#) event is used. When an error occurs during commit, the [OnError](#) event fires.

See Also

- [Commit](#)
- [TUniTransaction.CommitRetaining](#)
- [OnCommitRetaining](#)
- [OnError](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.4.2 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

- [TUniTransaction.CommitRetaining](#)

- [Commit](#)
- [OnCommit](#)
- [OnError](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.4.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.4.4 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 [TUniTransaction.RollbackRetaining](#) method execution, the [OnRollbackRetaining](#) event is used.

When an error occurs during rollback, the [OnError](#) event fired.

See Also

- [Rollback](#)
- [TUniTransaction.RollbackRetaining](#)
- [OnRollbackRetaining](#)
- [OnError](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.19.4.5 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](#)
- [TUniTransaction.RollbackRetaining](#)
- [OnRollback](#)
- [OnError](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.20 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.20.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.20.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.20.2.1 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

```
UniQuery.SQL.Text := 'SELECT * FROM Dept WHERE DeptNo > 20 &Cond1';  
UniQuery.Macros[0].Value := 'and DName is NULL';  
UniQuery.Macros[0].Active:= False;
```

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.20.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.20.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.20.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.20.2.5 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.20.2.6 Name Property

Used to identify a particular macro.

Class

[TMacro](#)

Syntax

```
property Name: string;
```

Remarks

Use the Name property to identify a particular macro.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.20.2.7 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.21 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.21.1 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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.21.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.21.2.1 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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.1.21.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.21.3.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.21.3.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.21.3.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.21.3.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.12.1.21.3.5 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.21.3.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.22 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);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.22.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.22.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.22.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.22.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.22.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.22.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.22.2.5 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.23 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);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.23.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.23.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.23.2.1 Enabled Property

Sets SmartFetch mode enabled or not.

Class

[TSmartFetchOptions](#)

Syntax

```
property Enabled: Boolean default False;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.23.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.23.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.1.23.2.4 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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.12.2 Types

Types in the **DBAccess** unit.

Types

Name	Description
TAfterExecuteEvent	This type is used for the TCustomDADataset.AfterExecute and TCustomDASQL.AfterExecute events.
TAfterFetchEvent	This type is used for the TCustomDADataset.AfterFetch event.
TBeforeFetchEvent	This type is used for the TCustomDADataset.BeforeFetch event.
TConnectionLostEvent	This type is used for the TCustomDACConnection.OnConnectionLost event.
TDAConnectionErrorEvent	This type is used for the TCustomDACConnection.OnError event.
TDATransactionErrorEvent	This type is used for the TDATransaction.OnError event.
TRefreshOptions	Represents the set of TRefreshOption .
TUpdateExecuteEvent	This type is used for the TCustomDADataset.AfterUpdateExecute and TCustomDADataset.BeforeUpdateExecute events.

© 1997-2025
Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.2.1 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.2.2 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.2.3 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.

© 1997-2025

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.12.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.2.5 TDAConnectionErrorEvent Procedure Reference

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

Unit

[DBAccess](#)

Syntax

```
TDAConnectionErrorEvent = procedure (Sender: TObject; E: EDAError;  
var Fail: boolean) of object;
```

Parameters

Sender

An object that raised the event.

E

The error information.

Fail

False, if an error dialog should be prevented from being displayed and EAbort exception should be raised to cancel current operation .

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.2.6 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.2.7 TRefreshOptions Set

Represents the set of [TRefreshOption](#).

Unit

[DBAccess](#)

Syntax

```
TRefreshOptions = set of TRefreshOption;
```

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.2.8 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.3 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.3.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.3.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.3.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.3.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.4 Variables

Variables in the **DBAccess** unit.

Variables

Name	Description
ChangeCursor	When set to True allows data access components to change screen cursor for the execution time.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.12.4.1 ChangeCursor Variable

When set to True allows data access components to change screen cursor for the execution time.

Unit

[DBAccess](#)

Syntax

```
changeCursor: boolean = True;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.13 LiteCollation

This unit contains types for registering user-defined collations.

Types

Name	Description
TLiteAnsiCollation	This type is used for registering a user-defined non-Unicode collation.
TLiteCollation	This type is used for registering a user-defined collation.
TLiteWideCollation	This type is used for registering a user-defined Unicode collation.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.13.1 Types

Types in the **LiteCollation** unit.

Types

Name	Description
TLiteAnsiCollation	This type is used for registering a user-defined non-Unicode collation.
TLiteCollation	This type is used for registering a user-defined collation.
TLiteWideCollation	This type is used for registering a user-defined Unicode collation.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.13.1.1 TLiteAnsiCollation Function Reference

This type is used for registering a user-defined non-Unicode collation.

Unit

[LiteCollation](#)

Syntax

```
TLiteAnsiCollation = function (const Str1: AnsiString; const
Str2: AnsiString): Integer;
```

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.13.1.2 TLiteCollation Function Reference

This type is used for registering a user-defined collation.

Unit

[LiteCollation](#)

Syntax

```
TLiteCollation = function (const Str1: string; const Str2: string): Integer;
```

Remarks

Collation parameter data types depend on Delphi version.

Delphi version	Parameter data type	Description
Delphi 2007 and lower	String = AnsiString	non-Unicode collation
Delphi 2009 and higher	String = WideString	Unicode collation

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.13.1.3 TLiteWideCollation Function Reference

This type is used for registering a user-defined Unicode collation.

Unit

[LiteCollation](#)

Syntax

```
TLitewideCollation = function (const Str1: string; const Str2: string): Integer;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.14 LiteFunction

This unit contains types for registering user-defined functions.

Types

Name	Description
TLiteFunction	This type is used for the registering a user-defined

function.

© 1997-2025

Devart. All Rights

Reserved.

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

6.14.1 Types

Types in the **LiteFunction** unit.

Types

Name	Description
TLiteFunction	This type is used for the registering a user-defined function.

© 1997-2025

Devart. All Rights

Reserved.

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

6.14.1.1 TLiteFunction Function Reference

This type is used for the registering a user-defined function.

Unit

[LiteFunction](#)

Syntax

```
TLiteFunction = function (InValues: array of Variant): Variant;
```

Remarks

If the UseUnicode connection specific option is true then input string parameters will be represented as WideString else input string parameters will be represented as AnsiString.

© 1997-2025

Devart. All Rights

Reserved.

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

6.15 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.

© 1997-2025

Devart. All Rights Reserved.

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

6.15.1 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.

© 1997-2025

Devart. All Rights

Reserved.

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

6.15.1.1 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](#)

© 1997-2025

Devart. All Rights

Reserved.

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

6.15.1.1.1 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.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.2 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](#)

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.2.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.2.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.2.4 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.3.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.3.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.3.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.3.5 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](#)

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.15.1.1.3.6 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.3.7 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.3.8 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.1.3.9 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.2.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.2.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.2.2.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.3 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.3.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.4 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.4.1 Members

[TMemData](#) class overview.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.5 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**

© 1997-2025

Devart. All Rights Reserved.

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

6.15.1.5.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.5.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.5.2.1 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.

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.5.2.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.5.2.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.5.2.4 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.15.1.5.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.5.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.6 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.6.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.6.2 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.6.2.1 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.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.6.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.6.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.1.6.3.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.2 Types

Types in the **MemData** unit.

Types

Name	Description
TLocateExOptions	Represents the set of TLocateExOption .
TUpdateRecKinds	Represents the set of TUpdateRecKind.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.2.1 TLocateExOptions Set

Represents the set of [TLocateExOption](#).

Unit

[MemData](#)

Syntax

```
TLocateExOptions = set of TLocateExOption;
```

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.15.2.2 TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

[MemData](#)

Syntax

```
TUpdateRecKinds = set of TUpdateRecKind;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.3.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.3.2 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.3.3 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)[DAC Forum](#)[Provide Feedback](#)**6.15.3.4 TLocateExOption Enumeration**

Allows to set additional search parameters which will be used by the LocateEx method.

Unit

[MemData](#)

Syntax

```
TLocateExOption = (lxCaseInsensitive, lxPartialKey, lxNearest, lxNext, lxUp, lxPartialCompare);
```

Values

Value	Meaning
lxCaseInsensitive	Similar to loCaseInsensitive. Key fields and key values are matched without regard to the case.
lxNearest	LocateEx moves the cursor to a specific record in a dataset or to the first record in the dataset that is greater than the values specified in the KeyValues parameter. For this option to work correctly dataset should be sorted by the fields the search is performed in. If dataset is not sorted, the function may return a

	line that is not connected with the search condition.
lxNext	LocateEx searches from the current record.
lxPartialCompare	Similar to lxPartialKey, but the difference is that it can process value entries in any position. For example, 'HAM' would match both 'HAMM', 'HAMMER.', and also 'MR HAMMER'.
lxPartialKey	Similar to loPartialKey. Key values can include only a part of the matching key field value. For example, 'HAM' would match both 'HAMM' and 'HAMMER.', but not 'MR HAMMER'.
lxUp	LocateEx searches from the current record to the first record.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.3.5 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.15.3.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16 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.

Types

Name	Description
TJsonOptions	Additional options used when saving the dataset in the JSON format.

Enumerations

Name	Description
TJsonOption	Additional options used when saving the dataset to the JSON format.
TXmlIndentFormat	Specifies the indentation format in XML format files or streams when saving the

	dataset using the TMemDataSet.SaveToXML methods.
--	--

© 1997-2025

Devart. All Rights Reserved.

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

6.16.1 Classes

Classes in the **MemDS** unit.

Classes

Name	Description
TMemDataSet	A base class for working with data and manipulating data in memory.

© 1997-2025

Devart. All Rights Reserved.

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

6.16.1.1 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.

© 1997-2025

Devart. All Rights Reserved.

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

6.16.1.1.1 Members

[TMemDataSet](#) class overview.

Properties

Name	Description
CachedUpdates	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive	Specifies the upper and lower boundaries for a range.
LocalConstraints	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate	Used to prevent implicit update of rows on database server.
Prepared	Determines whether a query is prepared for execution or not.
Ranged	Indicates whether a range is applied to a dataset.
UpdateRecordTypes	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange	Applies a range to the dataset.
ApplyUpdates	Overloaded. Writes dataset's pending cached updates to a database.

CancelRange	Removes any ranges currently in effect for a dataset.
CancelUpdates	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates	Clears the cached updates buffer.
DeferredPost	Makes permanent changes to the database server.
EditRangeEnd	Enables changing the ending value for an existing range.
EditRangeStart	Enables changing the starting value for an existing range.
GetBlob	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
Locate	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare	Allocates resources and creates field components for a dataset.
RestoreUpdates	Marks all records in the cache of updates as unapplied.
RevertRecord	Cancels changes made to the current record when cached updates are enabled.
SaveToJSON	Overloaded. Saves the current dataset data to a file or a stream in the JSON

	format.
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.

Devart. All Rights Reserved.

6.16.1.1.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.2.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.2.2 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 [TCustomDACConnection](#) component, the default value of the sort type depends on the [TCustomDACConnection.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';
```

© 1997-2025

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.16.1.1.2.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.2.4 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`.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.2.5 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.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.2.6 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](#)

© 1997-2025
Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.16.1.1.2.7 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](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.2.8 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.2.9 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3 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.
SaveToJSON	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.2 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.

© 1997-2025

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.

Pay attention that running this method with Interbase and Firebird results in the following

error: *UniQuery1: Cannot perform this operation on a closed dataset* . It occurs because InterBase and Firebird require an active transaction for any data operation. When you open a dataset without a transaction, UniDAC starts a transaction internally and allows you to read and write data. However, if you explicitly start a transaction and then either roll it back or commit it, the dataset will close (as there will be no active transaction). To avoid the error, you can use one of the following approaches:

1. Call the `CommitRetaining` method instead of `Commit` (or call the `RollbackRetaining` method for the except block instead of `Rollback`). permanently saves all changes associated with the transaction to the database server while retaining the transaction context. You can read more about `CommitRetaining` and `RollbackRetaining` in [TUniConnection.CommitRetaining](#) and [TUniConnection.RollbackRetaining](#).

2. Use different transactions for connections and datasets. Additionally, use separate transactions for reading and modifying data. You can find a code example below.

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 first part of the code illustrates how to apply a dataset's cached updates to a database with InterBase and Firebird in response to a button click, while the second part shows how to do the same without using InterBase and Firebird:

```
procedure ApplyButtonClick(Sender: TObject);
begin
  with Query1 do begin
    Connection1.StartTransaction;
    try
      // ... modify data
      ApplyUpdates;           // try to write the updates to the database
      Connection1.Commit;    // on success, commit the changes
    except
      RestoreUpdates;       // restore update result for applied records
      Connection1.Rollback; // on failure, undo the changes
      raise;                // raise the exception to prevent a call to Commit
    end;
    CommitUpdates;         //on success, clear the cache
  end;
end;
// use the different approach when working with InterBase/Firebird
// open the connection
```

```
...
UniConnection1.AutoCommit := false;
UniConnection1.Open;
...
// open the query
...
with UniQuery1 do begin
    CachedUpdates := true;
    Transaction := UniTransaction1;
    UpdateTransaction := UniTransaction2; // use different transaction for u
    Open;
end;
...
procedure ApplyButtonClick(Sender: TObject);
begin
    with UniQuery1 do begin
        if not UpdateTransaction.Active then
            UpdateTransaction.StartTransaction;
        try
            // modify data
            ApplyUpdates;
            UpdateTransaction.CommitRetaining;
        except
            RestoreUpdates;
            UpdateTransaction.RollbackRetaining;
            raise;
        end;
        CommitUpdates;
    end;
end;
```

See Also

- [TMemDataSet.CachedUpdates](#)
- [TMemDataSet.CancelUpdates](#)
- [TMemDataSet.CommitUpdates](#)
- [TMemDataSet.UpdateStatus](#)

© 1997-2025

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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.3 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.5 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.6 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.7 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.8 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.9 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.

© 1997-2025

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.

© 1997-2025

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

```
UniQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
```

See Also

- [TBlob](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.10 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.

© 1997-2025

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.

© 1997-2025

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']), [!PartialKey]);
```

See Also

- [TMemDataSet.IndexFieldNames](#)
- [TMemDataSet.LocateEx](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.11 LocateEx Method

Excludes features that don't need to be included to the [TMemDataSet.Locate](#) method of TDataSet.

Class

[TMemDataSet](#)

Overload List

Name	Description
LocateEx(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateExOptions)	Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.
LocateEx(const KeyFields: string; const KeyValues: variant; Options: TLocateExOptions)	Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified field names.

© 1997-2025

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.

© 1997-2025

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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.12 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.13 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.14 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.15 SaveToJSON Method

Saves the current dataset data to a file or a stream in the JSON format.

Class

[TMemDataSet](#)

Overload List

Name	Description
SaveToJSON(Destination: TStream; Options: TJsonOptions)	Saves the current dataset data to a stream in the JSON format.
SaveToJSON(const FileName: string; Options: TJsonOptions)	Saves the current dataset data to a file in the JSON format.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Saves the current dataset data to a stream in the JSON format.

Class

[TMemDataSet](#)

Syntax

```
procedure SaveToJSON(Destination: TStream; Options: TJsonOptions = []); overload;
```

Parameters

Destination

Holds a TStream object.

Options

Holds additional options used when saving the dataset. See [TJsonOptions](#) for details. By default, no additional options are set.

Remarks

Call the SaveToJSON method to save the current dataset data to a stream in the JSON format. A TStream object must be created and its position must be set in a preferable value.

See Also

- [TJsonOptions](#)
- [TVirtualTable.LoadFromFile](#)
- [TVirtualTable.LoadFromStream](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Saves the current dataset data to a file in the JSON format.

Class

[TMemDataSet](#)

Syntax

```
procedure SaveToJSON(const FileName: string; Options: TJsonOptions = []); overload;
```

Parameters

FileName

Holds the name of a destination file.

Options

Holds additional options used when saving the dataset. See [TJsonOptions](#) for details. By default, no additional options are set.

Remarks

Call the SaveToJson method to save the current dataset data to a file in the JSON format.

If the destination file already exists, it is overwritten. It remains open from the first call to SaveToJson until the dataset is closed. Other applications can read this file while it is open, but they cannot write to it.

See Also

- [TJsonOptions](#)
- [TVirtualTable.LoadFromFile](#)
- [TVirtualTable.LoadFromStream](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.16 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; IndentFormat: TXmlIndentFormat)	Saves the current dataset data to a stream in the XML format compatible with ADO format.
SaveToXML(const FileName: string; IndentFormat: TXmlIndentFormat)	Saves the current dataset data to a file in the XML format compatible with ADO format.

© 1997-2025

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; IndentFormat: TXmlIndentFormat = ifSpace); overload;
```

Parameters

Destination

Holds a TStream object.

IndentFormat

Specifies how to format indentation in XML format streams when saving the dataset. The default value is ifSpace. For other values, see the [TXmlIndentFormat](#) class.

Remarks

Call the SaveToXML method to save the current dataset data to a file or stream in XML format, compatible with ADO.

- If the destination file already exists, it will be overwritten.
- The destination stream remains open from the first call to SaveToXML until the dataset is closed. The file can be read by other applications while open, but it cannot be written to during that time.

When saving to a stream, ensure that a TStream object is created, and its position is set to the desired value.

See Also

- [TVirtualTable.LoadFromFile](#)
- [TVirtualTable.LoadFromStream](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Saves the current dataset data to a file in the XML format compatible with ADO format.

Class

[TMemDataSet](#)

Syntax

```
procedure SaveToXML(const FileName: string; IndentFormat: TXmlIndentFormat = ifSpace); overload;
```

Parameters

FileName

Holds the name of a destination file.

IndentFormat

Specifies how to format indentation in XML format files when saving the dataset. The default value is ifSpace. For other values, see the [TXmlIndentFormat](#) class.

See Also

- [TVirtualTable.LoadFromFile](#)
- [TVirtualTable.LoadFromStream](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.17 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](#)

Reserved.

6.16.1.1.3.18 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.19 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.20 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.21 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](#)

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.3.22 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](#)

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.4 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.4.1 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.1.1.4.2 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.2 Types

Types in the **MemDS** unit.

Types

Name	Description
TJsonOptions	Additional options used when saving the dataset in the JSON format.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.2.1 TJsonOptions Set

Additional options used when saving the dataset in the JSON format.

Unit

[MemDS](#)

Syntax

```
TJsonOptions = set of TJsonOption;
```

See Also

- [TMemDataSet.SaveToJSON](#)
- [TMemDataSet.SaveToJSON](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.3 Enumerations

Enumerations in the **MemDS** unit.

Enumerations

Name	Description
TJsonOption	Additional options used when saving the dataset to the JSON format.
TXmlIndentFormat	Specifies the indentation format in XML format files or streams when saving the dataset using the TMemDataSet.SaveToXML methods.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.3.1 TJsonOption Enumeration

Additional options used when saving the dataset to the JSON format.

Unit

[MemDS](#)

Syntax

```
TJsonOption = (joDataOnly, joDataAsArray, joIncludeNulls);
```

Values

Value	Meaning
joDataAsArray	When the option is set, each row of the dataset is saved as a JSON array. Otherwise (by default), each row of the dataset is saved as a JSON object.
joDataOnly	When the option is set, only the dataset rows are saved to JSON. By default, information about the dataset structure is also saved to JSON.
joIncludeNulls	When the option is set, fields with NULL values are saved to JSON. Otherwise (by default), NULL fields are excluded from JSON.

See Also

- [TJsonOptions](#)
- [TMemDataSet.SaveToJSON](#)
- [TMemDataSet.SaveToJSON](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.16.3.2 TXmlIndentFormat Enumeration

Specifies the indentation format in XML format files or streams when saving the dataset using the [TMemDataSet.SaveToXML](#) methods.

Unit

[MemDS](#)

Syntax

```
TXmlIndentFormat = (ifSpace, ifTabulation, ifNone);
```

Values

Value	Meaning
ifNone	No indentation is used.
ifSpace	Space characters are used for indentation.
ifTabulation	Tabs are used for indentation.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.17 OracleUniProvider

This unit contains the TOraUtils class that allows you to use features of Oracle database.

Classes

Name	Description
TOraUtils	This class class is used for implementation of specific

	Oracle operations, such as changing a user's password.
--	--

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.17.1 Classes

Classes in the **OracleUniProvider** unit.

Classes

Name	Description
TOraUtils	This class class is used for implementation of specific Oracle operations, such as changing a user's password.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.17.1.1 TOraUtils Class

This class class is used for implementation of specific Oracle operations, such as changing a user's password.

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

Unit

[oracleUniProvider](#)

Syntax

```
ToraUtils = class(System.TObject);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.17.1.1.1 Members

[TOraUtils](#) class overview.

