# Table of Contents

## Part I  What's New

## Part II  General Information

1. Overview ............................................................................................................. 9
2. Features ............................................................................................................... 12
3. Requirements ...................................................................................................... 16
4. Compatibility ...................................................................................................... 16
5. Using Several DAC Products in One IDE ....................................................... 21
6. Component List .................................................................................................. 21
7. Hierarchy Chart .................................................................................................. 23
8. Editions ............................................................................................................... 25
9. Licensing ............................................................................................................. 28
10. Getting Support ................................................................................................. 32
11. Frequently Asked Questions ............................................................................. 33

## Part III  Getting Started

1. Installation .......................................................................................................... 45
2. Migrating from BDE and IBX ............................................................................ 47
3. Connecting to InterBase and Firebird ............................................................... 48
4. Deleting Data From Tables .............................................................................. 53
5. Creating Database Objects ................................................................................ 56
6. Inserting Data Into Tables ................................................................................ 61
7. Retrieving Data From Tables ........................................................................... 66
8. Modifying Data in Tables ................................................................................ 68
9. Using Stored Procedures .................................................................................. 72
10. Using Stored Procedures with Result Sets ..................................................... 78
11. Demo Projects .................................................................................................. 80
12. Deployment ........................................................................................................ 86

## Part IV  Using IBDAC

1. Updating Data with IBDAC Dataset Components ........................................... 88
2. Master/Detail Relationships ............................................................................. 90
3. Automatic Key Field Value Generation ......................................................... 92
4. Data Type Mapping .......................................................................................... 93
5. Data Encryption ................................................................................................. 100
6. Working in an Unstable Network ..................................................................... 102
7. Disconnected Mode .......................................................................................... 103
Part V Reference

1 CRAccess ................................................................................................................. 136
   Classes ......................................................................................................................... 137
      TCRCursor Class ......................................................................................................... 137
      Members ....................................................................................................................... 138
   Types ............................................................................................................................... 139
      TBeforFetchProc Procedure Reference ....................................................................... 139
   Enumerations .................................................................................................................. 139
      TCRIsolationLevel Enumeration ................................................................................ 140
      TCRTransactionAction Enumeration ....................................................................... 141
      TCursorState Enumeration ...................................................................................... 141
2 CRBatchMove ........................................................................................................... 142
   Classes ............................................................................................................................ 143
      TCRBatchMove Class ................................................................................................. 143
      Members ....................................................................................................................... 144
      Properties .................................................................................................................... 146
         AbortOnKeyViol Property ...................................................................................... 148
         AbortOnProblem Property ...................................................................................... 148
         ChangedCount Property ......................................................................................... 149
         CommitCount Property .......................................................................................... 149
         Destination Property .............................................................................................. 150
         FieldMappingMode Property ................................................................................ 150
         KeyViolCount Property .......................................................................................... 150
         Mappings Property .................................................................................................. 151
         Mode Property ......................................................................................................... 152
         MovedCount Property ............................................................................................. 152
         ProblemCount Property ......................................................................................... 152
         RecordCount Property ............................................................................................. 153
         Source Property ......................................................................................................... 154
      Methods ....................................................................................................................... 154
         Execute Method ......................................................................................................... 155
      Events ........................................................................................................................... 155
         OnBatchMoveProgress Event .................................................................................. 156
## Contents

### Types
- TCRBatchMoveProgressEvent Procedure Reference .............................. 156

### Enumerations
- TCRBatchMode Enumeration ............................................................ 157
- TCRFieldMappingMode Enumeration ................................................ 158

### 3 CRENcryption

#### Classes
- TCREncoder Class .................................................................................. 159
  - Members ................................................................................................ 160
  - Properties .............................................................................................. 161
  - DataHeader Property .............................................................................. 162
  - EncryptionAlgorithm Property ............................................................. 162
  - HashAlgorithm Property ....................................................................... 162
  - InvalidHashAction Property ................................................................. 163
  - Password Property ................................................................................ 164
  - Methods ................................................................................................ 164
  - SetKey Method ...................................................................................... 165

#### Enumerations
- TCREncDataHeader Enumeration .......................................................... 165
- TCREncryptionAlgorithm Enumeration .................................................. 166
- TCRHashAlgorithm Enumeration ........................................................... 167
- TCRInvalidHashAction Enumeration ..................................................... 168

### 4 CRGrid

#### Classes
- TCRDBGrid Class .................................................................................... 159
  - Members ................................................................................................ 160
  - Properties .............................................................................................. 161
  - Columns Property .................................................................................. 172
  - Filtered Property ................................................................................... 173
  - LevelDelimiterChar Property ................................................................. 173
  - OnMemoClick Property ......................................................................... 174
  - OptionsEx Property ................................................................................. 174
  - OptionsMenu Property .......................................................................... 175
  - Methods ................................................................................................ 176
  - ActivateFilterEdit Method ..................................................................... 177
  - ActivateSearchEdit Method ................................................................... 178
  - AdjustColumns Method ......................................................................... 178
  - ApplyFilter Method ................................................................................ 179
  - CalcTitleLevel Method .......................................................................... 179
  - ClearFilters Method ............................................................................... 180
  - ClearSorting Method ............................................................................. 180
  - DataChanged Method ............................................................................. 180
  - GetGridSize Method .............................................................................. 181
  - GetTitleLevel Method ............................................................................ 181
  - Events .................................................................................................. 182
  - OnGetCellParams Event ......................................................................... 182

### 5 CRVio

#### Enumerations
- TIPVersion Enumeration ....................................................................... 183

### 6 DAAlerter

#### Classes
- TDAAlerter Class .................................................................................... 184
  - Members ................................................................................................ 185
7 DADump ........................................................................................................................................ 192
Classes ............................................................................................................................................... 193
TDADump Class ................................................................................................................................... 193
Members ............................................................................................................................................... 194
Properties ............................................................................................................................................... 195
Connection Property ......................................................................................................................... 196
Debug Property .................................................................................................................................... 197
Options Property ................................................................................................................................. 197
SQL Property ......................................................................................................................................... 198
TableNames Property ......................................................................................................................... 199
Methods ............................................................................................................................................... 199
Backup Method .................................................................................................................................... 200
BackupQuery Method ......................................................................................................................... 201
BackupToFile Method ......................................................................................................................... 201
BackupToStream Method ..................................................................................................................... 202
Restore Method ..................................................................................................................................... 203
RestoreFromFile Method .................................................................................................................... 203
RestoreFromStream Method ............................................................................................................... 204
Events .................................................................................................................................................... 205
OnBackupProgress Event .................................................................................................................... 205
OnError Event ....................................................................................................................................... 206
OnRestoreProgress Event .................................................................................................................... 207
TDADumpOptions Class ....................................................................................................................... 207
Members ............................................................................................................................................... 208
Properties ............................................................................................................................................... 208
AddDrop Property ............................................................................................................................... 209
Completelnset Property ....................................................................................................................... 209
GenerateHeader Property ..................................................................................................................... 210
QuoteNames Property .......................................................................................................................... 210
Types ..................................................................................................................................................... 211
TDADumpBackupEvent Procedure Reference .................................................................................... 211
TDADumpRestoreEvent Procedure Reference .................................................................................. 212
8 DALoader .......................................................................................................................................... 212
Classes ............................................................................................................................................... 213
TDAColumn Class ............................................................................................................................... 214
Members ............................................................................................................................................... 214
Properties ............................................................................................................................................... 215
FieldType Property .............................................................................................................................. 215
Name Property ....................................................................................................................................... 216
TDAColumns Class .............................................................................................................................. 216
Members ............................................................................................................................................... 217
InterBase Data Access Components

Events ................................................................. 250
  AfterExecute Event ........................................ 251
  BeforeExecute Event ........................................ 252
  OnError Event ............................................... 252
TDAStatement Class ........................................... 253
  Members ....................................................... 253
  Properties .................................................... 254
    EndLine Property .......................................... 255
    EndOffset Property ........................................ 256
    EndPos Property ........................................... 256
    Omit Property .............................................. 256
    Params Property .......................................... 257
    Script Property .......................................... 257
    SQL Property .............................................. 258
    StartLine Property ....................................... 258
    StartOffset Property ..................................... 259
    StartPos Property ........................................ 259
  Methods ....................................................... 259
    Execute Method .......................................... 260
TDAStatements Class ........................................ 260
  Members ....................................................... 261
  Properties .................................................... 261
    Items Property (Indexer) ............................... 262
Types .................................................................. 262
  TAfterStatementExecuteEvent Procedure Reference ........................................ 263
  TBeforeStatementExecuteEvent Procedure Reference ........................................ 263
  TOnErrorEvent Procedure Reference .................................. 264
Enumerations .................................................... 264
  TErrorAction Enumeration .................................. 265
10 DASQLMonitor ................................................ 265
Classes .................................................................. 266
  TCustomDASQLMonitor Class ............................... 267
    Members ............................................................ 267
    Properties ....................................................... 268
      Active Property ........................................ 269
      DBMonitorOptions Property ......................... 269
      Options Property ........................................ 269
      TraceFlags Property .................................... 270
    Events .......................................................... 271
      OnSQL Event ................................................ 271
TDBMonitorOptions Class .................................... 272
  Members ....................................................... 272
  Properties .................................................... 273
    Host Property .............................................. 273
    Port Property .............................................. 274
    ReconnectTimeout Property ............................ 274
    SendTimeout Property .................................... 275
Types .................................................................. 275
  TDATraceFlags Set .......................................... 276
  TMonitorOptions Set ....................................... 276
  TOnSQLEvent Procedure Reference ....................... 276
Enumerations .................................................... 277
  TDATraceFlag Enumeration ................................. 277
  TMonitorOption Enumeration .............................. 278
# 11 DBAccess