Methods

Name	Description
ChangePassword	Changes the password of an account to the new password.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.17.1.1.2 Methods

Methods of the **TOrUtils** class.

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

Public

Name	Description
ChangePassword	Changes the password of an account to the new password.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.17.1.1.2.1 ChangePassword Method

Changes the password of an account to the new password.

Class

[TOrUtils](#)

Syntax

```
class procedure ChangePassword(Connection: TCustomDACConnection;
```

```
NewPassword: string);
```

Parameters

Connection

NewPassword

Takes the new password.

Remarks

Call the ChangePassword method to replace the current password of an account with the new password.

The previous values must be provided for the Password and UserName properties before calling ChangePassword.

The ChangePassword method is used mainly when logging in to the user account fails due to an expired password or any other reason accompanied by an exception with ORA-2800 Oracle error code family (see Oracle Error Messages).

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18 SQLiteUniProvider

This unit contains the TLiteUtils class that allows you to use features of SQLite database.

Classes

Name	Description
TLiteUtils	This class class is used for implementation of specific SQLite operations, such as database encryption or collation management.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1 Classes

Classes in the **SQLiteUniProvider** unit.

Classes

Name	Description
T LiteUtils	This class class is used for implementation of specific SQLite operations, such as database encryption or collation management.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1 T LiteUtils Class

This class class is used for implementation of specific SQLite operations, such as database encryption or collation management.

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

Unit

[SQLiteUniProvider](#)

Syntax

```
T LiteUtils = class(System.TObject);
```

Remarks

Class that implements SQLite specific methods such as EncryptDatabase, RegisterCollation, UnRegisterCollation.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1.1 Members

[T LiteUtils](#) class overview.

Methods

Name	Description
EncryptDatabase	Used for setting a new password or changing an existing password.
RegisterAnsiCollation	This method is used for registering a user-defined non-Unicode collation.
RegisterCollation	This method is used for registering a user-defined String collation.
RegisterFunction	This method is used for registering a user-defined function.
RegisterWideCollation	This method is used for registering a user-defined Unicode collation.
UnRegisterAnsiCollation	This method is used for unregistering a user-defined non-Unicode collation.
UnRegisterCollation	This method is used for unregistering user-defined collation.
UnRegisterFunction	This method is used for unregistering a user-defined function.
UnRegisterWideCollation	This method is used for unregistering a user-defined Unicode collation.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1.2 Methods

Methods of the **TLiteUtils** class.

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

Public

Name	Description
EncryptDatabase	Used for setting a new password or changing an existing password.

RegisterAnsiCollation	This method is used for registering a user-defined non-Unicode collation.
RegisterCollation	This method is used for registering a user-defined String collation.
RegisterFunction	This method is used for registering a user-defined function.
RegisterWideCollation	This method is used for registering a user-defined Unicode collation.
UnRegisterAnsiCollation	This method is used for unregistering a user-defined non-Unicode collation.
UnRegisterCollation	This method is used for unregistering user-defined collation.
UnRegisterFunction	This method is used for unregistering a user-defined function.
UnRegisterWideCollation	This method is used for unregistering a user-defined Unicode collation.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1.2.1 EncryptDatabase Method

Used for setting a new password or changing an existing password.

Class

[TLiteUtils](#)

Syntax

```
class procedure EncryptDatabase(Connection: TCustomDAConnection;
```

```
NewKey: string);
```

Parameters

Connection

A pointer for TCustomDACConnection.

NewKey

A new password value.

Remarks

The database connection should be established before using this method. EncryptionKey value should be set when database is already encrypted. Encoding function will be supported by SQLite library.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1.2.2 RegisterAnsiCollation Method

This method is used for registering a user-defined non-Unicode collation.

Class

[TLiteUtils](#)

Syntax

```
class procedure RegisterAnsiCollation(Connection:  
TCustomDACConnection; Name: string; LiteAnsiCollation:  
TLiteAnsiCollation);
```

Parameters

Connection

Connection where user-defined collation should be registered.

Name

User-defined collation name.

LiteAnsiCollation

User-defined non-Unicode collation.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1.2.3 RegisterCollation Method

This method is used for registering a user-defined String collation.

Class

[TLiteUtils](#)

Syntax

```
class procedure RegisterCollation(Connection: TCustomDAConnection; Name: string; LiteCollation: TLiteCollation);
```

Parameters*Connection*

Connection with database where user-defined collation should be registered.

Name

User-defined collation name.

LiteCollation

User-defined collation.

Remarks

TLiteCollation has String parameters that depend on Delphi version:

Delphi version	Parameter data type	Description
Delphi 2007 and lower	String = AnsiString	non-Unicode collation
Delphi 2009 and higher	String = WideString	Unicode collation

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1.2.4 RegisterFunction Method

This method is used for registering a user-defined function.

Class

[TLiteUtils](#)

Syntax

```
class procedure RegisterFunction(Connection: TCustomDAConnection;  
const Name: string; ParamCount: Integer; LiteFunction:
```

```
TLiteFunction);
```

Parameters

Connection

Connection where user-defined function should be registered.

Name

User-defined function name.

ParamCount

The number of the input parameters for user-defined function.

LiteFunction

User-defined function to register.

Remarks

If UseUnicode connection specific option is true then input string parameters will be represented as WideString else input string parameters will be represented as AnsiString.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1.2.5 RegisterWideCollation Method

This method is used for registering a user-defined Unicode collation.

Class

[TLiteUtils](#)

Syntax

```
class procedure RegisterWideCollation(Connection:  
TCustomDACConnection; Name: string; LitewideCollation:  
TLitewideCollation);
```

Parameters

Connection

Connection where user-defined collation should be registered.

Name

User-defined collation name.

LiteWideCollation

User-defined Unicode collation.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.18.1.1.2.6 UnRegisterAnsiCollation Method

This method is used for unregistering a user-defined non-Unicode collation.

Class

[TLiteUtils](#)

Syntax

```
class procedure UnRegisterAnsiCollation(Connection:  
TCustomDAConnection; Name: string);
```

Parameters

Connection

Connection where user-defined collation should be unregistered.

Name

User-defined collation name.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1.2.7 UnRegisterCollation Method

This method is used for unregistering user-defined collation.

Class

[TLiteUtils](#)

Syntax

```
class procedure UnRegisterCollation(Connection:  
TCustomDAConnection; Name: string);
```

Parameters

Connection

Connection with database where user-defined collation should be unregistered.

Name

User-defined collation name.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.18.1.1.2.8 UnRegisterFunction Method

This method is used for unregistering a user-defined function.

Class

[TLiteUtils](#)

Syntax

```
class procedure UnRegisterFunction(Connection:  
TCustomDACConnection; Name: string; ParamCount: Integer);
```

Parameters

Connection

Connection where the user-defined function should be unregistered.

Name

User-defined function name.

ParamCount

The number of the input parameters for User-defined function.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.18.1.1.2.9 UnRegisterWideCollation Method

This method is used for unregistering a user-defined Unicode collation.

Class

[TLiteUtils](#)

Syntax

```
class procedure UnRegisterwideCollation(Connection:  
TCustomDACConnection; Name: string);
```

Parameters

Connection

Connection where the user-defined collation should be unregistered

Name

User-defined collation name.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.19 SQLServerUniProvider

This unit contains the `TMSSqlUtils` class that allows you to use features of SQL Server database.

Classes

Name	Description
TMSSqlUtils	This class class is used for implementation of specific SQL Server operations, such as changing a user's password.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.19.1 Classes

Classes in the `SQLServerUniProvider` unit.

Classes

Name	Description
TMSSqlUtils	This class class is used for implementation of specific SQL Server operations, such as changing a user's password.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.19.1.1 TMSSqlUtils Class

This class class is used for implementation of specific SQL Server operations, such as changing a user's password.

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

Unit

[SQLServerUniProvider](#)

Syntax

```
TMSSqlUtils = class(System.Object);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.19.1.1.1 Members

[TMSSqlUtils](#) class overview.

Methods

Name	Description
ChangePassword	Changes the password of an account to the new password.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.19.1.1.2 Methods

Methods of the **TMSSqlUtils** class.

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

Public

Name	Description
ChangePassword	Changes the password of an account to the new password.

See Also

- [TMSSqlUtils Class](#)

- [TMSSqlUtils Class Members](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.19.1.1.2.1 ChangePassword Method

Changes the password of an account to the new password.

Class

[TMSSqlUtils](#)

Syntax

```
class procedure ChangePassword(Connection: TCustomDACConnection;
NewPassword: string);
```

Parameters

Connection

NewPassword

Takes the new password.

Remarks

Call the ChangePassword method to replace an expired user's password with the new password. In SQL Server versions prior to SQL Server 2005, only the database administrator has permissions to change an expired user's password. Starting with SQL Server 2005, you can change the password using the ChangePassword method and SQL Native Client.

Note: Only an expired password can be changed using this method.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20 Uni

This unit contains main components of UniDAC.

Classes

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

TCustomUniDataSet	A base component for defining functionality for classes derived from it.
TCustomUniTable	A base class for retrieving and updating data in a single table without writing SQL statements.
TUniBlob	A class holding value of the BLOB fields and parameters.
TUniConnection	A component for setting up and controlling connection to such database servers as Oracle, SQL Server, MySQL, InterBase, Firebird, and PostgreSQL.
TUniDataSetOptions	Specifies the behaviour of a TCustomUniDataSet object.
TUniDataSource	TUniDataSource provides an interface between a UniDAC dataset components and data-aware controls on a form.
TUniEncryptor	The class that performs encrypting and decrypting of data.
TUniMacro	Holds the Name, Value, and Condition for a macro.
TUniMacros	Used to manage a list of TUniMacro objects for a TUniConnection component.
TUniMetaData	A component for obtaining metainformation about database objects from the server.
TUniParam	A class that is used to set the values of individual parameters passed with queries or stored procedures.
TUniParams	Used to control TUniParam objects.
TUniQuery	A component for executing queries and operating

	record sets. It also provides flexible way to update data.
TUniSQL	A component for executing SQL statements and calling stored procedures on the database server.
TUniStoredProc	A component for accessing and executing stored procedures and functions.
TUniTable	A component for retrieving and updating data in a single table without writing SQL statements.
TUniTransaction	A component for managing transactions in an application.
TUniUpdateSQL	A component for tuning update operations for the DataSet component.

Constants

Name	Description
UniDACVersion	Read this constant to get current version number for UniDAC.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1 Classes

Classes in the **Uni** unit.

Classes

Name	Description
TCustomUniDataSet	A base component for defining functionality for classes derived from it.
TCustomUniTable	A base class for retrieving and updating data in a single table without writing

	SQL statements.
TUniBlob	A class holding value of the BLOB fields and parameters.
TUniConnection	A component for setting up and controlling connection to such database servers as Oracle, SQL Server, MySQL, InterBase, Firebird, and PostgreSQL.
TUniDataSetOptions	Specifies the behaviour of a TCustomUniDataSet object.
TUniDataSource	TUniDataSource provides an interface between a UniDAC dataset components and data-aware controls on a form.
TUniEncryptor	The class that performs encrypting and decrypting of data.
TUniMacro	Holds the Name, Value, and Condition for a macro.
TUniMacros	Used to manage a list of TUniMacro objects for a TUniConnection component.
TUniMetaData	A component for obtaining metainformation about database objects from the server.
TUniParam	A class that is used to set the values of individual parameters passed with queries or stored procedures.
TUniParams	Used to control TUniParam objects.
TUniQuery	A component for executing queries and operating record sets. It also provides flexible way to update data.
TUniSQL	A component for executing SQL statements and calling stored procedures on the database server.

TUniStoredProc	A component for accessing and executing stored procedures and functions.
TUniTable	A component for retrieving and updating data in a single table without writing SQL statements.
TUniTransaction	A component for managing transactions in an application.
TUniUpdateSQL	A component for tuning update operations for the DataSet component.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1 TCustomUniDataSet Class

A base component for defining functionality for classes derived from it.

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

Unit

[uni](#)

Syntax

```
TCustomUniDataSet = class(TCustomDADataset);
```

Remarks

TCustomUniDataSet is a base dataset component that defines functionality for classes derived from it. Applications should never use TCustomUniDataSet objects directly. Instead of TCustomUniDataSet, they should use TCustomUniDataSet descendants, such as [TUniQuery](#) and [TUniTable](#), which inherit its dataset-related properties and methods.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

TCustomUniDataSet

See Also

- [TUniQuery](#)
- [TUniTable](#)
- [TUniStoredProc](#)
- [TUniMetaData](#)

© 1997-2025

Devart. All Rights Reserved.

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

6.20.1.1.1 Members

[TCustomUniDataSet](#) 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.
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.
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.
LastInsertId	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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	Specifies the behaviour of a TCustomUniDataSet 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	Holds the 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.
SpecificOptions	Used to provide extended

	settings for each data provider.
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.
Transaction	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject	Points to an update object component which provides update SQL statements or update objects for flexible

	data update.
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.
UpdateTransaction	Used to specify the TUniTransaction object in the context of which update commands will be executed.

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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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	Assigns a command that calls stored procedure

	specified by name to the SQL property.
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	Determines if 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 (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.
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	Provides second and other result sets while executing multireult query.
ParamByName	Accesses parameter information based on a specified parameter 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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.2 Properties

Properties of the **TCustomUniDataSet** class.

For a complete list of the **TCustomUniDataSet** class members, see the [TCustomUniDataSet 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.
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.
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.
LastInsertId	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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	Specifies the behaviour of a TCustomUniDataSet 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	Holds the 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.
SpecificOptions	Used to provide extended settings for each data provider.
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.
Transaction	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject	Points to an update object component which provides update SQL statements or update objects for flexible data update.
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.
UpdateTransaction	Used to specify the TUniTransaction object in

the context of which update commands will be executed.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.2.1 DMLRefresh Property

Used to refresh record by RETURNING clause when insert or update is performed.

Class

[TCustomUniDataSet](#)

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

[TCustomDADDataSet.RefreshOptions](#) is ignored to avoid refetching field values from the server.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.2.2 LastInsertId Property

Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.

Class

[TCustomUniDataSet](#)

Syntax

```
property LastInsertId: Int64;
```

Remarks

The **LastInsertId** property retrieves the value of an ID field after executing an INSERT statement in MySQL, PostgreSQL, SQL Server, and Microsoft Access.

MySQL:

In MySQL, the LastInsertId property retrieves the ID generated for an AUTO_INCREMENT column by the most recent query. Use this property after performing an INSERT into a table with an AUTO_INCREMENT field.

PostgreSQL:

In PostgreSQL, the LastInsertId property returns the OID value generated for an OID column in tables with OIDs by the previous query.

SQL Server:

In SQL Server, the LastInsertId property retrieves the last identity value generated in any table during the current session. The property relies on the @@IDENTITY function and requires an additional request to the server to fetch the value.

Microsoft Access:

In Microsoft Access, the LastInsertId property retrieves the value generated by the most recent query for a primary key column of type AutoNumber. Use this property after performing an INSERT into a table with an AutoNumber primary key.

If the query does not perform insertion into a table that contains field of the types specified above, the value of LastInsertId won't be defined.

Example

To improve performance and avoid additional server requests, you can generate unique identifiers within your application logic before executing INSERT statements. Here is an

example of using LastInsertId for MySQL:

```
DataSet.SQL.Text := 'INSERT INTO users (username) VALUES (:username)';  
DataSet.ParamByName('username').AsString := 'JohnDoe';  
DataSet.Execute;  
ShowMessage('Last inserted ID: ' + IntToStr(DataSet.LastInsertId));
```

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.2.3 Options Property

Specifies the behaviour of a TCustomUniDataSet object.

Class

[TCustomUniDataSet](#)

Syntax

```
property Options: TUniDataSetOptions;
```

Remarks

The [TUniDataSetOptions](#) class publishes properties defined in TDADatasetOptions. Set the properties of Options to specify the behaviour of a TCustomUniDataSet object. Their descriptions can be found in the [TUniDataSetOptions](#) topic.

See Also

- [TCustomDADataset.Options](#)

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.2.4 Params Property

Holds the parameters for a query's SQL statement.

Class

[TCustomUniDataSet](#)

Syntax

```
property Params: TUniParams 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 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

- [TUniParam](#)
- [ParamByName](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.2.5 SpecificOptions Property

Used to provide extended settings for each data provider.

Class

[TCustomUniDataSet](#)

Syntax

```
property specificOptions: TSpecificOptionsList;
```

Remarks

Use the SpecificOptions property to provide extended settings for each data provider. SpecificOptions can be setup both at design time and run time.

At design time call the component editor by double click on it, and select the Options tab in the editor. Calling the SpecificOptions editor from the Object Inspector will open the component editor with Options tab active. Type or select the provider name, and change values of required properties. Then you can either close the editor, or select another provider name. Settings for all providers will be saved.

SpecificOptions can be setup at the same time for all providers that supposed to be used.

All options are applied right before opening or executing. If an option name is not recognized, an exception is raised and the command is not executed.

For example, when you set the SequenceMode option like it is shown in the second example, you can execute the script with the Oracle provider, but attempt to use it with other providers will fail.

You can learn more about server specific options of A:OraProv_article, A:SQLProv_article, A:MySQLProv_article, A:IBProv_article, A:PgSQLProv_article in the corresponding articles.

Example

You can also setup specific options at run time. Either of two formats can be used:

1. Using the provider name in an option name;
2. Not using the provider name in an option name;

In the second case options will be applied to the current provider, namely to the provider specified in the [TUniConnection.ProviderName](#) property of the assigned connection.

```
Example 1.  
UniQuery1.SpecificOptions.Add('Oracle.ScrollableCursor=True')  
UniQuery1.SpecificOptions.Add('InterBase.FieldsAsString=True')  
Example 2.  
UniQuery1.SpecificOptions.Add('SequenceMode=smInsert')
```

See Also

- [TUniConnection.ProviderName](#)
- [Using Oracle data access provider with UniDAC in Delphi](#)
- [Using SQL Server data access provider with UniDAC in Delphi](#)
- [Using MySQL data access provider with UniDAC in Delphi](#)
- [Using InterBase data access provider with UniDAC in Delphi](#)
- [Using PostgreSQL data access provider with UniDAC in Delphi](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.2.6 Transaction Property

Used to specify the [TUniTransaction](#) object in the context of which SQL commands will be executed, and queries retrieving data will be opened.

Class

[TCustomUniDataSet](#)

Syntax

```
property Transaction: TUniTransaction stored IsTransactionStored;
```

Remarks

Use the Transaction property to specify the [TUniTransaction](#) object in the context of which SQL commands will be executed, and queries retrieving data will be opened. If this property is not specified, the default transaction associated with linked [TUniConnection](#) will be used. This transaction will work in AutoCommit mode.

See Also

- [TUniTransaction](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.2.7 UpdateObject Property

Points to an update object component which provides update SQL statements or update objects for flexible data update.

Class

[TCustomUniDataSet](#)

Syntax

```
property UpdateObject: TUniUpdateSQL;
```

Remarks

The UpdateObject property points to an update object component which provides update SQL statements or update objects for flexible data update.

See Also

- [TUniUpdateSQL](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.2.8 UpdateTransaction Property

Used to specify the TUniTransaction object in the context of which update commands will be executed.

Class

[TCustomUniDataSet](#)

Syntax

```
property UpdateTransaction: TUniTransaction;
```

Remarks

Use the UpdateTransaction property to specify the TUniTransaction object in the context of which update commands will be executed. Update commands are commands that are executed automatically, when data is edited in the dataset with Insert/Post, Edit/Post, or with other similar methods.

If this property is not specified, the transaction object specified in the [Transaction](#) property, or the default transaction associates with linked [TUniConnection](#) will be used. This transaction will work in AutoCommit mode.

See Also

- [Transaction](#)
- [TUniTransaction](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.3 Methods

Methods of the **TCustomUniDataSet** class.

For a complete list of the **TCustomUniDataSet** class members, see the [TCustomUniDataSet 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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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	Assigns a command that calls stored procedure specified by name to the SQL property.
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	Determines if 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.
GetDataTypes (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.
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	Provides second and other result sets while executing multiresult query.
ParamByName	Accesses parameter information based on a specified parameter 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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.3.1 CreateProcCall Method

Assigns a command that calls stored procedure specified by name to the SQL property.

Class

[TCustomUniDataSet](#)

Syntax

```
procedure CreateProcCall(const Name: string);
```

Parameters

Name

Holds the stored procedure name.

Remarks

Call the CreateProcCall method to assign a command that calls stored procedure specified by Name to the SQL property. The Overload parameter must contain the number of overloaded procedures. Retrieves the information about parameters of the procedure from server. After calling CreateProcCall you can execute stored procedure by the Execute method.

See Also

- [TCustomDADataset.Execute](#)

- [TCustomDACConnection.ExecProc](#)
- [TUniStoredProc](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.3.2 FindParam Method

Determines if parameter with the specified name exists in a dataset.

Class

[TCustomUniDataSet](#)

Syntax

```
function FindParam(const Value: string): TUniParam;
```

Parameters

Value

Holds the name of the param for which to search.

Return Value

the TUniParam object for the specified Name.

Remarks

Call the FindParam method to determine if parameter with the specified name exists in a dataset. Name is the name of the parameter for which to search. If FindParam finds a parameter with a matching name, it returns the TUniParam object for the specified Name. Otherwise it returns nil.

See Also

- [Params](#)
- [ParamByName](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.3.3 OpenNext Method

Provides second and other result sets while executing multiresult query.

Class

[TCustomUniDataSet](#)

Syntax

```
function OpenNext: boolean;
```

Return Value

True, if DataSet opens. If there are no record sets to be represented, it will return False and the current record set will be closed.

Remarks

Call the OpenNext method to get second and other result sets while executing multiresult query. If DataSet opens, it returns True. If there are no record sets to be represented, it will return False and the current record set will be closed.

Example

Here is a small piece of code that demonstrates the approach of working with multiple datasets returned by a multi-statement query:

```
UniQuery.SQL.Clear;
UniQuery.SQL.Add('SELECT * FROM Table1;');
UniQuery.SQL.Add('SELECT * FROM Table2;');
UniQuery.SQL.Add('SELECT * FROM Table3;');
UniQuery.SQL.Add('SELECT * FROM Table4;');
UniQuery.SQL.Add('SELECT * FROM Table5;');
UniQuery.FetchAll := False;
UniQuery.Open;
repeat
  // < do something >
until not UniQuery.OpenNext;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.1.3.4 ParamByName Method

Accesses parameter information based on a specified parameter name.

Class

[TCustomUniDataSet](#)

Syntax

```
function ParamByName(const Value: string): TUniParam;
```

Parameters

Value

Holds the name of the parameter for which to retrieve information.

Return Value

a TUniParam 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 an parameter's value at runtime and returns TUniParam object.

Example

For 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

- [TUniParam](#)
- [Params](#)
- [FindParam](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.2 TCustomUniTable Class

A base class for retrieving and updating data in a single table without writing SQL statements.

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

Unit

[Uni](#)

Syntax

```
TCustomUniTable = class(TCustomUniDataSet);
```

Remarks

TCustomUniTable is a base component that defines functionality for classes derived from it. Applications should never use TCustomUniTable objects directly. Instead, they should use [TUniTable](#), which inherits all table-related properties and methods of TCustomUniTable.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

[TCustomUniDataSet](#)

TCustomUniTable

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.2.1 Members

[TCustomUniTable](#) 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.
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 TCustomUniDataSet)	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.
LastInsertId (inherited from TCustomUniDataSet)	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get

	the value of the ID field after executing INSERT statement.
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 TCustomUniDataSet)	Specifies the behaviour of a TCustomUniDataSet 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 TCustomUniDataSet)	Holds the 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.
SpecificOptions (inherited from TCustomUniDataSet)	Used to provide extended settings for each data provider.
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 that this component encapsulates.
Transaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomUniDataSet)	Points to an update object component which provides update SQL statements or update objects for flexible data update.
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.
UpdateTransaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which update commands will be executed.

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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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 TCustomUniDataSet)	Assigns a command that calls stored procedure specified by name to the SQL property.
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 TCustomUniDataset)	Determines if 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 (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.
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 TCustomUniDataSet)	Provides second and other result sets while executing multiresult query.
ParamByName (inherited from TCustomUniDataSet)	Accesses parameter information based on a specified parameter name.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Used to determine KeyFields and build query for TUniTable.
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)	Cancel changes made to the current record when cached updates are enabled.
SaveSQL (inherited from TCustomDADataset)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.2.2 Properties

Properties of the **TCustomUniTable** class.

For a complete list of the **TCustomUniTable** class members, see the [TCustomUniTable 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.
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 TCustomUniDataSet)	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.
LastInsertId (inherited from TCustomUniDataSet)	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after

	executing INSERT statement.
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 TCustomUniDataSet)	Specifies the behaviour of a TCustomUniDataSet 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 TCustomUniDataSet)	Holds the 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.
SpecificOptions (inherited from TCustomUniDataSet)	Used to provide extended settings for each data provider.
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 that this

	component encapsulates.
Transaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomUniDataSet)	Points to an update object component which provides update SQL statements or update objects for flexible data update.
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.
UpdateTransaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which update commands will be executed.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.2.2.1 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class

[TCustomUniTable](#)

Syntax

```
property OrderFields: string;
```

Remarks

TCustomUniTable uses the OrderFields property to build ORDER BY clause of SQL statements.

Place commas to separate fields in a single string.

TCustomUniTable is reopened when the OrderFields property is being changed.

See Also

- [TCustomUniTable](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.2.2.2 TableName Property

Used to specify the name of the database table that this component encapsulates.

Class

[TCustomUniTable](#)

Syntax

```
property TableName: string;
```

Remarks

Use the TableName property to specify the name of the database table that this component encapsulates. At design-time select a valid table name from the TableName drop-down list in the Object Inspector.

See Also

- [TCustomUniTable](#)
- [TCustomDACConnection.GetTableNames](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.2.3 Methods

Methods of the **TCustomUniTable** class.

For a complete list of the **TCustomUniTable** class members, see the [TCustomUniTable 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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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 TCustomUniDataSet)	Assigns a command that calls stored procedure specified by name to the SQL property.
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 TCustomUniDataSet)	Determines if 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.
GetDataTypes (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.
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 TCustomUniDataSet)	Provides second and other result sets while executing multiresult query.
ParamByName (inherited from TCustomUniDataSet)	Accesses parameter information based on a specified parameter name.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Used to determine KeyFields and build query for TUniTable.
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)	Cancel changes made to the current record when cached updates are enabled.
SaveSQL (inherited from TCustomDADataset)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.2.3.1 PrepareSQL Method

Used to determine KeyFields and build query for TUniTable.

Class

[TCustomUniTable](#)

Syntax

```
procedure PrepareSQL;
```

Remarks

Use the PrepareSQL property to determine KeyFields and build a query for TUniTable.

PrepareSQL is called implicitly when TUniTable is being opened.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.3 TUniBlob Class

A class holding value of the BLOB fields and parameters.

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

Unit

[Uni](#)

Syntax

```
TUniBlob = class(TCompressedBlob);
```

Remarks

TUniBlob is a descendant of [TCompressedBlob](#) class. It holds value of the BLOB fields and parameters.

Note: You can affect performance of reading/writing BLOBs by changing MemData.DefaultPieceSize variable to different value. DefaultPieceSize defines size of data portion transferred through network at the single call.

Inheritance Hierarchy

[TSharedObject](#)

[TBlob](#)

[TCompressedBlob](#)

TUniBlob

See Also

- [TCompressedBlob](#)
- [TMemDataSet.GetBlob](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.3.1 Members

[TUniBlob](#) 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 (inherited from TCompressedBlob)	Used to indicate if the Blob is compressed.
CompressedSize (inherited from TCompressedBlob)	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.
---	---

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4 TUniConnection Class

A component for setting up and controlling connection to such database servers as Oracle, SQL Server, MySQL, InterBase, Firebird, and PostgreSQL.

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

Unit

[uni](#)

Syntax

```
TUniConnection = class(TCustomDAConnection);
```

Remarks

TUniConnection component is used to maintain connection to databases such as Oracle, SQL Server, MySQL, InterBase, Firebird, and PostgreSQL. Before connect you should provide connection settings such as ProviderName, Server, Username, Password, Port, and Database. Some extended connection options can be specified with the [TUniConnection.SpecificOptions](#). Set of properties that have to be assigned vary depending on used provider (the ProviderName property). To establish a database connection, it is necessary to call the [TCustomDAConnection.Connect](#) method or set the Connect property to True. There are also many properties at the connection level that affect default behavior of the queries executed within this session. Furthermore, you can control transactions using methods of this class.

All components which are dedicated to perform data access, such as TUniQuery, TUniSQL, TUniScript, must have their Connection property assigned with one of TUniConnection instances.

Inheritance Hierarchy

[TCustomDAConnection](#)

TUniConnection

See Also

- [TCustomDADataset.Connection](#)
- [TUniSQL.Connection](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.1 Members

[TUniConnection](#) class overview.

Properties

Name	Description
AutoCommit	Used to permit or prevent permanent updates, insertions, and deletions of data against the database server.
ConnectDialog (inherited from TCustomDACConnection)	Allows to link a TCustomConnectDialog component.
ConnectionString (inherited from TCustomDACConnection)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from TCustomDACConnection)	Allows customizing line breaks in string fields and parameters.
Database	Used to specify the database name that is a default source of data for SQL queries once a connection is established.
DefaultTransaction	Used to access default database connection transaction.
InTransaction (inherited from TCustomDACConnection)	Indicates whether the transaction is active.
LoginPrompt (inherited from TCustomDACConnection)	Specifies whether a login dialog appears immediately

	before opening a new connection.
Macros	Holds a collection of macros that can be used in Unified SQL statements.
Options (inherited from TCustomDACConnection)	Specifies the connection behavior.
Password (inherited from TCustomDACConnection)	Serves to supply a password for login.
Pooling (inherited from TCustomDACConnection)	Enables or disables using connection pool.
PoolingOptions (inherited from TCustomDACConnection)	Specifies the behaviour of connection pool.
Port	Used to specify the port number for TCP/IP connection.
ProviderName	Used to switch the current data access provider.
Server (inherited from TCustomDACConnection)	Serves to supply the server name for login.
SpecificOptions	Used to provide extended settings for each data provider.
Username (inherited from TCustomDACConnection)	Used to supply a user name for login.

Methods

Name	Description
ActiveMacroValueByName	Returns the value of the specified macro for the current provider.
ApplyUpdates (inherited from TCustomDACConnection)	Overloaded. Applies changes in datasets.
AssignConnect	Shares database connection between the TUniConnection components.
Commit (inherited from TCustomDACConnection)	Commits current transaction.
CommitRetaining	Permanently stores all changes of data associated with the default database

	transaction to the database and then retains the transaction context.
Connect (inherited from TCustomDACConnection)	Establishes a connection to the server.
CreateDataSet	Creates an instance of the TCustomUniDataSet class and assigns its TCustomDADataSet.Connection property.
CreateSQL	Creates an instance of the TUniSQL class and assigns its TUniSQL.Connection property.
CreateTransaction	Creates an instance of the TUniTransaction class and adds itself to its TUniTransaction.Connections .
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.
GetStoredProcNames (inherited from TCustomDACConnection)	Returns a list of stored procedures from the server.
GetTableNames (inherited from	Provides a list of available tables names.

TCustomDACConnection)	
MonitorMessage (inherited from TCustomDACConnection)	Sends a specified message through the TCustomDASQLMonitor component.
ParamByName	Provides access to output parameters and their values after executing an SQL statement with the TCustomDACConnection.ExeSQL method.
PerformConnect (inherited from TCustomDACConnection)	Establishes a connection to the server regardless of the LoginPrompt property.
Ping (inherited from TCustomDACConnection)	Used to check state of connection to the server.
ReleaseSavepoint	Destroys 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.
RollbackRetaining	Used to roll back all changes of data associated with the transaction and retain the transaction context.
RollbackToSavepoint	Cancels all updates for the current transaction.
Savepoint	Defines a point in the transaction to which you can later roll back.
StartTransaction	Overloaded. Starts a new transaction at the server.

Events

Name	Description
OnConnectionLost (inherited from	This event occurs when connection was lost.

TCustomDACConnection)	
OnError (inherited from TCustomDACConnection)	This event occurs when an error has arisen in the connection.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.2 Properties

Properties of the **TUniConnection** class.

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

Public

Name	Description
ConnectDialog (inherited from TCustomDACConnection)	Allows to link a TCustomConnectDialog component.
ConnectionString (inherited from TCustomDACConnection)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from TCustomDACConnection)	Allows customizing line breaks in string fields and parameters.
InTransaction (inherited from TCustomDACConnection)	Indicates whether the transaction is active.
LoginPrompt (inherited from TCustomDACConnection)	Specifies whether a login dialog appears immediately before opening a new connection.
Options (inherited from TCustomDACConnection)	Specifies the connection behavior.
Password (inherited from TCustomDACConnection)	Serves to supply a password for login.
Pooling (inherited from TCustomDACConnection)	Enables or disables using connection pool.
PoolingOptions (inherited from	Specifies the behaviour of connection pool.

TCustomDACConnection)	
Server (inherited from TCustomDACConnection)	Serves to supply the server name for login.
Username (inherited from TCustomDACConnection)	Used to supply a user name for login.

Published

Name	Description
AutoCommit	Used to permit or prevent permanent updates, insertions, and deletions of data against the database server.
Database	Used to specify the database name that is a default source of data for SQL queries once a connection is established.
DefaultTransaction	Used to access default database connection transaction.
Macros	Holds a collection of macros that can be used in Unified SQL statements.
Port	Used to specify the port number for TCP/IP connection.
ProviderName	Used to switch the current data access provider.
SpecificOptions	Used to provide extended settings for each data provider.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.2.1 AutoCommit Property

Used to permit or prevent permanent updates, insertions, and deletions of data against the database server.

Class

[TUniConnection](#)

Syntax

```
property AutoCommit: boolean;
```

Remarks

Use the AutoCommit property to permit or prevent permanent updates, insertions, and deletions of data against the database server without explicit calls to Commit or Rollback methods.

Set AutoCommit to True to permit implicit call to Commit method after every database access. The default value is True.

Note: The AutoCommit property in TUniConnection globally specifies whether all queries to modify database are implicitly committed or not. When using the InterBase provider, [TUniTable](#), [TUniQuery](#), [TUniStoredProc](#), [TUniSQL](#) and [TUniLoader](#) components have their own AutoCommit specific options. This allows them to selectively specify their implicit transaction committing behavior after each data modifying access. The AutoCommit specific option behaviour is described in the [UniDAC and InterBase/Firebird](#) article.

Example

This procedure removes all records from Dept table and makes this change permanent.

```
procedure TForm1.DeleteClick(Sender: TObject);  
begin  
    UniSQL.Connection := UniConnection;  
    UniConnection.AutoCommit := False;  
    UniSQL.SQL.Text := 'DELETE FROM Dept';  
    UniSQL.Execute; // delete all records, commit is not performed  
    UniConnection.Rollback; // restore deleted records  
    UniConnection.AutoCommit := True;  
    UniSQL.SQL.Text := 'DELETE FROM Dept';  
    UniSQL.Execute; // delete all records, commit is performed  
    UniConnection.Rollback; // couldn't restore deleted records  
end;
```

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.2.2 Database Property

Used to specify the database name that is a default source of data for SQL queries once a connection is established.

Class

[TUniConnection](#)

Syntax

```
property Database: string;
```

Remarks

Use the Database property to specify the database name that is a default source of data for SQL queries once a connection is established.

Altering the Database property makes new database name take effect immediately.

This property is available for Access, Advantage, SAP Sybase ASE, DB2, DBF, InterBase, MySQL, NexusDB, PostgreSQL, SQL Server, and SQLite providers.

SQL Server provider note:

When Database is not assigned, the SQL Server provider will use the default database for the current SQL Server login specified in the [TCustomDACConnection.Username](#) property.

See Also

- [TCustomDACConnection.Server](#)
- [TCustomDACConnection.Username](#)
- [TCustomDACConnection.Password](#)

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.2.3 DefaultTransaction Property

Used to access default database connection transaction.

Class

[TUniConnection](#)

Syntax

```
property DefaultTransaction: TUniTransaction;
```

Remarks

Use the DefaultTransaction property to access default database connection transaction. By default this is internal connection transaction. You can set it to external transaction component. To restore internal transaction set this property to nil.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.2.4 Macros Property

Holds a collection of macros that can be used in Unified SQL statements.

Class

[TUniConnection](#)

Syntax

```
property Macros: TUniMacros stored IsMacrosStored;
```

Remarks

The Macros property holds a collection of macros that can be used in Unified SQL statements.

Connection Macros are defined by "{MacroName}" and affect all associated datasets.

To work with Macros you can use traditional or "predefined" way.

For detailed information on using macros refer to article [Unified SQL](#) .

Example

Here is the traditional way to work with macros:

```
if UniConnection.ProviderName = 'Oracle' then
  UniConnection.MacroByName('tablename').Value := 'dept'
else
  if UniConnection.ProviderName = 'MySQL' then
    UniConnection.MacroByName('tablename').Value := 'test.dept';
```

See Also

- [Unified SQL](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.2.5 Port Property

Used to specify the port number for TCP/IP connection.

Class

[TUniConnection](#)

Syntax

```
property Port: integer default DefValPort;
```

Remarks

Use the Port property to specify the port number for TCP/IP connection. This property is available only for the MySQL provider.

The default value is 0.

See Also

- [TCustomDACConnection.Server](#)
- [Database](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.2.6 ProviderName Property

Used to switch the current data access provider.

Class

[TUniConnection](#)

Syntax

```
property ProviderName: string;
```

Remarks

UniDAC consists of [two constituents](#). The first constituent is the general UniDAC Engine that provides unified programming interface for developers. The second constituent is the data access layer which consists of data access providers. These providers are intended for interacting between UniDAC Engine and database servers.

The ProviderName property is intended to switch the current data access provider. If the value of ProviderName is changed while a connection is active, the connection will be forced to close. The following four providers names are acceptable:

- Oracle - provider for Oracle;
- SQL Server - provider for Microsoft SQL Server;
- MySQL - provider for MySQL;
- InterBase - provider for InterBase, Firebird, and Yaffil database servers.
- PostgreSQL - provider for PostgreSQL.

See Also

- [TCustomDACConnection.Server](#)
- [Database](#)
- [Port](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.2.7 SpecificOptions Property

Used to provide extended settings for each data provider.

Class

[TUniConnection](#)

Syntax

```
property specificOptions: TspecificOptionsList;
```

Remarks

Use the SpecificOptions property to provide extended settings for each data provider. SpecificOptions can be setup both in design time and run time.

At design time call the component editor by double click on it, and select the Options tab in the editor. Calling the SpecificOptions editor from the Object Inspector will open the component editor with Options tab active. Type or select the provider name, and change values of required properties. Then you can either close the editor, or select another provider name. Settings for all providers will be saved.

SpecificOptions can be setup at the same time for all providers that supposed to be used.

All options are applied at the connect time. If an option name is not recognized, an exception is raised and connection is not established.

For example, when you set the Direct option like it is shown in the second example, you can connect with the Oracle and MySQL provider, but attempt to connect with SQL Server and InterBase providers will fail.

Example

You can also setup specific options at run time. Either of two formats can be used:

1. Using the provider name in an option name;
2. Not using the provider name in an option name;

In the second case options will be applied to the current provider, namely to the provider specified in the [ProviderName](#) property.