<table>
<thead>
<tr>
<th>Classes</th>
<th>279</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDAError Class</td>
<td>284</td>
</tr>
<tr>
<td>Members</td>
<td>284</td>
</tr>
<tr>
<td>Properties</td>
<td>285</td>
</tr>
<tr>
<td>Component Property</td>
<td>285</td>
</tr>
<tr>
<td>ErrorCode Property</td>
<td>286</td>
</tr>
<tr>
<td>TCDataSource Class</td>
<td>286</td>
</tr>
<tr>
<td>Members</td>
<td>287</td>
</tr>
<tr>
<td>TCustomConnectDialog Class</td>
<td>287</td>
</tr>
<tr>
<td>Members</td>
<td>287</td>
</tr>
<tr>
<td>Properties</td>
<td>289</td>
</tr>
<tr>
<td>CancelButton Property</td>
<td>290</td>
</tr>
<tr>
<td>Caption Property</td>
<td>290</td>
</tr>
<tr>
<td>ConnectButton Property</td>
<td>290</td>
</tr>
<tr>
<td>DialogClass Property</td>
<td>291</td>
</tr>
<tr>
<td>LabelSet Property</td>
<td>292</td>
</tr>
<tr>
<td>PasswordLabel Property</td>
<td>292</td>
</tr>
<tr>
<td>Retries Property</td>
<td>292</td>
</tr>
<tr>
<td>SavePassword Property</td>
<td>293</td>
</tr>
<tr>
<td>ServerLabel Property</td>
<td>293</td>
</tr>
<tr>
<td>StoreLoginInfo Property</td>
<td>294</td>
</tr>
<tr>
<td>UsernameLabel Property</td>
<td>294</td>
</tr>
<tr>
<td>Methods</td>
<td>295</td>
</tr>
<tr>
<td>Execute Method</td>
<td>295</td>
</tr>
<tr>
<td>GetServerList Method</td>
<td>296</td>
</tr>
<tr>
<td>TCustomDAConnection Class</td>
<td>296</td>
</tr>
<tr>
<td>Members</td>
<td>297</td>
</tr>
<tr>
<td>Properties</td>
<td>299</td>
</tr>
<tr>
<td>ConnectDialog Property</td>
<td>300</td>
</tr>
<tr>
<td>ConnectString Property</td>
<td>301</td>
</tr>
<tr>
<td>ConvertSQL Property</td>
<td>302</td>
</tr>
<tr>
<td>InTransaction Property</td>
<td>303</td>
</tr>
<tr>
<td>LoginPrompt Property</td>
<td>303</td>
</tr>
<tr>
<td>Options Property</td>
<td>304</td>
</tr>
<tr>
<td>Password Property</td>
<td>305</td>
</tr>
<tr>
<td>Pooling Property</td>
<td>305</td>
</tr>
<tr>
<td>PoolingOptions Property</td>
<td>306</td>
</tr>
<tr>
<td>Server Property</td>
<td>307</td>
</tr>
<tr>
<td>Username Property</td>
<td>308</td>
</tr>
<tr>
<td>Methods</td>
<td>308</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>310</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>310</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>311</td>
</tr>
<tr>
<td>Commit Method</td>
<td>311</td>
</tr>
<tr>
<td>Connect Method</td>
<td>312</td>
</tr>
<tr>
<td>CreateSQL Method</td>
<td>313</td>
</tr>
<tr>
<td>Disconnect Method</td>
<td>313</td>
</tr>
<tr>
<td>ExecProc Method</td>
<td>314</td>
</tr>
<tr>
<td>ExecProcEx Method</td>
<td>316</td>
</tr>
<tr>
<td>ExecSQL Method</td>
<td>317</td>
</tr>
<tr>
<td>ExecSQLEx Method</td>
<td>318</td>
</tr>
<tr>
<td>GetDatabaseNames Method</td>
<td>319</td>
</tr>
<tr>
<td>GetKeyFieldNames Method</td>
<td>320</td>
</tr>
<tr>
<td>Method/Property</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>GetStoredProcNames Method</td>
<td>321</td>