```
Example 1.  
UniConnection1.SpecificOptions.Add('oracle.Direct=True')
```

```
UniConnection1.SpecificOptions.Add('InterBase.CharLength=0')
Example 2.
UniConnection1.SpecificOptions.Add('Direct=True')
```

See Also

- [ProviderName](#)
- [Using Oracle data access provider with UniDAC in Delphi](#)
- [Using SQL Server data access provider with UniDAC in Delphi](#)
- [Using MySQL data access provider with UniDAC in Delphi](#)
- [Using InterBase data access provider with UniDAC in Delphi](#)
- [Using PostgreSQL data access provider with UniDAC in Delphi](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3 Methods

Methods of the **TUniConnection** class.

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

Public

Name	Description
ActiveMacroValueByName	Returns the value of the specified macro for the current provider.
ApplyUpdates (inherited from TCustomDACConnection)	Overloaded. Applies changes in datasets.
AssignConnect	Shares database connection between the TUniConnection components.
Commit (inherited from TCustomDACConnection)	Commits current transaction.
CommitRetaining	Permanently stores all changes of data associated with the default database transaction to the database and then retains the

	transaction context.
Connect (inherited from TCustomDACConnection)	Establishes a connection to the server.
CreateDataSet	Creates an instance of the TCustomUniDataSet class and assigns its TCustomDADataset.Connection property.
CreateSQL	Creates an instance of the TUniSQL class and assigns its TUniSQL.Connection property.
CreateTransaction	Creates an instance of the TUniTransaction class and adds itself to its TUniTransaction.Connections .
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.
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.
ParamByName	Provides access to output parameters and their values after executing an SQL statement with the TCustomDACConnection.ExecuteSQL method.
PerformConnect (inherited from TCustomDACConnection)	Establishes a connection to the server regardless of the LoginPrompt property.
Ping (inherited from TCustomDACConnection)	Used to check state of connection to the server.
ReleaseSavepoint	Destroys 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.
RollbackRetaining	Used to roll back all changes of data associated with the transaction and retain the transaction context.
RollbackToSavepoint	Cancels all updates for the current transaction.
Savepoint	Defines a point in the transaction to which you can later roll back.
StartTransaction	Overloaded. Starts a new transaction at the server.

See Also

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

Reserved.

6.20.1.4.3.1 ActiveMacroValueByName Method

Returns the value of the specified macro for the current provider.

Class

[TUniConnection](#)

Syntax

```
function ActiveMacroValueByName(const Name: string): Variant;
```

Parameters

Name

The name of the macro.

Return Value

The value of the specified macro.

See Also

- [Unified SQL](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3.2 AssignConnect Method

Shares database connection between the TUniConnection components.

Class

[TUniConnection](#)

Syntax

```
procedure AssignConnect(Source: TUniConnection);
```

Parameters

Source

Preconnected TUniConnection component which connection is to be shared with the current TUniConnection component.

Remarks

Use the AssignConnect method to share database connection between the TUniConnection components.

AssignConnect assumes that the Source parameter points to a preconnected TUniConnection component which connection is to be shared with the current TUniConnection component. Note that AssignConnect doesn't make any references to the Source TUniConnection component. So before disconnecting parent TUniConnection component call AssignConnect(Nil) or the Disconnect method for all assigned connections.

See Also

- [TCustomDACConnection.Connect](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3.3 CommitRetaining Method

Permanently stores all changes of data associated with the default database transaction to the database and then retains the transaction context.

Class

[TUniConnection](#)

Syntax

```
procedure CommitRetaining;
```

Remarks

Call the CommitRetaining method to permanently store to the database server all changes of data associated with the default database transaction and then retain the transaction context.

See Also

- [TCustomDACConnection.Commit](#)
- [TCustomDACConnection.StartTransaction](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3.4 CreateDataSet Method

Creates an instance of the [TCustomUniDataSet](#) class and assigns its [TCustomDADataset.Connection](#) property.

Class

[TUniConnection](#)

Syntax

```
function CreateDataSet(AOwner: TComponent = nil):  
TCustomDADataset; override;
```

Return Value

an instance of the class.

Remarks

Call the CreateDataSet method to create an instance of the [TCustomUniDataSet](#) class and assign its [TCustomDADataset.Connection](#) property.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3.5 CreateSQL Method

Creates an instance of the [TUniSQL](#) class and assigns its [TUniSQL.Connection](#) property.

Class

[TUniConnection](#)

Syntax

```
function CreateSQL: TCustomDASQL; override;
```

Return Value

an instance of the class.

Remarks

Call the CreateSQL method creates an instance of the [TUniSQL](#) class and assign its [TUniSQL.Connection](#) property.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.20.1.4.3.6 CreateTransaction Method

Creates an instance of the [TUniTransaction](#) class and adds itself to its [TUniTransaction.Connections](#).

Class

[TUniConnection](#)

Syntax

```
function CreateTransaction: TDATransaction; override;
```

Return Value

an instance of the class.

Remarks

Call the CreateTransaction method to create an instance of the [TUniTransaction](#) class and add itself to its [TUniTransaction.Connections](#).

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3.7 ParamByName Method

Provides access to output parameters and their values after executing an SQL statement with the [TCustomDACConnection.ExecSQL](#) method.

Class

[TUniConnection](#)

Syntax

```
function ParamByName(const Name: string): TUniParam;
```

Parameters

Name

Holds the parameter name (should be equal to the one that occurred in the SQL statement).

Return Value

a reference for the matching parameter.

Remarks

Call the ParamByName method to get access to output parameters and their values after executing an SQL statement with the [TCustomDACConnection.ExecSQL](#) method. The Name parameter should equal to the parameter name as it occurred in the SQL statement.

This method implicitly calls the [TUniSQL.ParamByName](#) method of [TUniSQL](#).

See Also

- [TCustomDACConnection.ExecSQL](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3.8 ReleaseSavepoint Method

Destroys the specified savepoint without affecting any work that has been performed after its creation.

Class

[TUniConnection](#)

Syntax

```
procedure ReleaseSavepoint(const Name: string);
```

Parameters

Name

Holds the savepoint name.

Remarks

Call the ReleaseSavepoint method to destroy the specified savepoint without affecting any work that has been performed after its creation.

See Also

- [Savepoint](#)
- [RollbackToSavepoint](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.20.1.4.3.9 RollbackRetaining Method

Used to roll back all changes of data associated with the transaction and retain the transaction context.

Class

[TUniConnection](#)

Syntax

```
procedure RollbackRetaining;
```

Remarks

Use the RollbackRetaining method to roll back all changes of data associated with the transaction and retain the transaction context.

Note: this method is only supported for the InterBase provider.

See Also

- [TCustomDACConnection.Rollback](#)
- [TCustomDACConnection.StartTransaction](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3.10 RollbackToSavepoint Method

Cancels all updates for the current transaction.

Class

[TUniConnection](#)

Syntax

```
procedure RollbackToSavepoint(const Name: string);
```

Parameters

Name

Holds the savepoint name.

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](#)
- [TCustomDACConnection.Rollback](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3.11 Savepoint Method

Defines a point in the transaction to which you can later roll back.

Class

[TUniConnection](#)

Syntax

```
procedure Savepoint(const Name: string);
```

Parameters

Name

Holds a valid name for identifying a savepoint.

Remarks

Call the `Savepoint` method to define a point in the transaction to which you can later roll back. 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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.4.3.12 StartTransaction Method

Starts a new transaction at the server.

Class

[TUniConnection](#)

Overload List

Name	Description
StartTransaction	Call the StartTransaction method to begin a new transaction at the server.
StartTransaction(IsolationLevel: TCRIsoationLevel; ReadOnly: boolean)	Starts a new transaction at the server, and specifies whether the transaction is read-only and how database modifications should be handled.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Call the StartTransaction method to begin a new transaction at the server.

Class

[TUniConnection](#)

Syntax

```
procedure StartTransaction; overload; override;
```

Remarks

Call the StartTransaction method to begin a new transaction at the server. Before calling StartTransaction, an application should check the value of the [TCustomDACConnection.InTransaction](#) property. If the result is True, it means 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 Exception. Calling StartTransaction when connection is closed also raises

Exception.

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.

Use the `IsolationLevel` property to specify how transactions containing database modifications are handled.

Values of the `TCRIsolationLevel` enumeration correspond to the following isolation levels of supported database servers:

	SQL standard	Oracle	SQL Server	MySQL	InterBase/Firebird
<code>ilReadCommitted</code>	ReadCommitted	<code>ilReadCommitted</code>	<code>ilReadCommitted</code>	<code>ilReadCommitted</code>	<code>ibReadCommitted</code>
<code>ilReadUnCommitted</code>	ReadUnCommitted	-	<code>ilReadUnCommitted</code>	<code>ilReadUnCommitted</code>	-
<code>ilRepeatableRead</code>	RepeatableRead	-	<code>ilRepeatableRead</code>	<code>ilRepeatableRead</code>	-
<code>ilIsolated</code>	Serializable	-	<code>ilIsolated</code>	<code>ilSerializable</code>	<code>ibTableStability</code>
<code>ilSnapshot</code>	Serializable without locks	<code>ilSerializable</code>	<code>ilSnapshot</code>	-	<code>ibSnapshot</code>
<code>ilCustom</code>	<i>This value is introduced for future needs. Currently not implemented.</i>				

The `ReadOnly` parameter determines that a read-only transaction will be started. It means that data within the transaction can not be modified. You will get an exception on attempt to post any changes.

The `ReadOnly` parameter has sense only for Oracle and InterBase providers.

See Also

- [TCustomDACConnection.Commit](#)
- [TCustomDACConnection.Rollback](#)
- [TCustomDACConnection.InTransaction](#)
- [StartTransaction](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Starts a new transaction at the server, and specifies whether the transaction is read-only and how database modifications should be handled.

Class

[TUniConnection](#)

Syntax

```
procedure StartTransaction(IsolationLevel: TCRIsolationLevel;  
ReadOnly: boolean = False); reintroduce; overload;
```

Parameters

IsolationLevel

Specifies how transactions containing database modifications are handled.

ReadOnly

if True, a read-only transaction will be started.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.5 TUniDataSetOptions Class

Specifies the behaviour of a TCustomUniDataSet object.

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

Unit

[uni](#)

Syntax

```
TUniDataSetOptions = class (TDADatasetOptions);
```

Remarks

The [TUniDataSetOptions](#) class publishes properties defined in TDADatasetOptions. Set the properties of Options to specify the behaviour of a TCustomUniDataSet object.

Inheritance Hierarchy

[TDADatasetOptions](#)

TUniDataSetOptions

See Also

- [TCustomDADataset.Options](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.5.1 Members

[TUniDataSetOptions](#) class overview.

Properties

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.
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.
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.
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.
TrimVarChar	Used to specify whether to discard all trailing spaces in the variable-length 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.
--	---

© 1997-2025

Devart. All Rights Reserved.

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

6.20.1.5.2 Properties

Properties of the **TUniDataSetOptions** class.

For a complete list of the **TUniDataSetOptions** class members, see the [TUniDataSetOptions 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
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.
FullRefresh	Used to specify the fields to include in the automatically generated SQL statement when calling the method.
SetEmptyStrToNull	Force replace of empty strings with NULL values in data. The default value is False.
TrimVarChar	Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.5.2.1 EnableBCD Property

Used to enable currency type. Default value of this option is False.

Class

[TUniDataSetOptions](#)

Syntax

```
property EnableBCD: boolean;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.5.2.2 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class

[TUniDataSetOptions](#)

Syntax

```
property EnableFMTBCD: boolean;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.5.2.3 FullRefresh Property

Used to specify the fields to include in the automatically generated SQL statement when calling the method.

Class

[TUniDataSetOptions](#)

Syntax

```
property FullRefresh: boolean;
```

Remarks

Use the FullRefresh property to specify what fields to include in the automatically generated SQL statement when calling the [TCustomDADDataSet.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 [TUniQuery.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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.5.2.4 SetEmptyStrToNull Property

Force replace of empty strings with NULL values in data. The default value is False.

Class

[TUniDataSetOptions](#)

Syntax

```
property SetEmptyStrToNull: boolean;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.5.2.5 TrimVarChar Property

Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

Class

[TUniDataSetOptions](#)

Syntax

```
property TrimVarChar: boolean;
```

Remarks

Use the TrimVarChar property to specify whether to discard all trailing spaces in the variable-length string fields of a dataset. The default value is False.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.6 TUniDataSource Class

TUniDataSource provides an interface between a UniDAC dataset components and data-aware controls on a form.

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

Unit

[Uni](#)

Syntax

```
TUniDataSource = class(TCRDataSource);
```

Remarks

TUniDataSource provides an interface between a UniDAC dataset components and data-aware controls on a form.

TUniDataSource inherits its functionality directly from the TDataSource component.

At design-time assign individual data-aware components' DataSource properties from their drop-down listboxes.

If you place onto a form a TUniDataSource component close to a dataset, this dataset will be linked to it automatically.

Inheritance Hierarchy

[TCRDataSource](#)

TUniDataSource

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.6.1 Members

[TUniDataSource](#) class overview.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.7 TUniEncryptor Class

The class that performs encrypting and decrypting of data.

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

Unit

[uni](#)

Syntax

```
TUniEncryptor = class(TCREncryptor);
```

Inheritance Hierarchy

[TCREncryptor](#)

TUniEncryptor

© 1997-2025

Devart. All Rights Reserved.

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

6.20.1.7.1 Members

[TUniEncryptor](#) 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.

© 1997-2025

Devart. All Rights Reserved.

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

6.20.1.8 TUniMacro Class

Holds the Name, Value, and Condition for a macro.

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

Unit

[uni](#)

Syntax

```
TUniMacro = class(TCollectionItem);
```

Remarks

A TUniMacro object holds the Name, Value, and Condition for a macro. This macro can be used in Unified SQL statements.

For detailed information on using macros refer to article [Unified SQL](#).

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.8.1 Members

[TUniMacro](#) class overview.

Properties

Name	Description
Condition	Holds a condition for the macro, which determines whether macro is evaluated to its Value or an empty string.
Name	Used to refer to this macro in Unified SQL statements and other macros.
Value	Holds a string expression that macro evaluates to if Condition is enabled.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.8.2 Properties

Properties of the **TUniMacro** class.

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

Published

Name	Description
Condition	Holds a condition for the macro, which determines whether macro is evaluated to its Value or an empty string.
Name	Used to refer to this macro in Unified SQL statements and other macros.
Value	Holds a string expression that macro evaluates to if Condition is enabled.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.8.2.1 Condition Property

Holds a condition for the macro, which determines whether macro is evaluated to its Value or an empty string.

Class

[TUniMacro](#)

Syntax

```
property Condition: string;
```

Remarks

The Condition property holds a condition for the macro, which determines whether macro is evaluated to its Value or an empty string.

Macro condition is name of another custom TUniMacro or predefined macro like MySQL, Oracle, etc. If the condition macro is defined, the current macro evaluates to what is specified in the Value property, otherwise it returns empty string.

If the condition is not specified (represents empty string), then macro always evaluates to Value.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.8.2.2 Name Property

Used to refer to this macro in Unified SQL statements and other macros.

Class

[TUniMacro](#)

Syntax

```
property Name: string;
```

Remarks

Macro identifier to be used in Unified SQL statements.

The Name property is used to refer to this macro in Unified SQL statements and other macros. If there are several macros with same name in Macros of TUniConnection, the one that has valid condition is used.

When the macro is used in statements or as part of value of another macro, you should enclose the Name in braces {...}. When used as condition for another macro, the braces are not required.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.8.2.3 Value Property

Holds a string expression that macro evaluates to if Condition is enabled.

Class

[TUniMacro](#)

Syntax

```
property value: string;
```

Remarks

The Value property holds a string expression that macro evaluates to if Condition is enabled.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.9 TUniMacros Class

Used to manage a list of TUniMacro objects for a TUniConnection component.

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

Unit

[uni](#)

Syntax

```
TUniMacros = class(TOwnedCollection);
```

Remarks

Use TUniMacros to manage a list of TUniMacro objects for a TUniConnection component.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.9.1 Members

[TUniMacros](#) class overview.

Properties

Name	Description
Items	Used to iterate through all macros.

Methods

Name	Description
Add	Used to add a macro.

FindMacro	Searches for a TUniMacro object by its name.
MacroByName	Used to search for a macro with the specified name.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.9.2 Properties

Properties of the **TUniMacros** class.

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

Public

Name	Description
Items	Used to iterate through all macros.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.9.2.1 Items Property(Indexer)

Used to iterate through all macros.

Class

[TUniMacros](#)

Syntax

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

Parameters

Index

Holds an index in the range 0..Count - 1.

Remarks

Use the `Items` property to iterate through all macros. `Index` identifies the index in the range `0..Count - 1`. `Items` can reference a particular macro by its index, but the [MacroByName](#) method is preferred in order to avoid depending on the order of the macros.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.9.3 Methods

Methods of the **TUniMacros** class.

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

Public

Name	Description
Add	Used to add a macro.
FindMacro	Searches for a TUniMacro object by its name.
MacroByName	Used to search for a macro with the specified name.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.9.3.1 Add Method

Used to add a macro.

Class

[TUniMacros](#)

Syntax

```
procedure Add(const Name: string; const Value: string; const
Condition: string = '');
```

Parameters

Name

Holds the name of the macro

Value

Holds the value of the macro

Condition

Specifies the provider that the condition is applied to.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.9.3.2 FindMacro Method

Searches for a TUniMacro object by its name.

Class

[TUniMacros](#)

Syntax

```
function FindMacro(const Name: string): TUniMacro;
```

Parameters

Name

Holds the name of a macro to search for.

Return Value

TMacro object if a match was found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the name passed in Name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method rather than a direct reference to the [Items](#) property to avoid depending on the order of the entries.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.9.3.3 MacroByName Method

Used to search for a macro with the specified name.

Class

[TUniMacros](#)

Syntax

```
function MacroByName(const Name: string): TUniMacro;
```

Parameters

Name

Call the MacroByName method to find a Macro with the name passed in Value. If a match is found, MacroByName returns the Macro. Otherwise, an exception is raised. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate a macro by name without raising an exception if the parameter is not found, use the [FindMacro](#) method.

Return Value

TUniMacro object, if a macro with specified name was found.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.10 TUniMetaData Class

A component for obtaining metainformation about database objects from the server.

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

Unit

[Uni](#)

Syntax

```
TUniMetaData = class(TDAMetaData);
```

Remarks

The TUniMetaData 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](#)

TUniMetaData

See Also

- [TCustomDADataset.Debug](#)
- [TCustomDASQL.Debug](#)
- [DBMonitor](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.10.1 Members

[TUniMetaData](#) 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 the connection which will be used by TUniMetaData to request metadata from server.
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.
Transaction	Used to set or return the transaction to be used by the component.
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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.10.2 Properties

Properties of the **TUniMetaData** class.

For a complete list of the **TUniMetaData** class members, see the [TUniMetaData 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.
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.

Published

Name	Description
Connection	Used to specify the connection which will be used by TUniMetaData to request metadata from server.
Transaction	Used to set or return the transaction to be used by the component.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.10.2.1 Connection Property

Used to specify the connection which will be used by TUniMetaData to request metadata from server.

Class

[TUniMetaData](#)

Syntax

```
property Connection: TUniConnection;
```

Remarks

Use the Connection property to specify the connection which will be used by TUniMetaData to request metadata from server. If Connection is not connected, TUniMetaData will try to establish connection using the Connect method of the associated TUniConnection object as soon as it will be necessary.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.10.2.2 Transaction Property

Used to set or return the transaction to be used by the component.

Class

[TUniMetaData](#)

Syntax

```
property Transaction: TUniTransaction stored IsTransactionStored;
```

Remarks

Use the Transaction property to set or return the transaction to be used by the component.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.11 TUniParam 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 [TUniParam](#) members.

Unit

[uni](#)

Syntax

```
TUniParam = class(TDAParam);
```

Remarks

Use the properties of TUniParam to set the value of a parameter. Objects that use parameters create TUniParam objects to represent these parameters. For example, TUniParam objects are used by TUniSQL, TCustomUniDataSet.

TUniParam 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 TUniParam object. Conversely, TUniParam includes properties that indicate how the field value is passed as a

parameter.

Inheritance Hierarchy

[TDAParam](#)

TUniParam

See Also

- [TCustomUniDataSet](#)
- [TUniSQL](#)
- [TUniParams](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.11.1 Members

[TUniParam](#) 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.
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.