</tr>
<tr>
<td>GetTableNames Method</td>
<td>322</td>
</tr>
<tr>
<td>MonitorMessage Method</td>
<td>323</td>
</tr>
<tr>
<td>Ping Method</td>
<td>323</td>
</tr>
<tr>
<td>RemoveFromPool Method</td>
<td>324</td>
</tr>
<tr>
<td>Rollback Method</td>
<td>324</td>
</tr>
<tr>
<td>StartTransaction Method</td>
<td>325</td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td>325</td>
</tr>
<tr>
<td>OnConnectionLost Event</td>
<td>326</td>
</tr>
<tr>
<td>OnError Event</td>
<td>327</td>
</tr>
<tr>
<td><strong>TCustomDADataSet Class</strong></td>
<td>327</td>
</tr>
<tr>
<td><strong>Members</strong></td>
<td>328</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td>335</td>
</tr>
<tr>
<td>BaseSQL Property</td>
<td>339</td>
</tr>
<tr>
<td>Conditions Property</td>
<td>340</td>
</tr>
<tr>
<td>Connection Property</td>
<td>340</td>
</tr>
<tr>
<td>DataTypeMap Property</td>
<td>341</td>
</tr>
<tr>
<td>Debug Property</td>
<td>341</td>
</tr>
<tr>
<td>DetailFields Property</td>
<td>342</td>
</tr>
<tr>
<td>Disconnected Property</td>
<td>343</td>
</tr>
<tr>
<td>FetchRows Property</td>
<td>343</td>
</tr>
<tr>
<td>FilterSQL Property</td>
<td>344</td>
</tr>
<tr>
<td>FinalSQL Property</td>
<td>344</td>
</tr>
<tr>
<td>IsQuery Property</td>
<td>345</td>
</tr>
<tr>
<td>KeyFields Property</td>
<td>345</td>
</tr>
<tr>
<td>MacroCount Property</td>
<td>346</td>
</tr>
<tr>
<td>Macros Property</td>
<td>347</td>
</tr>
<tr>
<td>MasterFields Property</td>
<td>348</td>
</tr>
<tr>
<td>MasterSource Property</td>
<td>349</td>
</tr>
<tr>
<td>Options Property</td>
<td>349</td>
</tr>
<tr>
<td>ParamCheck Property</td>
<td>351</td>
</tr>
<tr>
<td>ParamCount Property</td>
<td>352</td>
</tr>
<tr>
<td>Params Property</td>
<td>353</td>
</tr>
<tr>
<td>ReadOnly Property</td>
<td>353</td>
</tr>
<tr>
<td>RefreshOptions Property</td>
<td>354</td>
</tr>
<tr>
<td>RowsAffected Property</td>
<td>355</td>
</tr>
<tr>
<td>SQL Property</td>
<td>355</td>
</tr>
<tr>
<td>SQLDelete Property</td>
<td>356</td>
</tr>
<tr>
<td>SQLInsert Property</td>
<td>357</td>
</tr>
<tr>
<td>SQLLock Property</td>
<td>357</td>
</tr>
<tr>
<td>SQLRecCount Property</td>
<td>358</td>
</tr>
<tr>
<td>SQLRefresh Property</td>
<td>359</td>
</tr>
<tr>
<td>SQLUpdate Property</td>
<td>360</td>
</tr>
<tr>
<td>UniDirectional Property</td>
<td>361</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td>361</td>
</tr>
<tr>
<td>AddWhere Method</td>
<td>365</td>
</tr>
<tr>
<td>BreakExec Method</td>
<td>366</td>
</tr>
<tr>
<td>CreateBlobStream Method</td>
<td>367</td>
</tr>
<tr>
<td>DeleteWhere Method</td>
<td>367</td>
</tr>
<tr>
<td>Execute Method</td>
<td>367</td>
</tr>
<tr>
<td>Execute Method</td>
<td>368</td>
</tr>
<tr>
<td>Execute Method</td>
<td>368</td>
</tr>
<tr>
<td>Executive Method</td>
<td>369</td>
</tr>
<tr>
<td>Executing Method</td>
<td>370</td>
</tr>
<tr>
<td>Fetched Method</td>
<td>370</td>
</tr>
</tbody>
</table>