Reserved.

6.20.1.12 TUniParams Class

Used to control TUniParam objects.

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

Unit

[Uni](#)

Syntax

```
TUniParams = class(TDAParams);
```

Remarks

Use TUniParams to manage a list of TUniParam objects for an object that uses field parameters. For example, TUniStoredProc objects and TUniQuery objects use TUniParams objects to create and access their parameters.

Inheritance Hierarchy

[TDAParams](#)

TUniParams

See Also

- [TUniParam](#)
- [TCustomDASQL.Params](#)
- [TCustomDADDataSet.Params](#)
- [TCustomDADDataSet.Params](#)
- [TCustomDASQL.Params](#)
- [TUniParam](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.12.1 Members

[TUniParams](#) class overview.

Properties

Name	Description
Items (inherited from TDAParams)	Used to iterate through all 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.13 TUniQuery 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 [TUniQuery](#) members.

Unit

[Uni](#)

Syntax

```
TUniQuery = class(TCustomUniDataSet);
```

Remarks

TUniQuery is a direct descendant of the [TCustomUniDataSet](#) component. It publishes most of its inherited properties and events so that they can be manipulated at design-time.

Use TUniQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. TUniQuery 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, TUniQuery will retrieve primary keys for UpdatingTable from metadata. TUniQuery 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 [TCustomDADDataSet.BeforeUpdateExecute](#) event to assign the value to additional parameters and the [TCustomDADDataSet.AfterUpdateExecute](#) event to read them.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADDataSet](#)

[TCustomUniDataSet](#)

TUniQuery

See Also

- [Master/Detail Relationships](#)
- [TUniStoredProc](#)
- [TUniTable](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.13.1 Members

[TUniQuery](#) 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.
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 TCustomUniDataSet)	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.
LastInsertId (inherited from TCustomUniDataSet)	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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 TCustomUniDataSet)	Specifies the behaviour of a TCustomUniDataSet 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 TCustomUniDataSet)	Holds the 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.
SpecificOptions (inherited from TCustomUniDataSet)	Used to provide extended settings for each data provider.
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.
Transaction (inherited from TCustomUniDataset)	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomUniDataset)	Points to an update object component which provides update SQL statements or update objects for flexible data update.
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.
UpdateTransaction (inherited from TCustomUniDataset)	Used to specify the TUniTransaction object in the context of which update commands will be executed.
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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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 TCustomUniDataSet)	Assigns a command that calls stored procedure specified by name to the SQL property.
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 TCustomUniDataSet)	Determines if 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 (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.
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 TCustomUniDataSet)	Provides second and other result sets while executing multiresult query.
ParamByName (inherited from TCustomUniDataSet)	Accesses parameter information based on a specified parameter 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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.13.2 Properties

Properties of the **TUniQuery** class.

For a complete list of the **TUniQuery** class members, see the [TUniQuery 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.
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 TCustomUniDataSet)	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.
LastInsertId (inherited from TCustomUniDataSet)	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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 TCustomUniDataSet)	Specifies the behaviour of a TCustomUniDataSet 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 TCustomUniDataSet)	Holds the 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.
SpecificOptions (inherited from TCustomUniDataSet)	Used to provide extended settings for each data provider.
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.
Transaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomUniDataSet)	Points to an update object component which provides update SQL statements or update objects for flexible data update.
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.
UpdateTransaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which update commands will be executed.

Published

Name	Description
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.13.2.1 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

[TUniQuery](#)

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

- [TUniStoredProc.LockMode](#)
- [TUniTable.LockMode](#)

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.20.1.13.2.2 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

[TUniQuery](#)

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 [TCustomUniDataSet.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

For example:

1. For the query where the only allowed value for UpdatingTable property is 'Orders';
2. For the query where allowed values for UpdatingTable are 'Orders' and 'Order Details'.

In the first case (or on default) editable field is ShipName, in the second - Quantity field.

Example 1.

```
SELECT OrderID, ShipName FROM Orders;
```

Example 2.

```
SELECT A.OrderID, A.ShipName, B.Quantity FROM Orders A,  
[Order Details] B WHERE (A.OrderID=B.OrderID);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14 TUniSQL 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 [TUniSQL](#) members.

Unit

[Uni](#)

Syntax

```
TUniSQL = class(TCustomDASQL);
```

Remarks

The TUniSQL component is a direct descendant of the [TCustomDASQL](#) class.

Use The TUniSQL 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](#)

TUniSQL

See Also

- [TUniQuery](#)
- [TUniScript](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.1 Members

[TUniSQL](#) class overview.

Properties

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

ChangeCursor (inherited from TCustomDASQL)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
Connection	Used to specify the connection in which the script will be executed.
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.
LastInsertId	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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 (inherited from TCustomDASQL)	Used to contain parameters for a 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.
SpecificOptions	Provides extended settings for each data provider.
SQL (inherited from TCustomDASQL)	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.
Transaction	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.

Methods

Name	Description
BreakExec (inherited from TCustomDASQL)	Breaks execution of an SQL statement on the server.
CreateProcCall	Assigns a command that calls stored procedure specified by Name to the SQL property.
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 a parameter with the specified name.
MacroByName (inherited from TCustomDASQL)	Finds a macro with the specified name.
ParamByName	Searches for 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.2 Properties

Properties of the **TUniSQL** class.

For a complete list of the **TUniSQL** class members, see the [TUniSQL 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.
LastInsertId	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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 (inherited from TCustomDASQL)	Used to contain parameters for a 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.

Published

Name	Description
Connection	Used to specify the connection in which the script will be executed.
SpecificOptions	Provides extended settings for each data provider.
Transaction	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data

will be opened.

See Also

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

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.2.1 Connection Property

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

Class

[TUniSQL](#)

Syntax

```
property Connection: TUniConnection;
```

Remarks

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

See Also

- [TUniConnection](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.2.2 LastInsertId Property

Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.

Class

[TUniSQL](#)

Syntax

```
property LastInsertId: Int64;
```

Remarks

The **LastInsertId** property retrieves the value of an ID field after executing an INSERT statement in MySQL, PostgreSQL, SQL Server, and Microsoft Access.

MySQL:

In MySQL, the LastInsertId property retrieves the ID generated for an AUTO_INCREMENT column by the most recent query. Use this property after performing an INSERT into a table with an AUTO_INCREMENT field.

PostgreSQL:

In PostgreSQL, the LastInsertId property returns the OID value generated for an OID column in tables with OIDs by the previous query.

SQL Server:

In SQL Server, the LastInsertId property retrieves the last identity value generated in any table during the current session. The property relies on the @@IDENTITY function and requires an additional request to the server to fetch the value.

Microsoft Access:

In Microsoft Access, the LastInsertId property retrieves the value generated by the most recent query for a primary key column of type AutoNumber. Use this property after performing an INSERT into a table with an AutoNumber primary key.

If the query does not perform insertion into a table that contains field of the types specified above, the value of LastInsertId won't be defined.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.2.3 SpecificOptions Property

Provides extended settings for each data provider.

Class

[TUniSQL](#)

Syntax

```
property SpecificOptions: TSpecificOptionsList;
```

Remarks

Use the SpecificOptions property to provide extended settings for each data provider. SpecificOptions can be setup both design time and run time.

At design time call the component editor by double click on it, and select the Options tab in the editor. Calling the SpecificOptions editor from the Object Inspector will open the component editor with Options tab active. Type or select the provider name, and change values of required properties. Then you can either close the editor, or select another provider name. Settings for all providers will be saved.

SpecificOptions can be setup at the same time for all providers that supposed to be used.

All options are applied right before executing. If an option name is not recognized, an exception is raised and commands are not executed.

Example

You can also setup specific options at run time. Either of two formats can be used:

1. Using the provider name in an option name;
2. Not using the provider name in an option name.

In the second case options will be applied to the current provider, namely to the provider specified in the [TUniConnection.ProviderName](#) property of assigned connection.

When you set the AutoDDL option like it is shown in the second example, you can execute the script with the InterBase provider, but attempt to execute it with other providers will fail.

```
Example 1.  
UniSQL1.SpecificOptions.Add('InterBase.AutoDDL=True')  
Example 2.  
UniSQL1.SpecificOptions.Add('AutoDDL=True')
```

See Also

- [TUniConnection.ProviderName](#)
- [Using Oracle data access provider with UniDAC in Delphi](#)

- [Using SQL Server data access provider with UniDAC in Delphi](#)
- [Using MySQL data access provider with UniDAC in Delphi](#)
- [Using InterBase data access provider with UniDAC in Delphi](#)
- [Using PostgreSQL data access provider with UniDAC in Delphi](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.2.4 Transaction Property

Used to specify the [TUniTransaction](#) object in the context of which SQL commands will be executed, and queries retrieving data will be opened.

Class

[TUniSQL](#)

Syntax

```
property Transaction: TUniTransaction stored IsTransactionStored;
```

Remarks

Use the Transaction property to specify the [TUniTransaction](#) object in the context of which SQL commands will be executed, and queries retrieving data will be opened. If this property is not specified, the default transaction associated with linked [TUniConnection](#) will be used. This transaction will work in AutoCommit mode.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.3 Methods

Methods of the **TUniSQL** class.

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

Public

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

BreakExec (inherited from TCustomDASQL)	Breaks execution of an SQL statement on the server.
CreateProcCall	Assigns a command that calls stored procedure specified by Name to the SQL property.
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 a parameter with the specified name.
MacroByName (inherited from TCustomDASQL)	Finds a macro with the specified name.
ParamByName	Searches for 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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.3.1 CreateProcCall Method

Assigns a command that calls stored procedure specified by Name to the SQL property.

Class

[TUniSQL](#)

Syntax

```
procedure CreateProcCall(const Name: string);
```

Parameters

Name

Holds the stoped procedure name.

Remarks

Call the CreateProcCall method to assign a command that calls stored procedure specified by Name to the SQL property. This procedure also retrieves information about parameters of the procedure from server. After calling CreateProcCall you can assign parameter values of the stored procedure using, for example, [TCustomDASQL.Params](#) or [ParamByName](#), and then execute it with the [TCustomDASQL.Execute](#) method.

See Also

- [TCustomDASQL.Execute](#)
- [TUniStoredProc](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.3.2 FindParam Method

Searches for a parameter with the specified name.

Class

[TUniSQL](#)

Syntax

```
function FindParam(const value: string): TUniParam;
```

Parameters

Value

Holds the name of the parameter to search.

Return Value

a parameter, if a match is found. Nil otherwise.

Remarks

Call the FindParam method to find a parameter with the name passed in Name argument. If a match is found, FindParam returns the parameter. Otherwise, it returns nil.

See Also

- [TUniParam](#)
- [ParamByName](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.14.3.3 ParamByName Method

Searches for a parameter with the specified name.

Class

[TUniSQL](#)

Syntax

```
function ParamByName(const Value: string): TUniParam;
```

Parameters

Value

Holds the name of the parameter to search.

Return Value

a parameter, if a match is found. Nil otherwise.

Remarks

Call the ParamByName method to find a parameter with the name passed as Name.

If a match is found, ParamByName returns the parameter. Otherwise, it raises an exception.

Example

```
UniSQL1.Execute;  
Edit1.Text := UniSQL1.ParamByName('Contact').AsString;
```

See Also

- [TUniParam](#)
- [FindParam](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.15 TUniStoredProc Class

A component for accessing and executing stored procedures and functions.

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

Unit

[uni](#)

Syntax

```
TUniStoredProc = class(TCustomUniDataSet);
```

Remarks

Use TUniStoredProc 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](#)

[TCustomUniDataSet](#)

TUniStoredProc

See Also

- [TUniQuery](#)
- [TUniSQL](#)

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.20.1.15.1 Members

[TUniStoredProc](#) 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.
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 TCustomUniDataSet)	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.
LastInsertId (inherited from TCustomUniDataSet)	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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 TCustomUniDataSet)	Specifies the behaviour of a TCustomUniDataSet 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 TCustomUniDataSet)	Holds the 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.
SpecificOptions (inherited from TCustomUniDataSet)	Used to provide extended settings for each data provider.
SQL (inherited from TCustomDADataset)	Used to provide a SQL statement that a query component executes when its Open method is called.

SQLDelete (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from TCustomDADataset)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from TCustomDADataset)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.
SQLUpdate (inherited from TCustomDADataset)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StoredProcName	Used to specify the name of the stored procedure to call on the server.
Transaction (inherited from TCustomUniDataset)	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomUniDataset)	Points to an update object component which provides update SQL statements or update objects for flexible data update.
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.
UpdateTransaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which update commands will be executed.

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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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 TCustomUniDataSet)	Assigns a command that calls stored procedure specified by name to the SQL property.

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 TCustomUniDataSet)	Determines if 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 (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.
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 TCustomUniDataSet)	Provides second and other result sets while executing multireult query.
ParamByName (inherited from TCustomUniDataSet)	Accesses parameter information based on a specified parameter name.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and

	parses cursor for a query.
PrepareSQL	Describes the stored procedure parameters.
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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.15.2 Properties

Properties of the **TUniStoredProc** class.

For a complete list of the **TUniStoredProc** class members, see the [TUniStoredProc 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.
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 TCustomUniDataSet)	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.
LastInsertId (inherited from TCustomUniDataSet)	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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 TCustomUniDataSet)	Specifies the behaviour of a TCustomUniDataSet 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 TCustomUniDataSet)	Holds the 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.
SpecificOptions (inherited from TCustomUniDataSet)	Used to provide extended settings for each data provider.
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.
Transaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomUniDataSet)	Points to an update object component which provides update SQL statements or update objects for flexible data update.
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.
UpdateTransaction (inherited from	Used to specify the TUniTransaction object in

TCustomUniDataSet)	the context of which update commands will be executed.
-------------------------------------	--

Published

Name	Description
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.15.2.1 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

[TUniStoredProc](#)

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

- [TUniQuery.LockMode](#)
- [TUniTable.LockMode](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.15.2.2 StoredProcName Property

Used to specify the name of the stored procedure to call on the server.

Class

[TUniStoredProc](#)

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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.15.3 Methods

Methods of the **TUniStoredProc** class.

For a complete list of the **TUniStoredProc** class members, see the [TUniStoredProc 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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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 TCustomUniDataSet)	Assigns a command that calls stored procedure specified by name to the SQL property.
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 TCustomUniDataSet)	Determines if 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 (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	Provides a list of available

TCustomDADataset)	key field names.
GetOrderBy (inherited from TCustomDADataset)	Retrieves an ORDER BY clause from a SQL statement.
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 TCustomUniDataSet)	Provides second and other result sets while executing multiresult query.
ParamByName (inherited from TCustomUniDataSet)	Accesses parameter information based on a specified parameter name.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Describes the stored procedure parameters.
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)	Cancel changes made to the current record when cached updates are enabled.

SaveSQL (inherited from TCustomDADataset)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.15.3.1 ExecProc Method

Executes a SQL statement on the server.

Class

[TUniStoredProc](#)

Syntax

```
procedure ExecProc;
```

Remarks

The ExecProc method is equal to the [TCustomDADataset.Execute](#) method. It is included for compatibility with the TStoredProc component.

See Also

- [TCustomDADataset.Execute](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.15.3.2 PrepareSQL Method

Describes the stored procedure parameters.

Class

[TUniStoredProc](#)

Syntax

```
procedure PrepareSQL(IsQuery: boolean = False);
```

Parameters*IsQuery*

If True, the SELECT statement is generated.

Remarks

Call the PrepareSQL method to describe parameters of stored procedure. The Execute method calls it automatically if it is necessary. You can define parameters at design time if ParameterEditor is open. Set the IsQuery parameter to True to prepare SELECT statement. Set it to False or omit it to prepare EXECUTE PROCEDURE statement. This parameter has sense only for InterBase server.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.16 TUniTable 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 [TUniTable](#) members.

Unit

[Uni](#)

Syntax

```
TUniTable = class(TCustomUniTable);
```

Remarks

The TUniTable component allows retrieving and updating data in a single table without writing SQL statements. Use TUniTable to access data in a table . Use the TableName property to specify table name. TUniTable 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](#)

[TCustomDADDataSet](#)

[TCustomUniDataSet](#)

[TCustomUniTable](#)**TUniTable**

See Also

- [Master/Detail Relationships](#)
- [TCustomUniDataSet](#)
- [TUniQuery](#)
- [TUniStoredProc](#)

© 1997-2025

Devart. All Rights Reserved.

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

6.20.1.16.1 Members

[TUniTable](#) 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.
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 TCustomUniDataSet)	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.
LastInsertId (inherited from TCustomUniDataSet)	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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 TCustomUniDataSet)	Specifies the behaviour of a TCustomUniDataSet 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 TCustomUniDataSet)	Holds the 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.
SpecificOptions (inherited from TCustomUniDataSet)	Used to provide extended settings for each data provider.
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.
Transaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in

	the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from TCustomUniDataSet)	Points to an update object component which provides update SQL statements or update objects for flexible data update.
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.
UpdateTransaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which update commands will be executed.

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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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 TCustomUniDataSet)	Assigns a command that calls stored procedure specified by name to the SQL property.
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 TCustomUniDataSet)	Determines if 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 (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.
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 TCustomUniDataSet)	Provides second and other result sets while executing multiresult query.
ParamByName (inherited from TCustomUniDataSet)	Accesses parameter information based on a specified parameter name.
Prepare (inherited from TCustomDADataset)	Allocates, opens, and parses cursor for a query.
PrepareSQL (inherited from TCustomUniTable)	Used to determine KeyFields and build query for TUniTable.
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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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,

TCustomDADataset)	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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.16.2 Properties

Properties of the **TUniTable** class.

For a complete list of the **TUniTable** class members, see the [TUniTable 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.
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 TCustomUniDataset)	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.
LastInsertId (inherited from TCustomUniDataset)	Can be used with MySQL, PostgreSQL, SQL Server, and Microsoft Access to get the value of the ID field after executing INSERT statement.
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 TCustomUniDataSet)	Specifies the behaviour of a TCustomUniDataSet 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 TCustomUniDataSet)	Holds the 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.
SpecificOptions (inherited from TCustomUniDataSet)	Used to provide extended settings for each data provider.
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.
Transaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.
UniDirectional (inherited from TCustomDADataset)	Used if an application does not need bidirectional access to records in the result set.

UpdateObject (inherited from TCustomUniDataSet)	Points to an update object component which provides update SQL statements or update objects for flexible data update.
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.
UpdateTransaction (inherited from TCustomUniDataSet)	Used to specify the TUniTransaction object in the context of which update commands will be executed.

Published

Name	Description
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.16.2.1 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

[TUniTable](#)

Syntax

```
property LockMode: TLockMode default ImOptimistic;
```

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 ImOptimistic.

See Also

- [TUniStoredProc.LockMode](#)
- [TUniQuery.LockMode](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.16.2.2 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class

[TUniTable](#)

Syntax

```
property orderFields: string;
```

Remarks

TUniTable uses the OrderFields property to build ORDER BY clause of SQL statements. To set several field names to this property separate them with commas.

TUniTable is reopened when OrderFields is being changed.

See Also

- [TUniTable](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.16.2.3 TableName Property

Used to specify the name of the database table this component encapsulates.

Class

[TUniTable](#)

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.

See Also

- [TUniQuery](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17 TUniTransaction Class

A component for managing transactions in an application.

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

Unit

[uni](#)

Syntax

```
TUniTransaction = class(TDATransaction);
```

Remarks

The TUniTransaction component is used to provide discrete transaction control over connection. It can be used for manipulating simple local and global transactions.

Inheritance Hierarchy

[TDATransaction](#)

TUniTransaction

See Also

- [Transactions](#)
- [TCustomDACConnection.StartTransaction](#)
- [TCustomDACConnection.Commit](#)
- [TCustomDACConnection.Rollback](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.1 Members

[TUniTransaction](#) class overview.

Properties

Name	Description
Active (inherited from TDATransaction)	Used to determine if the transaction is active.
Connections	Used to specify a connection for the given index.
ConnectionsCount	Used to get the number of connections associated with the transaction component.
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
AddConnection	Binds a TCustomDAConnection object with the transaction component.
Commit (inherited from TDATransaction)	Commits the current transaction.
CommitRetaining	Stores to the database server all changes of data associated with the transaction permanently and then retains the transaction context.
RemoveConnection	Disassociates the specified connections from the transaction.
Rollback (inherited from TDATransaction)	Discards all modifications of data associated with the current transaction and ends the transaction.
RollbackRetaining	Rolls back all data changes associated with the transaction and retains the transaction context.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.2 Properties

Properties of the **TUniTransaction** class.

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

Public

Name	Description
Active (inherited from TDATransaction)	Used to determine if the transaction is active.
Connections	Used to specify a connection for the given index.
ConnectionsCount	Used to get the number of connections associated with the transaction component.
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.2.1 Connections Property(Indexer)

Used to specify a connection for the given index.

Class

[TUniTransaction](#)

Syntax