© 2021 Devart
## TCustomDASQL Class

<table>
<thead>
<tr>
<th>Member</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetching Method</td>
<td>371</td>
</tr>
<tr>
<td>FetchingAll Method</td>
<td>371</td>
</tr>
<tr>
<td>FindKey Method</td>
<td>372</td>
</tr>
<tr>
<td>FindMacro Method</td>
<td>372</td>
</tr>
<tr>
<td>FindNearest Method</td>
<td>373</td>
</tr>
<tr>
<td>FindParam Method</td>
<td>374</td>
</tr>
<tr>
<td>GetDataType Method</td>
<td>375</td>
</tr>
<tr>
<td>GetFieldObject Method</td>
<td>375</td>
</tr>
<tr>
<td>GetFieldPrecision Method</td>
<td>376</td>
</tr>
<tr>
<td>GetFieldScale Method</td>
<td>377</td>
</tr>
<tr>
<td>GetKeyFieldNames Method</td>
<td>377</td>
</tr>
<tr>
<td>GetOrderBy Method</td>
<td>378</td>
</tr>
<tr>
<td>GotoCurrent Method</td>
<td>379</td>
</tr>
<tr>
<td>Lock Method</td>
<td>379</td>
</tr>
<tr>
<td>MacroByName Method</td>
<td>380</td>
</tr>
<tr>
<td>ParamByName Method</td>
<td>381</td>
</tr>
<tr>
<td>Prepare Method</td>
<td>382</td>
</tr>
<tr>
<td>RefreshRecord Method</td>
<td>382</td>
</tr>
<tr>
<td>RestoreSQL Method</td>
<td>383</td>
</tr>
<tr>
<td>SaveSQL Method</td>
<td>384</td>
</tr>
<tr>
<td>SetOrderBy Method</td>
<td>384</td>
</tr>
<tr>
<td>SQLSaved Method</td>
<td>385</td>
</tr>
<tr>
<td>UnLock Method</td>
<td>385</td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td></td>
</tr>
<tr>
<td>AfterExecute Event</td>
<td>386</td>
</tr>
<tr>
<td>AfterFetch Event</td>
<td>387</td>
</tr>
<tr>
<td>AfterUpdateExecute Event</td>
<td>387</td>
</tr>
<tr>
<td>BeforeFetch Event</td>
<td>388</td>
</tr>
<tr>
<td>BeforeUpdateExecute Event</td>
<td>388</td>
</tr>
<tr>
<td><strong>TCustomDASQL Class</strong></td>
<td>389</td>
</tr>
<tr>
<td><strong>Members</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td></td>
</tr>
<tr>
<td>ChangeCursor Property</td>
<td>392</td>
</tr>
<tr>
<td>Connection Property</td>
<td>393</td>
</tr>
<tr>
<td>Debug Property</td>
<td>394</td>
</tr>
<tr>
<td>FinalSQL Property</td>
<td>394</td>
</tr>
<tr>
<td>MacroCount Property</td>
<td>395</td>
</tr>
<tr>
<td>Macros Property</td>
<td>396</td>
</tr>
<tr>
<td>ParamCheck Property</td>
<td>396</td>
</tr>
<tr>
<td>ParamCount Property</td>
<td>397</td>
</tr>
<tr>
<td>Params Property</td>
<td>397</td>
</tr>
<tr>
<td>ParamValues Property (Indexer)</td>
<td>398</td>
</tr>
<tr>
<td>Prepared Property</td>
<td>399</td>
</tr>
<tr>
<td>RowsAffected Property</td>
<td>400</td>
</tr>
<tr>
<td>SQL Property</td>
<td>400</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td></td>
</tr>
<tr>
<td>BreakExec Method</td>
<td>401</td>
</tr>
<tr>
<td>Execute Method</td>
<td>402</td>
</tr>
<tr>
<td>Execute Method</td>
<td>402</td>
</tr>
<tr>
<td>Execute Method</td>
<td>403</td>
</tr>
<tr>
<td>Executing Method</td>
<td>403</td>
</tr>
<tr>
<td>FindMacro Method</td>
<td>404</td>
</tr>
<tr>
<td>FindParam Method</td>
<td>404</td>
</tr>
<tr>
<td>MacroByName Method</td>
<td>405</td>
</tr>
<tr>
<td><strong>IXContents</strong></td>
<td></td>
</tr>
</tbody>
</table>
InterBase Data Access Components

ParamByName Method .................................................................................................................. 407
Prepare Method ............................................................................................................................ 408
UnPrepare Method ........................................................................................................................ 409
WaitExecuting Method ............................................................................................................... 409
Events ........................................................................................................................................ 410
AfterExecute Event .................................................................................................................... 410
TCustomDAUpdateSQL Class ...................................................................................................... 411
Members ....................................................................................................................................... 411
Properties .................................................................................................................................... 413
 DataSet Property ......................................................................................................................... 414
DeleteObject Property ............................................................................................................... 415
DeleteSQL Property .................................................................................................................... 415
InsertObject Property .............................................................................................................. 416
InsertSQL Property ................................................................................................................... 416
LockObject Property .................................................................................................................. 417
LockSQL Property ..................................................................................................................... 417
ModifyObject Property ............................................................................................................. 418
ModifySQL Property ................................................................................................................ 418
RefreshObject Property ........................................................................................................... 419
RefreshSQL Property .............................................................................................................. 419
SQL Property(Indexer) ............................................................................................................... 420
Methods ...................................................................................................................................... 420
Apply Method ............................................................................................................................ 421
ExecSQL Method ....................................................................................................................... 422
TDACondition Class ................................................................................................................... 422
Members ....................................................................................................................................... 423
Properties .................................................................................................................................... 423
 Enabled Property ....................................................................................................................... 424
Name Property .......................................................................................................................... 424
Value Property .......................................................................................................................... 425
Methods ...................................................................................................................................... 425
Disable Method .......................................................................................................................... 425
Enable Method ........................................................................................................................... 426
TDAConditions Class .................................................................................................................. 426
Members ....................................................................................................................................... 427
Properties .................................................................................................................................... 428
 Condition Property(Indexer) ...................................................................................................... 429
Enabled Property ....................................................................................................................... 429
Items Property(Indexer) ............................................................................................................ 429
Text Property .............................................................................................................................. 430
WhereSQL Property .................................................................................................................. 430
Methods ...................................................................................................................................... 431
Add Method .................................................................................................................................. 431
Add Method .................................................................................................................................. 431
Add Method .................................................................................................................................. 432
Delete Method ............................................................................................................................ 433
Disable Method .......................................................................................................................... 434
Enable Method ........................................................................................................................... 434
Find Method ............................................................................................................................... 434
Get Method .................................................................................................................................. 435
IndexOf Method ........................................................................................................................... 435
Remove Method .......................................................................................................................... 436
TDAConnectionOptions Class ....................................................................................................... 436
Members ....................................................................................................................................... 436
© 2021 Devart
<table>
<thead>
<tr>
<th>Class</th>
<th>Members</th>
<th>Properties</th>
<th>Methods</th>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAMetaData Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDAParam Class</td>
<td></td>
<td>AsBlob Property</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDAMapRules Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDAMetaData Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDAParam Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDAParams Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDATransaction Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Page</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commit Method</td>
<td>507</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rollback Method</td>
<td>508</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StartTransaction Method</td>
<td>508</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>509</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OnCommit Event</td>
<td>510</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OnCommitRetaining Event</td>
<td>510</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OnError Event</td>
<td>511</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OnRollback Event</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OnRollbackRetaining Event</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMacro Class</td>
<td>513</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Members</td>
<td>514</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>514</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Active Property</td>
<td>515</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AsDateTime Property</td>
<td>516</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AsFloat Property</td>
<td>516</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AsInteger Property</td>
<td>516</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AsString Property</td>
<td>517</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name Property</td>
<td>517</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value Property</td>
<td>518</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMacros Class</td>
<td>518</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Members</td>
<td>519</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>519</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Items Property (Indexer)</td>
<td>520</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>520</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AssignValues Method</td>
<td>521</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expand Method</td>
<td>522</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FindMacro Method</td>
<td>522</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsEqual Method</td>
<td>523</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MacroByName Method</td>
<td>523</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scan Method</td>
<td>524</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPoolingOptions Class</td>
<td>525</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Members</td>
<td>525</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>526</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConnectionLifetime Property</td>
<td>526</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaxPoolSize Property</td>
<td>527</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MinPoolSize Property</td>
<td>527</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validate Property</td>
<td>528</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSmartFetchOptions Class</td>
<td>528</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Members</td>
<td>529</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>529</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enabled Property</td>
<td>530</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LiveBlock Property</td>
<td>530</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrefetchedFields Property</td>
<td>531</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQLGetKeyValues Property</td>
<td>531</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Types</td>
<td>532</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAAfterExecuteEvent Procedure Reference</td>
<td>532</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAAfterFetchEvent Procedure Reference</td>
<td>533</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBeforeFetchEvent Procedure Reference</td>
<td>533</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TConnectionLostEvent Procedure Reference</td>
<td>534</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDAConnectionErrorHandlerProcedure Reference</td>
<td>534</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDATransactionErrorHandlerProcedure Reference</td>
<td>534</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRefreshOptions Set</td>
<td>535</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TUpdateExecuteEvent Procedure Reference</td>
<td>536</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2021 Devart
12 IBC