```
property Connections[Index: integer]: TUniConnection;
```

Parameters

Index

Holds the index to specify the connection for.

Remarks

Specifies a connection for the given index.

See Also

- [ConnectionsCount](#)
- [RemoveConnection](#)
- [AddConnection](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.2.2 ConnectionsCount Property

Used to get the number of connections associated with the transaction component.

Class

[TUniTransaction](#)

Syntax

```
property ConnectionsCount: integer;
```

Remarks

Use the ConnectionsCount property for getting the number of connections associated with the transaction component.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.2.3 IsolationLevel Property

Used to specify how the transactions containing database modifications are handled.

Class

[TUniTransaction](#)

Syntax

```
property IsolationLevel: TCRIsolationLevel;
```

Remarks

Use the IsolationLevel property to specify how the transactions containing database modifications are handled.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.3 Methods

Methods of the **TUniTransaction** class.

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

Public

Name	Description
AddConnection	Binds a TCustomDACConnection object with the transaction component.
Commit (inherited from TDATransaction)	Commits the current transaction.
CommitRetaining	Stores to the database server all changes of data associated with the transaction permanently and then retains the transaction context.
RemoveConnection	Disassociates the specified connections from the transaction.
Rollback (inherited from TDATransaction)	Discards all modifications of data associated with the current transaction and ends the transaction.
RollbackRetaining	Rolls back all data changes associated with the transaction and retains the transaction context.
StartTransaction (inherited from TDATransaction)	Begins a new transaction.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.3.1 AddConnection Method

Binds a TCustomDACConnection object with the transaction component.

Class

[TUniTransaction](#)

Syntax

```
procedure AddConnection(Connection: TUniConnection);
```

Parameters

Connection

Holds a TCustomDACConnection object to associate with the transaction component.

Remarks

Use the AddConnection method to associate a TCustomDACConnection object with the transaction component.

See Also

- [RemoveConnection](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.3.2 CommitRetaining Method

Stores to the database server all changes of data associated with the transaction permanently and then retains the transaction context.

Class

[TUniTransaction](#)

Syntax

```
procedure CommitRetaining;
```

Remarks

Call the CommitRetaining method to store to the database server all changes of data associated with the transaction permanently and then retain the transaction context.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.3.3 RemoveConnection Method

Disassociates the specified connections from the transaction.

Class

[TUniTransaction](#)

Syntax

```
procedure RemoveConnection(Connection: TUniConnection);
```

Parameters

Connection

Holds the connections to disassociate.

Remarks

Call the RemoveConnection method to disassociate the specified connections from the transaction.

See Also

- [Connections](#)
- [AddConnection](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.17.3.4 RollbackRetaining Method

Rolls back all data changes associated with the transaction and retains the transaction context.

Class

[TUniTransaction](#)

Syntax

```
procedure RollbackRetaining;
```

Remarks

Call the RollbackRetaining method to roll back all changes of data associated with the transaction and retain the transaction context.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.18 TUniUpdateSQL Class

A component for tuning update operations for the DataSet component.

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

Unit

[Uni](#)

Syntax

```
TUniUpdateSQL = class(TCustomDAUpdateSQL);
```

Remarks

Use the TUniUpdateSQL component to provide DML statements for the dataset components that return read-only result set. This component also allows setting objects that can be used for executing update operations. You may prefer to use directly SQLInsert, SQLUpdate, and SQLDelete properties of the [TCustomDADataSet](#) descendants.

Inheritance Hierarchy

[TCustomDAUpdateSQL](#)

TUniUpdateSQL

See Also

- [TCustomUniDataSet.UpdateObject](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.1.18.1 Members

[TUniUpdateSQL](#) 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.

Reserved.

6.20.2 Constants

Constants in the **Uni** unit.

Constants

Name	Description
UniDACVersion	Read this constant to get current version number for UniDAC.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.20.2.1 UniDACVersion Constant

Read this constant to get current version number for UniDAC.

Unit

[uni](#)

Syntax

```
UniDACVersion = '10.3.2';
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.21 UniAlerter

This unit contains the implementation of the TUniAlerter component.

Classes

Name	Description
TUniAlerter	A component for sending and receiving database events.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.21.1 Classes

Classes in the **UniAlerter** unit.

Classes

Name	Description
TUniAlerter	A component for sending and receiving database events.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.21.1.1 TUniAlerter Class

A component for sending and receiving database events.

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

Unit

[uniAlerter](#)

Syntax

```
TUniAlerter = class(TDAALerter);
```

Remarks

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

Events are transaction-based. This means that the waiting connection does not get event until the transaction posting the event commits.

Note: not all DBMS supports event notification. Currently TUniAlerter can be used with Oracle, PostgreSQL, and InterBase(Firebird).

TUniAlerter uses the following DBMS-specific features to send and receive events:

Oracle: *DBMS_ALERT* package;

PostgreSQL: *NOTIFY* and *LISTEN* commands;

InterBase: *POST_EVENT* command;

Inheritance Hierarchy

[TDAAlerter](#)

TUniAlerter

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.21.1.1.1 Members

[TUniAlerter](#) 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	Used to specify the connection for TUniAlerter.

Methods

Name	Description
SendEvent (inherited from TDAAlerter)	Sends an event with Name and content Message.
Start (inherited from TDAAlerter)	Starts waiting process.
Stop (inherited from TDAAlerter)	Stops waiting process.

Events

Name	Description
OnError (inherited from TDAAlert)	Occurs if an exception occurs in waiting process

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.21.1.1.2 Properties

Properties of the **TUniAlert** class.

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

Public

Name	Description
Active (inherited from TDAAlert)	Used to determine if TDAAlert waits for messages.
AutoRegister (inherited from TDAAlert)	Used to automatically register events whenever connection opens.

Published

Name	Description
Connection	Used to specify the connection for TUniAlert.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.21.1.1.2.1 Connection Property

Used to specify the connection for TUniAlert.

Class

[TUniAlerter](#)

Syntax

```
property Connection: TUniConnection;
```

Remarks

Use the Connection property to specify the connection for TUniAlerter.

See Also

- [TUniConnection](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.22 UniDacVcl

This unit contains the visual constituent of UniDAC.

Classes

Name	Description
TUniConnectDialog	A class that provides a dialog box for user to supply his login information.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.22.1 Classes

Classes in the **UniDacVcl** unit.

Classes

Name	Description
TUniConnectDialog	A class that provides a dialog box for user to supply his login information.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.22.1.1 TUniConnectDialog Class

A class that provides a dialog box for user to supply his login information.

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

Unit

[UniDacvc1](#)

Syntax

```
TUniConnectDialog = class(TCustomConnectDialog);
```

Remarks

The TUniConnectDialog component is a direct descendant of TCustomConnectDialog class. Use TUniConnectDialog to provide dialog box for user to supply provider name, server name, database, user name, port number, and password. You may want to customize appearance of dialog box using this class's properties.

Inheritance Hierarchy

[TCustomConnectDialog](#)

TUniConnectDialog

See Also

- [TCustomDACConnection.ConnectDialog](#)

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.22.1.1.1 Members

[TUniConnectDialog](#) 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	Points to the associated TUniConnection object.
DatabaseLabel	Used to specify a prompt for database name 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 number edit.
ProviderLabel	Used to specify a prompt for provider name.
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.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.22.1.1.2 Properties

Properties of the **TUniConnectDialog** class.

For a complete list of the **TUniConnectDialog** class members, see the [TUniConnectDialog 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	Points to the associated TUniConnection object.
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.
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.

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.

Published

Name	Description
DatabaseLabel	Used to specify a prompt for database name edit.
PortLabel	Used to specify a prompt for port number edit.
ProviderLabel	Used to specify a prompt for provider name.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.22.1.1.2.1 Connection Property

Points to the associated TUniConnection object.

Class

[TUniConnectDialog](#)

Syntax

```
property Connection: TUniConnection;
```

Remarks

The Connection property points to the associated TUniConnection object. This property is

read only.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.22.1.1.2.2 DatabaseLabel Property

Used to specify a prompt for database name edit.

Class

[TUniConnectDialog](#)

Syntax

```
property DatabaseLabel: string;
```

Remarks

Use the DatabaseLabel property to specify a prompt for database name edit.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.22.1.1.2.3 PortLabel Property

Used to specify a prompt for port number edit.

Class

[TUniConnectDialog](#)

Syntax

```
property PortLabel: string;
```

Remarks

Use the PortLabel property to specify a prompt for port number edit.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.22.1.1.2.4 ProviderLabel Property

Used to specify a prompt for provider name.

Class

[TUniConnectDialog](#)

Syntax

```
property ProviderLabel: string;
```

Remarks

Use the ProviderLabel property to specify a prompt for provider name.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.23 UniDump

This unit contains the implementation of the TUniDump component.

Classes

Name	Description
TUniDump	The class that serves for storing data from tables or editable views as a script and for restoring data from a received script.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.23.1 Classes

Classes in the **UniDump** unit.

Classes

Name	Description
TUniDump	The class that serves for storing data from tables or

	editable views as a script and for restoring data from a received script.
--	---

© 1997-2025

Devart. All Rights Reserved.

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

6.23.1.1 TUniDump Class

The class that serves for storing data from tables or editable views as a script and for restoring data from a received script.

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

Unit

[UniDump](#)

Syntax

```
TUniDump = class(TDADump);
```

Remarks

TUniDump serves to store data from tables or editable views as a script and to restore data from a received script.

Use the [TDADump.TableNames](#) property to specify the list of objects to be stored. To launch a generating script, call the [TDADump.Backup](#) method.

TUniDump also can generate scripts for a query. Just call the [TDADump.BackupQuery](#) method and pass a query statement into it. The object list assigned to the TableNames property is ignored if you call [TDADump.BackupQuery](#). The generated script can be viewed in the [TDADump.SQL](#) property.

TUniDump works on the client side. It causes large network loading.

Inheritance Hierarchy

[TDADump](#)

TUniDump

© 1997-2025

Devart. All Rights

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

Reserved.

6.23.1.1.1 Members

[TUniDump](#) 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.
Options (inherited from TDADump)	Used to specify the behaviour of a TDADump component.
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 , <code>M:Devart.Dac.TDADump.BackupToFile(System.String)</code> or <code>M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream)</code> method execution progress.
OnError (inherited from TDADump)	Occurs when server raises some error on TDADump.Restore .
OnRestoreProgress (inherited from TDADump)	Occurs to indicate the TDADump.Restore , TDADump.RestoreFromFile , or TDADump.RestoreFromStream method execution progress.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.24 UniLoader

This unit contains the implementation of the TUniLoader component.

Classes

Name	Description
TUniLoader	TUniLoader allows to load external data into a database table.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.24.1 Classes

Classes in the **UniLoader** unit.

Classes

Name	Description
TUniLoader	TUniLoader allows to load external data into a database table.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.24.1.1 TUniLoader Class

TUniLoader allows to load external data into a database table.

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

Unit

[UniLoader](#)

Syntax

```
TUniLoader = class(TDALoader);
```

Remarks

TUniLoader serves for fast loading of data to the database. To specify the name of the loading table set the TableName property. Use the Columns property to access individual columns. Write OnGetColumnData or OnPutData event handlers to read external data and pass it to the database. Call the Load method to start loading data.

For each type of database server TUniLoader uses its specific interfaces for loading with maximum speed.

For Oracle the Direct Path Load interface is used.

For SQL Server loading is based on the memory-based bulk-copy operations using the IRowsetFastLoad interface. Data loading is performed without transactions.

For PostgreSQL data are loaded using the COPY command.

For MySQL, InterBase, and Firebird loading uses INSERT SQL statements. In this case several rows are combined in one statement if possible. In Firebird 2.0 and higher INSERT statements are combined in one EXECUTE BLOCK statement.

Inheritance Hierarchy

[TDALoader](#)

TUniLoader

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.24.1.1.1 Members

[TUniLoader](#) class overview.

Properties

Name	Description
Columns (inherited from TDALoader)	Used to add a TDAColumn object for each field that will be loaded.
Connection (inherited from TDALoader)	property. Used to specify TCustomDAConnection in which TDALoader will be executed.
TableName (inherited from TDALoader)	Used to specify the name of the table to which data will be loaded.

Methods

Name	Description
CreateColumns (inherited from TDALoader)	Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName .
Load (inherited from TDALoader)	Starts loading data.
LoadFromDataSet (inherited from TDALoader)	Loads data from the specified dataset.
PutColumnData (inherited from TDALoader)	Overloaded. Puts the value of individual columns.

Events

Name	Description
OnGetColumnData (inherited from TDALoader)	Occurs when it is needed to put column values.
OnProgress (inherited from TDALoader)	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.
OnPutData (inherited from TDALoader)	Occurs when putting loading data by rows is needed.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.25 UniProvider

This unit contains the TUniProvider class for linking the server-specific providers to application.

Classes

Name	Description
TUniProvider	A base class components that are intended to link the server-specific providers to application.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.25.1 Classes

Classes in the **UniProvider** unit.

Classes

Name	Description
TUniProvider	A base class components that are intended to link the server-specific providers to application.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.25.1.1 TUniProvider Class

A base class components that are intended to link the server-specific providers to application.

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

Unit

[UniProvider](#)

Syntax

```
TUniProvider = class(TComponent);
```

Remarks

TUniProvider is a base class for components that are intended to link the server-specific providers to application.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.25.1.1.1 Members

[TUniProvider](#) class overview.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.26 UniScript

This unit contains the implementation of the TUniScript component.

Classes

Name	Description
TUniScript	A component for executing several SQL statements one by one.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.26.1 Classes

Classes in the **UniScript** unit.

Classes

Name	Description
TUniScript	A component for executing several SQL statements one by one.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.26.1.1 TUniScript Class

A component for executing several SQL statements one by one.

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

Unit

[UniScript](#)

Syntax

```
TUniScript = class(TDAscript);
```

Remarks

Often it is necessary to execute several SQL statements one by one. Known way is using a lot of components such as [TUniSQL](#). Usually it is not a good solution. With only one TUniScript component you can execute several SQL statements as one. This sequence of statements is named script. To separate single statements use semicolon (;), slash (/), and for PL/SQL in Oracle - only slash, also keyword 'GO' for SQL Server and DELIMITER for MySQL server. Note that slash must be the first character in line.

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

Inheritance Hierarchy

[TDA Script](#)

TUniScript

See Also

- [TUniSQL](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.26.1.1.1 Members

[TUniScript](#) class overview.

Properties

Name	Description
Connection	Used to specify the connection in which the script will be executed.
DataSet	Used to retrieve the results of SELECT statements execution inside a script.
Debug (inherited from TDA Script)	Used to display the script execution and all its parameter values.
Delimiter (inherited from TDA Script)	Used to set the delimiter string that separates script statements.
EndLine (inherited from TDA Script)	Used to get the current statement last line number in a script.
EndOffset (inherited from TDA Script)	Used to get the offset in the last line of the current statement.
EndPos (inherited from TDA Script)	Used to get the end position of the current statement.
Macros (inherited from TDA Script)	Used to change SQL script text in design- or run-time easily.
SpecificOptions	Provides extended settings

	for each data provider.
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.
Transaction	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.

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 server raises an error.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.26.1.1.2 Properties

Properties of the **TUniScript** class.

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

Public

Name	Description
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.
DataSet	Used to retrieve the results of SELECT statements execution inside a script.
Debug (inherited from TDA Script)	Used to display the script execution and all its parameter values.
Delimiter (inherited from TDA Script)	Used to set the delimiter string that separates script statements.
Macros (inherited from TDA Script)	Used to change SQL script text in design- or run-time easily.
SpecificOptions	Provides extended settings for each data provider.
SQL (inherited from TDA Script)	Used to get or set script text.
Transaction	Used to specify the TUniTransaction object in the context of which SQL commands will be executed, and queries retrieving data will be opened.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.26.1.1.2.1 Connection Property

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

Class

[TUniScript](#)

Syntax

```
property Connection: TUniConnection;
```

Remarks

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

See Also

- [TUniConnection](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.26.1.1.2.2 DataSet Property

Used to retrieve the results of SELECT statements execution inside a script.

Class

[TUniScript](#)

Syntax

```
property DataSet: TCustomUniDataSet;
```

Remarks

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

See Also

- [TDAScript.Execute](#)

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.26.1.1.2.3 SpecificOptions Property

Provides extended settings for each data provider.

Class

[TUniScript](#)

Syntax

```
property SpecificOptions: TStrings;
```

Remarks

Use the SpecificOptions property to provide extended settings for each data provider. SpecificOptions can be setup both design time and run time.

At design time call the component editor by double click on it, and select the Options tab in the editor. Calling the SpecificOptions editor from the Object Inspector will open the component editor with Options tab active. Type or select the provider name, and change values of required properties. Then you can either close the editor, or select another provider name. Settings for all providers will be saved.

SpecificOptions can be setup at the same time for all providers that supposed to be used.

All options are applied right before executing. If an option name is not recognized, an exception is raised and commands are not executed.

Example

You can also setup specific options at run time. Either of two formats can be used:

1. Using the provider name in an option name;
2. Not using the provider name in an option name

In the second case options will be applied to the current provider, namely to the provider specified in the [TUniConnection.ProviderName](#) property of assigned connection.

When you set the AutoDDL option like it is shown in the second example, you can execute the script with the InterBase provider, but attempt to execute it with other providers will fail.

```
Example 1.  
UniScript1.SpecificOptions.Add('InterBase.AutoDDL=True')  
Example 2.  
UniScript1.SpecificOptions.Add('AutoDDL=True')
```

See Also

- [TUniConnection.ProviderName](#)
- [Using Oracle data access provider with UniDAC in Delphi](#)
- [Using SQL Server data access provider with UniDAC in Delphi](#)
- [Using MySQL data access provider with UniDAC in Delphi](#)
- [Using InterBase data access provider with UniDAC in Delphi](#)
- [Using PostgreSQL data access provider with UniDAC in Delphi](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.26.1.1.2.4 Transaction Property

Used to specify the [TUniTransaction](#) object in the context of which SQL commands will be executed, and queries retrieving data will be opened.

Class

[TUniScript](#)

Syntax

```
property Transaction: TUniTransaction stored IsTransactionStored;
```

Remarks

Use the Transaction property to specify the [TUniTransaction](#) object in the context of which SQL commands will be executed, and queries retrieving data will be opened. If this property is not specified, the default transaction associated with linked [TUniConnection](#) will be used. This transaction will work in AutoCommit mode.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.27 UniSQLMonitor

This unit contains the implementation of the TUniSQLMonitor component.

Classes

Name	Description
TUniSQLMonitor	This component serves for monitoring dynamic SQL execution in UniDAC-based applications.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.27.1 Classes

Classes in the **UniSQLMonitor** unit.

Classes

Name	Description
TUniSQLMonitor	This component serves for monitoring dynamic SQL execution in UniDAC-based applications.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.27.1.1 TUniSQLMonitor Class

This component serves for monitoring dynamic SQL execution in UniDAC-based applications.

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

Unit

[UniSQLMonitor](#)

Syntax

```
TUniSQLMonitor = class(TCustomDASQLMonitor);
```

Remarks

Use TUniSQLMonitor to monitor dynamic SQL execution in UniDAC-based applications. TUniSQLMonitor 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 TUniSQLMonitor object. If an application has no TUniSQLMonitor instance, the Debug window is available to display SQL statements to be sent.