Classes

TCustomIBDataSet Class

Members

Properties

AutoCommit Property

Connection Property

Cursor Property

DMLRefresh Property

Encryption Property

FetchAll Property

GeneratorMode Property

GeneratorStep Property

Handle Property

IsQuery Property

KeyGenerator Property

LockMode Property

Options Property

Plan Property

RowDeleted Property

RowFetched Property

RowInserted Property

RowsUpdated Property

SmartFetch Property

SQLType Property

Transaction Property

UpdateObject Property

UpdateTransaction Property

Methods

CreateProcCall Method

FindParam Method

GetArray Method

GetBlob Method

ParamByName Method

TCustomIBCQuery Class

Members

TCustomIBCTable Class

Members

Properties

Exists Property

Methods

DeleteTable Method

EmptyTable Method

TIBCArrary Class

Members

Properties

ItemType Property

TIBCArrayField Class
Members .................................................................................................................. 620
Properties .................................................................................................................. 620
AsArray Property ...................................................................................................... 621

TIBCDataSetOptions Class .......................................................................................... 621
Members ...................................................................................................................... 622
Properties .................................................................................................................. 627
AutoCommit Property ................................................................................................. 629
ClientLibrary Property ............................................................................................... 630
Connected Property ................................................................................................... 631
ConnectPrompt Property ............................................................................................ 631
Database Property ..................................................................................................... 632
DatabaseInfo Property ............................................................................................... 633
DBSQLDialect Property .............................................................................................. 633
Debug Property ......................................................................................................... 634
DefaultTransaction Property ..................................................................................... 634
Handle Property ......................................................................................................... 635
LastError Property ..................................................................................................... 635
Options Property ........................................................................................................ 636
Params Property .......................................................................................................... 637
Password Property ...................................................................................................... 638
Port Property ............................................................................................................... 638
Server Property ......................................................................................................... 639
SQL Property ............................................................................................................... 639
SQLDialect Property .................................................................................................. 640
SSLOptions Property .................................................................................................. 640
TransactionCount Property ....................................................................................... 641
Transactions Property(Indexer) .................................................................................. 641
Username Property ..................................................................................................... 642

Methods ...................................................................................................................... 642
AddTransaction Method ............................................................................................... 645
AssignConnect Method ............................................................................................... 645
CommitRetaining Method ............................................................................................ 646
CreateDatabase Method ............................................................................................. 647
DropDatabase Method ............................................................................................... 647
FindDefaultTransaction Method ................................................................................ 647
ParamByName Method ............................................................................................... 648
RemoveTransaction Method ....................................................................................... 649
RollbackRetaining Method ......................................................................................... 649

TIBCConnectionOptions Class .................................................................................... 650
Members ...................................................................................................................... 650
Properties .................................................................................................................. 652
CharLength Property .................................................................................................. 654
Charset Property ......................................................................................................... 654
EnableBCD Property .................................................................................................. 655
EnableFMTBCD Property ............................................................................................ 655
EnableMemos Property .............................................................................................. 655
IPVersion Property .................................................................................................... 656
NoDBTriggers Property ............................................................................................... 657
Protocol Property ....................................................................................................... 657
Role Property ............................................................................................................... 657
TrustedAuthentication Property ................................................................................ 658
UseUnicode Property ............................................................................................... 658