Inheritance Hierarchy

[TCustomDASQLMonitor](#)

TUniSQLMonitor

See Also

- [TCustomDADataset.Debug](#)
- [TCustomDASQL.Debug](#)
- [DBMonitor](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.27.1.1.1 Members

[TUniSQLMonitor](#) 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28 VirtualDataSet

6.28.1 Classes

Classes in the **VirtualDataSet** unit.

Classes

Name	Description
TCustomVirtualDataSet	A base class for representation of arbitrary data in tabular form.
TVirtualDataSet	A class for representation of arbitrary data in tabular form.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.1 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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.1.1 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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.2 TVirtualDataSet Class

A class for representation of arbitrary data in tabular form.

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

Unit

virtualDataSet

Syntax

```
TVirtualDataSet = class(TCustomVirtualDataSet);
```

Remarks

TVirtualDataSet is inherited from the TMemDataSet component. TVirtualDataSet does not store data in memory and interacts with data using event handlers. It can be used to represent of arbitrary data (arrays, lists etc.) in tabular form, for example, to visualize the data using data-aware controls.

Immediately after creating, a virtual dataset will be empty. Then you define new fields at design time using Fields Editor, so that the virtual dataset object becomes initialized and ready to be opened. When opening, the virtual dataset queries the number of rows, using a [TVirtualDataSet.OnGetRecordCount](#) event handler. Then the virtual dataset asks for the value of each field, using a [TVirtualDataSet.OnGetFieldValue](#) event handler.

When modifying records, the virtual dataset projects corresponding changes via

[TVirtualDataSet.OnInsertRecord](#), [TVirtualDataSet.OnModifyRecord](#) and [TVirtualDataSet.OnDeleteRecord](#) events.

When you close the virtual dataset it will discard its record set.

Note: TVirtualDataSet component is added to the Data Access page of the component palette, not to the VirtualDAC page.

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomVirtualDataSet](#)

TVirtualDataSet

See Also

- [TVirtualDataSet.OnGetRecordCount](#)
- [TVirtualDataSet.OnGetFieldValue](#)
- [TVirtualDataSet.OnInsertRecord](#)
- [TVirtualDataSet.OnModifyRecord](#)
- [TVirtualDataSet.OnDeleteRecord](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.2.1 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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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
OnDeleteRecord	Occurs when a record is deleted from the virtual data set.
OnGetFieldValue	Occurs when a virtual dataset asks for the value of a field.
OnGetRecordCount	E:\Devart.VirtualDac.TVirtualDataSet.OnGetRecordCount
OnInsertRecord	Occurs when a new record is added to the virtual data set.
OnModifyRecord	Occurs when a record is modified in the virtual data set.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.2.2 Events

Events of the **TVirtualDataSet** class.

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

Public

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.

Published

Name	Description
OnDeleteRecord	Occurs when a record is deleted from the virtual data set.
OnGetFieldValue	Occurs when a virtual dataset asks for the value of a field.
OnGetRecordCount	E:Devart.VirtualDac.TVirtualDataSet.OnGetRecordCount
OnInsertRecord	Occurs when a new record is added to the virtual data set.
OnModifyRecord	Occurs when a record is modified in the virtual data set.

See Also

- [TVirtualDataSet Class](#)

- [TVirtualDataSet Class Members](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.2.2.1 OnDeleteRecord Event

Occurs when a record is deleted from the virtual data set.

Class

[TVirtualDataSet](#)

Syntax

```
property OnDeleteRecord: TOnDeleteRecordEvent;
```

Remarks

Write the OnDeleteRecord event handler to process deletion of a record from the virtual dataset. When firing the event, the virtual dataset sends the record number of the record being deleted into the event handler.

See Also

- [OnInsertRecord](#)
- [OnModifyRecord](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.2.2.2 OnGetFieldValue Event

Occurs when a virtual dataset asks for the value of a field.

Class

[TVirtualDataSet](#)

Syntax

```
property OnGetFieldValue: TOnGetFieldValueEvent;
```

Remarks

Write the `OnGetFieldValue` event handler to return field data to the virtual dataset. Since virtual dataset does not store a data, when opening it fires the event to obtain the value of each field.

See Also

- [OnGetRecordCount](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.2.2.3 OnGetRecordCount Event

E:Devart.VirtualDac.TVirtualDataSet.OnGetRecordCount

Class

[TVirtualDataSet](#)

Syntax

```
property OnGetRecordCount: TOnGetRecordCountEvent;
```

Remarks

Write the `OnGetRecordCount` event handler to let the virtual dataset know, how many records it contains. Since virtual dataset does not store a data, when opening it fires the event to determine how many records will be kept in it. Then the virtual dataset will ask for the value of each field, using a [OnGetFieldValue](#) event handler.

See Also

- [OnGetFieldValue](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.2.2.4 OnInsertRecord Event

Occurs when a new record is added to the virtual data set.

Class

[TVirtualDataSet](#)

Syntax

```
property OnInsertRecord: TOnModifyRecordEvent;
```

Remarks

Write the OnInsertRecord event handler to process insertion of new record in the virtual dataset. When firing the event, the virtual dataset is positioned on the record being inserted.

See Also

- [OnModifyRecord](#)
- [OnDeleteRecord](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.1.2.2.5 OnModifyRecord Event

Occurs when a record is modified in the virtual data set.

Class

[TVirtualDataSet](#)

Syntax

```
property OnModifyRecord: TOnModifyRecordEvent;
```

Remarks

Write the OnModifyRecord event handler to process modification of a record in the virtual dataset. When firing the event, the virtual dataset is positioned on the record being modified.

See Also

- [OnInsertRecord](#)
- [OnDeleteRecord](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.2 Types

Types in the **VirtualDataSet** unit.

Types

Name	Description
TOnDeleteRecordEvent	This type is used for the TVirtualDataSet.OnDeleteRecord event.
TOnGetFieldValueEvent	This type is used for the TVirtualDataSet.OnGetFieldValue event.
TOnGetRecordCountEvent	This type is used for the TVirtualDataSet.OnGetRecordCount event.
TOnModifyRecordEvent	This type is used for TVirtualDataSet.OnInsertRecord and TVirtualDataSet.OnModifyRecord events.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.2.1 TOnDeleteRecordEvent Procedure Reference

This type is used for the [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.

© 1997-2025

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Devart. All Rights Reserved.

6.28.2.2 TOnGetFieldValueEvent Procedure Reference

This type is used for the [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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.2.3 TOnGetRecordCountEvent Procedure Reference

This type is used for the [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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.28.2.4 TOnModifyRecordEvent Procedure Reference

This type is used for [TVirtualDataSet.OnInsertRecord](#) and [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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29 VirtualQuery

6.29.1 Classes

Classes in the **VirtualQuery** unit.

Classes

Name	Description
TCustomVirtualQuery	A base class that implements TVirtualQuery functionality.
TDataSetLink	Used to link a TDataSet descendant as a data

	source for querying data in TVirtualQuery .
TDataSetLinks	This type is used for the TCustomVirtualQuery.SourceDataSets property.
TVirtualCollationManager	Used to register user-defined collations.
TVirtualFunctionManager	Used to register user-defined functions.
TVirtualQuery	Used to retrieve data simultaneously from various RDBMS'es.
TVirtualQueryOptions	Used to set up the behaviour of the TVirtualQuery class.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.1 TCustomVirtualQuery Class

A base class that implements [TVirtualQuery](#) functionality.

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

Unit

virtualQuery

Syntax

```
TCustomVirtualQuery = class(TCustomDADataset);
```

Inheritance Hierarchy

[TMemDataSet](#)

[TCustomDADataset](#)

TCustomVirtualQuery

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.1.1 Members

[TCustomVirtualQuery](#) 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.
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.
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.
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 TVirtualQuery 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 TCustomDADataset)	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 (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.
SourceDataSets	Contains a collection of source datasets for querying data.
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.
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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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.
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 TCustomDADataset)	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 (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.
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.
ParamByName (inherited from TCustomDADataset)	Sets or uses parameter information for a specific parameter based on its 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.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.
OnRegisterCollations	Occurs when the connection is opened to register the user-defined collation used in the query text.
OnRegisterFunctions	Occurs when the query is opened to register the user-defined functions used in the query text.
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.
--	---

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.1.2 Properties

Properties of the **TCustomVirtualQuery** class.

For a complete list of the **TCustomVirtualQuery** class members, see the [TCustomVirtualQuery 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.
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.

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.
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 TVirtualQuery 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 TCustomDADataset)	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 (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.
SourceDataSets	Contains a collection of source datasets for querying data.
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.
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.1.2.1 Options Property

Used to specify the behaviour of TVirtualQuery object.

Class

[TCustomVirtualQuery](#)

Syntax

```
property Options: TVirtualQueryOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TVirtualQuery object.

See Also

- [TVirtualQuery](#)

© 1997-2025

Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.1.2.2 SourceDataSets Property

Contains a collection of source datasets for querying data.

Class

[TCustomVirtualQuery](#)

Syntax

```
property SourceDataSets: TDataSetLinks;
```

Remarks

Use the property to create a list of the data sources to which the SQL statement will be executed. Each data source has to be a TDataSet descendant, connected to a database and opened prior to SQL statement execution in the TVirtualQuery (if [TVirtualQueryOptions.AutoOpenSources](#) option is set to False). Each data source gets its own "schema name" and "table name" which are used to identify the data source in the SQL statement. Each data source must have a unique combination of schema name and table name.

See Also

- [TVirtualQueryOptions.AutoOpenSources](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.1.3 Events

Events of the **TCustomVirtualQuery** class.

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

Public

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.
OnRegisterCollations	Occurs when the connection is opened to register the user-defined collation used in the query text.
OnRegisterFunctions	Occurs when the query is opened to register the user-defined functions used in the query text.
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.1.3.1 OnRegisterCollations Event

Occurs when the connection is opened to register the user-defined collation used in the query text.

Class

[TCustomVirtualQuery](#)

Syntax

```
property OnRegisterCollations: TRegisterCollationsEvent;
```

Remarks

Occurs after a component has executed a query to a database.

See Also

- [TCustomDADataset.Execute](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.1.3.2 OnRegisterFunctions Event

Occurs when the query is opened to register the user-defined functions used in the query text.

Class

[TCustomVirtualQuery](#)

Syntax

```
property OnRegisterFunctions: TRegisterFunctionsEvent;
```

Remarks

The event occurs before a component has executed a query.

See Also

- [TCustomDADataset.Execute](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.2 TDataSetLink Class

Used to link a TDataSet descendant as a data source for querying data in [TVirtualQuery](#).

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

Unit

virtualQuery

Syntax

```
TDataSetLink = class(TCollectionItem);
```

Remarks

Add a TDataSetLink instance to the [TCustomVirtualQuery.SourceDataSets](#) collection using the [TDataSetLinks.Add](#) method to link a TDataSet descendant as a data source for querying data in [TVirtualQuery](#).

See Also

- [TVirtualQuery](#)
- [TCustomVirtualQuery.SourceDataSets](#)
- [TDataSetLinks](#)

- [TDataSetLinks.Add](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.2.1 Members

[TDataSetLink](#) class overview.

Properties

Name	Description
DataSet	Defines a TDataSet descendant to be linked as a data source for querying data in TVirtualQuery .
SchemaName	Defines the schema name which will be used to identify the linked source dataset in a SQL statement.
TableName	Defines the table name which will be used to identify the linked source dataset in a SQL statement.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.2.2 Properties

Properties of the **TDataSetLink** class.

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

Published

Name	Description
DataSet	Defines a TDataSet descendant to be linked as a data source for querying data in TVirtualQuery .

SchemaName	Defines the schema name which will be used to identify the linked source dataset in a SQL statement.
TableName	Defines the table name which will be used to identify the linked source dataset in a SQL statement.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.2.2.1 DataSet Property

Defines a TDataSet descendant to be linked as a data source for querying data in [TVirtualQuery](#).

Class

[TDataSetLink](#)

Syntax

```
property DataSet: TDataSet;
```

See Also

- [TVirtualQuery](#)
- [SchemaName](#)
- [TableName](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.2.2.2 SchemaName Property

Defines the schema name which will be used to identify the linked source dataset in a SQL statement.

Class

[TDataSetLink](#)

Syntax

```
property SchemaName: string;
```

Remarks

Can be left empty. In this case either no schema name or the "main" schema name can be used when referring to the linked source dataset in a SQL statement.

Combination of schema name and table name must be unique for each linked dataset.

See Also

- [DataSet](#)
- [TableName](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.2.2.3 TableName Property

Defines the table name which will be used to identify the linked source dataset in a SQL statement.

Class

[TDataSetLink](#)

Syntax

```
property TableName: string stored GetTableNameStored;
```

Remarks

Must be filled.

Combination of schema name and table name must be unique for each linked dataset.

See Also

- [DataSet](#)
- [SchemaName](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.3 TDataSetLinks Class

This type is used for the [TCustomVirtualQuery.SourceDataSets](#) property.

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

Unit

virtualQuery

Syntax

```
TDataSetLinks = class(TCollection);
```

Remarks

TDataSetLinks is the TCollection descendant which contains a collection of the [TDataSetLink](#) instances, each of which links a TDataSet descendant as a data source for querying data in [TVirtualQuery](#).

See Also

- [TVirtualQuery](#)
- [TDataSetLink](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.3.1 Members

[TDataSetLinks](#) class overview.

Methods

Name	Description
Add	Overloaded. Adds a new TDataSetLink instance to the collection.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.3.2 Methods

Methods of the **TDataSetLinks** class.

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

Public

Name	Description
Add	Overloaded. Adds a new TDataSetLink instance to the collection.

See Also

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

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.3.2.1 Add Method

Adds a new [TDataSetLink](#) instance to the collection.

Class

[TDataSetLinks](#)

Overload List

Name	Description
Add	Adds a new TDataSetLink instance to the

	collection.
Add(DataSet: TDataSet; const SchemaName: string; const TableName: string)	Adds a new TDataSetLink instance to the collection and fills its properties.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Adds a new [TDataSetLink](#) instance to the collection.

Class

[TDataSetLinks](#)

Syntax

```
function Add: TDataSetLink; overload;
```

Return Value

A instance which has been added.

Remarks

Fill the [TDataSetLink.DataSet](#) property of the returned TDataSetLink instance to link a TDataSet descendant as a data source for querying data in [TVirtualQuery](#). Fill [TDataSetLink.SchemaName](#) and [TDataSetLink.TableName](#) properties to identify the source dataset in a SQL statement. Combination of schema name and table name must be unique for each linked dataset. Also, a source dataset can be linked using the [Add](#) method.

See Also

- [TVirtualQuery](#)
- [TDataSetLink](#)
- [TDataSetLink.DataSet](#)
- [TDataSetLink.SchemaName](#)
- [TDataSetLink.TableName](#)
- [Add](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

Adds a new [TDataSetLink](#) instance to the collection and fills its properties.

Class

[TDataSetLinks](#)

Syntax

```
function Add(DataSet: TDataSet; const SchemaName: string; const
TableName: string): TDataSetLink; overload;
```

Parameters

DataSet

Defines a TDataSet descendant to be linked as a data source for querying data in [TVirtualQuery](#).

SchemaName

Defines the schema name which will be used to identify the linked source dataset in a SQL statement. Can be left empty. In this case either no schema name or the "main" schema name can be used when referring to the dataset in a SQL statement.

TableName

Defines the table name which will be used to identify the linked source dataset in a SQL statement. Must be filled.

Return Value

A instance which has been added.

Remarks

Combination of schema name and table name must be unique for each linked dataset. Also, a source dataset can be linked using the [TDataSetLinks.Add](#) method.

See Also

- [TVirtualQuery](#)
- [TDataSetLink](#)
- [TDataSetLink.DataSet](#)
- [TDataSetLink.SchemaName](#)
- [TDataSetLink.TableName](#)
- [TDataSetLinks.Add](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4 TVirtualCollationManager Class

Used to register user-defined collations.

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

Unit

virtualQuery

Syntax

```
TVirtualCollationManager = class(System.TObject);
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4.1 Members

[TVirtualCollationManager](#) class overview.

Methods

Name	Description
RegisterAnsiCollation	Overloaded. Used to register a user-defined non-Unicode collation.
RegisterCollation	Overloaded. Used to register a user-defined collation.
RegisterDefaultCollations	Used to register a user-defined default collation.
RegisterWideCollation	Overloaded. Used to register a user-defined Unicode collation.
UnRegisterAnsiCollation	Used to unregister a user-defined non-Unicode collation.
UnRegisterCollation	Used to unregister a user-defined collation.

UnRegisterDefaultCollations	Used to unregister a user-defined default collation.
UnRegisterWideCollation	Used to unregister a user-defined Unicode collation.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4.2 Methods

Methods of the **TVirtualCollationManager** class.

For a complete list of the **TVirtualCollationManager** class members, see the

[TVirtualCollationManager Members](#) topic.

Public

Name	Description
RegisterAnsiCollation	Overloaded. Used to register a user-defined non-Unicode collation.
RegisterCollation	Overloaded. Used to register a user-defined collation.
RegisterDefaultCollations	Used to register a user-defined default collation.
RegisterWideCollation	Overloaded. Used to register a user-defined Unicode collation.
UnRegisterAnsiCollation	Used to unregister a user-defined non-Unicode collation.
UnRegisterCollation	Used to unregister a user-defined collation.
UnRegisterDefaultCollations	Used to unregister a user-defined default collation.
UnRegisterWideCollation	Used to unregister a user-defined Unicode collation.

See Also

- [TVirtualCollationManager Class](#)

- [TVirtualCollationManager Class Members](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4.2.1 RegisterAnsiCollation Method

Used to register a user-defined non-Unicode collation.

Class

[TVirtualCollationManager](#)

Overload List

Name	Description
RegisterAnsiCollation(const Name: string; VirtualAnsiCollation: TVirtualAnsiCollation)	Used to specify a user-defined non-Unicode collation that can be applied in an SQL statement.
RegisterAnsiCollation(const Name: string; VirtualAnsiCollation: TVirtualAnsiCollationMethod)	Used to specify a user-defined non-Unicode collation that can be applied in an SQL statement.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Used to specify a user-defined non-Unicode collation that can be applied in an SQL statement.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure RegisterAnsiCollation(const Name: string;  
VirtualAnsiCollation: TVirtualAnsiCollation); overload;
```

Parameters

Name

User-defined collation name.

VirtualAnsiCollation

User-defined non-Unicode collation.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Used to specify a user-defined non-Unicode collation that can be applied in an SQL statement.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure RegisterAnsiCollation(const Name: string;
VirtualAnsiCollation: TVirtualAnsiCollationMethod); overload;
```

Parameters

Name

User-defined collation name.

VirtualAnsiCollation

User-defined non-Unicode collation.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4.2.2 RegisterCollation Method

Used to register a user-defined collation.

Class

[TVirtualCollationManager](#)

Overload List

Name	Description
RegisterCollation(const Name: string; VirtualCollation: TVirtualCollation)	Used to specify a user-defined collation that can be applied in an SQL statement.
RegisterCollation(const Name: string; VirtualCollation: TVirtualCollationMethod)	Used to specify a user-defined collation that can be applied in an SQL statement.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Used to specify a user-defined collation that can be applied in an SQL statement.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure RegisterCollation(const Name: string; virtualCollation: TVirtualCollation); overload;
```

Parameters

Name

User-defined collation name.

VirtualCollation

User-defined collation.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Used to specify a user-defined collation that can be applied in an SQL statement.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure RegisterCollation(const Name: string; virtualCollation: TVirtualCollationMethod); overload;
```

Parameters

Name

User-defined collation name.

VirtualCollation

User-defined collation.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4.2.3 RegisterDefaultCollations Method

Used to register a user-defined default collation.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure RegisterDefaultCollations;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4.2.4 RegisterWideCollation Method

Used to register a user-defined Unicode collation.

Class

[TVirtualCollationManager](#)

Overload List

Name	Description
RegisterWideCollation(const Name: string; VirtualWideCollation: TVirtualWideCollation)	Used to specify a user-defined Unicode collation that can be applied in an SQL statement.
RegisterWideCollation(const Name: string; VirtualWideCollation: TVirtualWideCollationMethod)	Used to specify a user-defined Unicode collation that can be applied in an SQL statement.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Used to specify a user-defined Unicode collation that can be applied in an SQL statement.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure RegisterWideCollation(const Name: string;  
VirtualWideCollation: TVirtualWideCollation); overload;
```

Parameters

Name

User-defined collation name.

VirtualWideCollation

User-defined Unicode collation.

© 1997-2025

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

Used to specify a user-defined Unicode collation that can be applied in an SQL statement.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure RegisterWideCollation(const Name: string;  
VirtualWideCollation: TVirtualWideCollationMethod); overload;
```

Parameters

Name

User-defined collation name.

VirtualWideCollation

User-defined Unicode collation.

© 1997-2025

Devart. All Rights

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.29.1.4.2.5 UnRegisterAnsiCollation Method

Used to unregister a user-defined non-Unicode collation.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure UnRegisterAnsiCollation(const Name: string);
```

Parameters*Name*

User-defined collation name.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4.2.6 UnRegisterCollation Method

Used to unregister a user-defined collation.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure UnRegisterCollation(const Name: string);
```

Parameters*Name*

User-defined collation name.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4.2.7 UnRegisterDefaultCollations Method

Used to unregister a user-defined default collation.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure UnRegisterDefaultCollations;
```

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.4.2.8 UnRegisterWideCollation Method

Used to unregister a user-defined Unicode collation.

Class

[TVirtualCollationManager](#)

Syntax

```
procedure UnRegisterWideCollation(const Name: string);
```

Parameters

Name

User-defined collation name.

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.5 TVirtualFunctionManager Class

Used to register user-defined functions.

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

Unit

virtualQuery

Syntax

```
TVirtualFunctionManager = class(System.TObject);
```

© 1997-2025

Devart. All Rights

Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.5.1 Members

[TVirtualFunctionManager](#) class overview.

Methods

Name	Description
RegisterFunction	Overloaded. Used to register a new function.

© 1997-2025

Devart. All Rights Reserved.

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

6.29.1.5.2 Methods

Methods of the **TVirtualFunctionManager** class.

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

Public

Name	Description
RegisterFunction	Overloaded. Used to register a new function.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

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

6.29.1.5.2.1 RegisterFunction Method

Used to register a new function.

Class

[TVirtualFunctionManager](#)

Overload List

Name	Description
RegisterFunction(const Name: string; ParamCount: Integer; VirtualFunction: TVirtualFunction)	Used to specify the function that is called when a user-defined function is called in an SQL statement.
RegisterFunction(const Name: string; ParamCount: Integer; VirtualMethod: TVirtualMethod)	Used to specify the method that is called when a user-defined function is called in an SQL statement.

© 1997-2025

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

Devart. All Rights Reserved.

Used to specify the function that is called when a user-defined function is called in an SQL statement.

Class

[TVirtualFunctionManager](#)

Syntax

```
procedure RegisterFunction(const Name: string; ParamCount: Integer; VirtualFunction: TVirtualFunction); overload;
```

Parameters

Name

Used to specify the name of the function that will be defined.

ParamCount

Used to specify the number of function parameters.

VirtualFunction

Used to specify the function that is called when a user-defined function is called in an SQL statement.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Used to specify the method that is called when a user-defined function is called in an SQL statement.

Class

[TVirtualFunctionManager](#)

Syntax

```
procedure RegisterFunction(const Name: string; ParamCount: Integer; VirtualMethod: TVirtualMethod); overload;
```

Parameters

Name

Used to specify the name of the method that will be defined.

ParamCount

Used to specify the number of method parameters.

VirtualMethod

Used to specify the method that is called when a user-defined function is called in an SQL statement.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.6 TVirtualQuery Class

Used to retrieve data simultaneously from various RDBMS'es.

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

Unit

virtualQuery

Syntax

```
TVirtualQuery = class(TCustomVirtualQuery);
```

Remarks

TVirtualQuery component is used to retrieve data simultaneously from several different RDBMS'es. Instead of a database connection, it use a collection of TDataSet descendants defined in the [TCustomVirtualQuery.SourceDataSets](#) property as the data source, for which a SQL statement can be build. The SQLite is used as an internal SQL-engine, so the SQLite syntax has to be used for SQL statements.

Use TVirtualQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. 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, TVirtualQuery will retrieve primary keys for UpdatingTable from metadata. TVirtualQuery 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](#)

[TCustomVirtualQuery](#)

TVirtualQuery

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.6.1 Members

[TVirtualQuery](#) class overview.

Properties

Name	Description
BaseSQL (inherited from TCustomDADataset)	Used to return SQL text without any changes performed by <code>AddWhere</code> , <code>SetOrderBy</code> , and <code>FilterSQL</code> .
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.
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.
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.
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 TCustomVirtualQuery)	Used to specify the behaviour of TVirtualQuery 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 TCustomDADataset)	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 (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.
SourceDataSets (inherited from TCustomVirtualQuery)	Contains a collection of source datasets for querying data.
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.
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.
CloneCursor (inherited from TCustomDADataset)	Shares data from another dataset.
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.
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 TCustomDADataset)	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 (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.
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.
ParamByName (inherited from TCustomDADataset)	Sets or uses parameter information for a specific parameter based on its 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)	Cancel changes made to the current record when

	cached updates are enabled.
SaveSQL (inherited from TCustomDADataset)	Saves the SQL property value to BaseSQL.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
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.
OnRegisterCollations (inherited from TCustomVirtualQuery)	Occurs when the connection is opened to register the user-defined collation used in the query text.
OnRegisterFunctions (inherited from TCustomVirtualQuery)	Occurs when the query is opened to register the user-defined functions used in the query text.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.6.2 Properties

Properties of the **TVirtualQuery** class.

For a complete list of the **TVirtualQuery** class members, see the [TVirtualQuery 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.
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.
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.
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 TCustomVirtualQuery)	Used to specify the behaviour of TVirtualQuery 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 TCustomDADataset)	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 (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.
SourceDataSets (inherited from TCustomVirtualQuery)	Contains a collection of source datasets for querying data.
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.
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.
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

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.6.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

[TVirtualQuery](#)

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.

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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.6.2.2 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

[TVirtualQuery](#)

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 [TCustomVirtualQuery.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
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.7 TVirtualQueryOptions Class

Used to set up the behaviour of the TVirtualQuery class.

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

Unit

virtualquery

Syntax

```
TVirtualQueryOptions = class(TDADatasetOptions);
```

Inheritance Hierarchy

[TDADatasetOptions](#)

TVirtualQueryOptions

See Also

- [TVirtualQuery](#)

© 1997-2025

Devart. All Rights Reserved.

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

6.29.1.7.1 Members

[TVirtualQueryOptions](#) class overview.

Properties

Name	Description
AutoOpenSources	Used to automatically open data sources when SQL statement executed
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.
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.
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. 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.
TrimVarChar	Used to specify whether to discard all trailing spaces in the variable-length 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.
UseUnicode	Used to enable or disable Unicode support.

© 1997-2025

Devart. All Rights Reserved.

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

6.29.1.7.2 Properties

Properties of the **TVirtualQueryOptions** class.

For a complete list of the **TVirtualQueryOptions** class members, see the [TVirtualQueryOptions 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
AutoOpenSources	Used to automatically open data sources when SQL statement executed
FullRefresh	Used to specify the fields to include in the automatically generated SQL statement when calling the method.
SetEmptyStrToNull	Force replace of empty strings with NULL values in data. Default value is False.
TrimVarChar	Used to specify whether to discard all trailing spaces in

	the variable-length string fields of a dataset.
UseUnicode	Used to enable or disable Unicode support.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.7.2.1 AutoOpenSources Property

Used to automatically open data sources when SQL statement executed

Class

[TVirtualQueryOptions](#)

Syntax

```
property AutoOpenSources: boolean default False;
```

Remarks

Use the property to automatically open data sources specified in the [TCustomVirtualQuery.SourceDataSets](#) list when SQL statement executed. If AutoOpenSources is False, each data source has to be opened prior to SQL statement execution in the [TVirtualQuery](#). If AutoOpenSources is True, data sources will be opened automatically. The default value is False;

See Also

- [TVirtualQuery](#)
- [TCustomVirtualQuery.SourceDataSets](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.7.2.2 FullRefresh Property

Used to specify the fields to include in the automatically generated SQL statement when calling the method.

Class

[TVirtualQueryOptions](#)

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 [TVirtualQuery.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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.7.2.3 SetEmptyStrToNull Property

Force replace of empty strings with NULL values in data. Default value is False.

Class

[TVirtualQueryOptions](#)

Syntax

```
property SetEmptyStrToNull: boolean;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.7.2.4 TrimVarChar Property

Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

Class

[TVirtualQueryOptions](#)

Syntax

```
property TrimVarChar: boolean;
```

Remarks

Use the TrimVarChar property to specify whether to discard all trailing spaces in the variable-length string fields of a dataset. The default value is False.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.1.7.2.5 UseUnicode Property

Used to enable or disable Unicode support.

Class

[TVirtualQueryOptions](#)

Syntax

```
property UseUnicode: boolean default DefValUseUnicode;
```

Remarks

Use the UseUnicode property to enable or disable Unicode support. When set to True, all character data is stored as WideString, and TStringField is used instead of TWideStringField.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.2 Types

Types in the **VirtualQuery** unit.

Types

Name	Description
TRegisterFunctionsEvent	This type is used for the TCustomVirtualQuery.RegisterFunctions events.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.29.2.1 TRegisterFunctionsEvent Procedure Reference

This type is used for the TCustomVirtualQuery.RegisterFunctions events.

Unit

VirtualQuery

Syntax

```
TRegisterFunctionsEvent = procedure (Sender: TObject; const
FunctionManager: TVirtualFunctionManager) of object;
```

Parameters

Sender

An object that raised the event.

FunctionManager

Used to register user-defined functions.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30 VirtualTable

6.30.1 Classes

Classes in the **VirtualTable** unit.

Classes

Name	Description
TVirtualAutoIncField	A field class for generating auto-increment field values.
TVirtualTable	A base class for storing data in memory.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.1 TVirtualAutoIncField Class

A field class for generating auto-increment field values.

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

Unit

virtualTable

Syntax

```
TVirtualAutoIncField = class(TAutoIncField);
```

Remarks

TVirtualAutoIncField is a TAutoIncField descendant. It is created in cases when TFieldDef of the ftAutoInc type is created either at design time using the TVirtualTable.FieldDefs property, or at run time using the [TVirtualTable.AddField](#) method.

An initial field value is specified by the [TVirtualAutoIncField.InitialValue](#) property, and its increment is specified by the [TVirtualAutoIncField](#) property. Auto-generation can be enabled or disabled using the [TVirtualAutoIncField.AutoGenerateValue](#) property.

See Also

- [TVirtualAutoIncField.InitialValue](#)
- [TVirtualAutoIncField.Increment](#)
- [TVirtualAutoIncField.AutoGenerateValue](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.1.1 Members

[TVirtualAutoIncField](#) class overview.

Properties

Name	Description
AutoGenerateValue	Indicates whether the field values can be automatically generated.
Increment	Specifies the increment of the auto-incrementing field.
InitialValue	Specifies the initial value of the auto-incrementing field.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.1.2 Properties

Properties of the **TVirtualAutoIncField** class.

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

Published

Name	Description
AutoGenerateValue	Indicates whether the field values can be automatically generated.
Increment	Specifies the increment of the auto-incrementing field.
InitialValue	Specifies the initial value of the auto-incrementing field.

See Also

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

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

Reserved.

6.30.1.1.2.1 AutoGenerateValue Property

Indicates whether the field values can be automatically generated.

Class

[TVirtualAutoIncField](#)

Syntax

```
property AutoGenerateValue default arAutoInc;
```

Remarks

Use the AutoGenerateValue property to indicate whether the field values can be automatically generated.

When the property value is arAutoInc, a new incrementing value for the field will be generated automatically when inserting a new record. Otherwise (arNone and arDefault values), the field will be empty. The default value is arAutoInc.

Note: An attempt to change the AutoGenerateValue property when the field dataset is open generates an exception. Thus, AutoGenerateValue only works with persistent field components, which remain when the dataset is closed.

See Also

- [InitialValue](#)
- [Increment](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.1.2.2 Increment Property

Specifies the increment of the auto-incrementing field.

Class

[TVirtualAutoIncField](#)

Syntax

```
property Increment: integer default 1;
```

Remarks

Use the Increment property to specify the increment of the auto-incrementing field.

The increment can be negative. The default value is 1.

See Also

- [InitialValue](#)
- [AutoGenerateValue](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.1.2.3 InitialValue Property

Specifies the initial value of the auto-incrementing field.

Class

[TVirtualAutoIncField](#)

Syntax

```
property InitialValue: integer default - 1;
```

Remarks

Use the InitialValue property to specify the initial value of the auto-incrementing field.

The initial value can be negative. The default value is 1.

See Also

- [Increment](#)
- [AutoGenerateValue](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2 TVirtualTable Class

A base class for storing data in memory.

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

Unit

virtualTable

Syntax

```
TVirtualTable = class(TMemDataSet);
```

Remarks

TVirtualTable is inherited from the TMemDataSet component. TVirtualTable stores data in memory and does not have linked data files. To add fields to virtual table at design time use Fields Editor. Call the [TVirtualTable.AddField](#) method to add fields at run time.

Immediately after creating, virtual table will be empty. Then you define new fields or load existing table files so that the virtual table object becomes initialized and ready to be opened.

When you close virtual table it will discard its record set. To keep the data you entered at design-time for later use you may wish to include the voStored option in the [TVirtualTable.Options](#) property. At run time you will need to call the [TVirtualTable.SaveToFile](#) method explicitly to store modifications to the file that may be retrieved back into the virtual table by calling the [TVirtualTable.LoadFromFile](#) method later.

Note: TVirtualTable component is added to the Data Access page of the component palette, not to the VirtualDAC page.

TVirtualTable supports auto-incrementing fields using a special [TVirtualAutoIncField](#) field class.

Inheritance Hierarchy

[TMemDataSet](#)

TVirtualTable

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.1 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.
Options	Used to specify actions to take on the fields data at the time of opening or closing TVirtualTable dataset.
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
AddField	Adds a new TFieldDef object with the name determined by Name.
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.
Clear	Removes all records from TVirtualTable.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
DeleteField	Deletes a field specified by name.
DeleteFields	Deletes all fields.
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.
SaveToFile	Saves data of a TVirtualTable component to a file.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToStream	Copies data from a TVirtualTable component to a stream.
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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.2 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.
Options	Used to specify actions to take on the fields data at the time of opening or closing TVirtualTable dataset.

See Also

- [TVirtualTable Class](#)

- [TVirtualTable Class Members](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.2.1 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.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.2.2 Options Property

Used to specify actions to take on the fields data at the time of opening or closing TVirtualTable dataset.

Class

[TVirtualTable](#)

Syntax

```
property Options: TVirtualTableOptions default [voPersistentData, voStored, voSkipUnsupportedFieldTypes];
```

Remarks

The Options property specifies what actions to take on the fields data at the time of opening or closing TVirtualTable dataset.

© 1997-2025
Devart. All Rights
Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3 Methods

Methods of the **TVirtualTable** class.

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

Public

Name	Description
AddField	Adds a new TFieldDef object with the name determined by Name.
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.
Clear	Removes all records from TVirtualTable.
CommitUpdates (inherited from TMemDataSet)	Clears the cached updates buffer.
DeferredPost (inherited from TMemDataSet)	Makes permanent changes to the database server.
DeleteField	Deletes a field specified by name.
DeleteFields	Deletes all fields.
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.
SaveToFile	Saves data of a TVirtualTable component to a file.
SaveToJSON (inherited from TMemDataSet)	Overloaded. Saves the current dataset data to a file or a stream in the JSON format.
SaveToStream	Copies data from a TVirtualTable component to a stream.
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](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3.1 AddField Method

Adds a new TFieldDef object with the name determined by Name.

Class

[TVirtualTable](#)

Syntax

```
procedure AddField(const Name: string; FieldType: TFieldType;  
Size: integer = 0; Required: boolean = False);
```

Parameters

Name

Holds the name of the TFieldDef object to add.

FieldType

Holds the type of the TFieldDef object to add.

Size

Holds the size of the string (if the type of TFieldDef object was specified as ftString or ftWideString).

Required

Holds an indicator that determines whether filling the Size parameter is required.

Remarks

Call the AddField method to add a new TFieldDef object with the name determined by Name. FieldType can be ftString, ftWideString, ftSmallint, ftInteger, ftAutoInc, ftWord, ftBoolean, ftLargeint, ftFloat, ftCurrency, ftDate, ftTime, ftDateTime, ftBlob, or ftMemo. When you add ftString or ftWideString field you should specify Size of the string. When the ftAutoInc field type is set, a field of the special [TVirtualAutoIncField](#) type will be created in the virtual table, which can generate auto-increment field values.

See Also

- [DeleteField](#)
- [DeleteFields](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3.2 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 SQL server oriented dataset.

See Also

- [TVirtualTable](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3.3 Clear Method

Removes all records from TVirtualTable.

Class

[TVirtualTable](#)

Syntax

```
procedure Clear;
```

Remarks

Call the Clear method to remove all records from TVirtualTable.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3.4 DeleteField Method

Deletes a field specified by name.

Class

[TVirtualTable](#)

Syntax

```
procedure DeleteField(const Name: string);
```

Parameters

Name

Holds the name of the field to delete.

Remarks

Call the DeleteField method to delete a field specified by Name.

See Also

- [AddField](#)
- [DeleteFields](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3.5 DeleteFields Method

Deletes all fields.

Class

[TVirtualTable](#)

Syntax

```
procedure DeleteFields;
```

Remarks

Call the DeleteFields method to delete all fields.

See Also

- [DeleteField](#)

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3.6 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 file name to load into the dataset as the value of the FileName parameter. This file may be an XML document in ADO-compatible format, JSON format, or in virtual table data format.

The file format is detected automatically.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3.7 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 dataset data 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 dataset data is copied as the value of the Stream parameter. Data in the stream may be in ADO-compatible format, JSON format, or in virtual table data format. The data format is detected automatically.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3.8 SaveToFile Method

Saves data of a TVirtualTable component to a file.

Class

[TVirtualTable](#)

Syntax

```
procedure SaveToFile(const FileName: string; storeFields: boolean = True; storeAllData: boolean = false);
```

Parameters

FileName

Holds the name of the file to save data to.

StoreFields

Indicates whether to save fields to a file.

StoreAllData

Remarks

Call the `SaveToFile` method to save data of a `TVirtualTable` component to a file. Specify the name of the file as the value of the `FileName` parameter.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.1.2.3.9 SaveToStream Method

Copies data from a `TVirtualTable` component to a stream.

Class

[TVirtualTable](#)

Syntax

```
procedure SaveToStream(Stream: TStream; StoreFields: boolean = True; StoreAllData: boolean = false);
```

Parameters

Stream

Holds the name of the stream to which the field's value is saved.

StoreFields

Indicates whether to save the fields names to a file.

StoreAllData

Remarks

Call the `SaveToStream` method to copy data from a `TVirtualTable` component to a stream. Specify the name of the stream to which the field's value is saved as the value of the `Stream` parameter.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.2 Types

Types in the **VirtualTable** unit.

Types

Name	Description
TVirtualTableOptions	Represents the set of TVirtualTableOption .

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.2.1 TVirtualTableOptions Set

Represents the set of [TVirtualTableOption](#).

Unit

virtualTable

Syntax

```
TVirtualTableOptions = set of TVirtualTableOption;
```

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.3 Enumerations

Enumerations in the **VirtualTable** unit.

Enumerations

Name	Description
TVirtualTableOption	Specifies the actions to take on fields data at the time of opening or closing TVirtualTable dataset.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)

6.30.3.1 TVirtualTableOption Enumeration

Specifies the actions to take on fields data at the time of opening or closing TVirtualTable dataset.

Unit

virtualTable

Syntax

```
TVirtualTableOption = (voPersistentData, voStored);
```

Values

Value	Meaning
voPersistentData	Dataset will not dispose of its data at the time of dataset closing.
voStored	Dataset will keep its data set at design-time in DFM file along with other form's stored properties.

© 1997-2025

Devart. All Rights Reserved.

[Request Support](#)

[DAC Forum](#)

[Provide Feedback](#)