TIBCDatasetOptions Class ........................................................................................... 659
Members ...................................................................................................................... 659
<table>
<thead>
<tr>
<th>Class</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCDatasource Class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCDBKeyField Class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCEncryptor Class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCMetaData Class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transaction Property</td>
</tr>
<tr>
<td>TIBCParam Class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AsArray Property</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AsBlobBlob Property</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Methods</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetBlobData Method</td>
</tr>
<tr>
<td>TIBCParams Class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Items Property(Enumerable)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Methods</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FindParam Method</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ParamByName Method</td>
</tr>
<tr>
<td>TIBCQuery Class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FetchAll Property</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LockMode Property</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UpdatingTable Property</td>
</tr>
<tr>
<td>TIBCSQL Class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connection Property</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DescribeParams Property</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Handle Property</td>
</tr>
</tbody>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Class</th>
<th>Members</th>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Params Property</td>
<td>720</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQLType Property</td>
<td>720</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction Property</td>
<td>721</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>721</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateProcCall Method</td>
<td>722</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExecuteNext Method</td>
<td>723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FindParam Method</td>
<td>723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParamByName Method</td>
<td>724</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCTransaction Class</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Members</td>
<td>725</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>726</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClientCertFile Property</td>
<td>727</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClientPassPhrase Property</td>
<td>727</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClientPassPhraseFile Property</td>
<td>727</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enabled Property</td>
<td>728</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ServerPublicFile Property</td>
<td>728</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ServerPublicPath Property</td>
<td>729</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCSSLConnectionOptions Class</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Members</td>
<td>725</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>740</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LockMode Property</td>
<td>745</td>
<td></td>
<td></td>
</tr>
<tr>
<td>StoredProcName Property</td>
<td>746</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>747</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExecProc Method</td>
<td>751</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prepare Method</td>
<td>751</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrepareSQL Method</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCTable Class</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Members</td>
<td>753</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>763</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FetchAll Property</td>
<td>769</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LockMode Property</td>
<td>770</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TableName Property</td>
<td>770</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCTransaction Class</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Members</td>
<td>771</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>774</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Active Property</td>
<td>775</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connections Property(Indexer)</td>
<td>776</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConnectionsCount Property</td>
<td>776</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DefaultCloseAction Property</td>
<td>777</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DefaultConnection Property</td>
<td>777</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handle Property</td>
<td>778</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsolationLevel Property</td>
<td>778</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Params Property</td>
<td>779</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>779</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AddConnection Method</td>
<td>780</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commit Method</td>
<td>781</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CommitRetaining Method</td>
<td>781</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FindDefaultConnection Method</td>
<td>782</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReleaseSavepoint Method</td>
<td>782</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RemoveConnection Method</td>
<td>783</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rollback Method</td>
<td>784</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RollbackRetaining Method</td>
<td>784</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RollbackSavepoint Method</td>
<td>784</td>
<td></td>
<td></td>
</tr>
<tr>
<td>StartSavepoint Method</td>
<td>785</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### InterBase Data Access Components

<table>
<thead>
<tr>
<th>Events</th>
<th>786</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnError Event</td>
<td>787</td>
</tr>
<tr>
<td>TIBCUpdateSQL Class</td>
<td>787</td>
</tr>
<tr>
<td>Members</td>
<td>788</td>
</tr>
<tr>
<td>Types</td>
<td>789</td>
</tr>
<tr>
<td>TIBCTransactionErrorEvent</td>
<td>790</td>
</tr>
<tr>
<td>Procedures</td>
<td></td>
</tr>
<tr>
<td>Members</td>
<td>791</td>
</tr>
<tr>
<td>Constants</td>
<td>792</td>
</tr>
<tr>
<td>Variables</td>
<td>793</td>
</tr>
<tr>
<td>UseDefConnection Variable</td>
<td>793</td>
</tr>
<tr>
<td>Connections Variable</td>
<td>793</td>
</tr>
<tr>
<td>DefConnection Variable</td>
<td>793</td>
</tr>
<tr>
<td>TIBCTransactionAction Enumeration</td>
<td>794</td>
</tr>
<tr>
<td>BDACVersion Constant</td>
<td>794</td>
</tr>
</tbody>
</table>

#### 13 IBCAdmin

<table>
<thead>
<tr>
<th>Classes</th>
<th>797</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomIBCService Class</td>
<td>799</td>
</tr>
<tr>
<td>Members</td>
<td>799</td>
</tr>
<tr>
<td>Properties</td>
<td>800</td>
</tr>
<tr>
<td>Methods</td>
<td>801</td>
</tr>
<tr>
<td>Attach Method</td>
<td>802</td>
</tr>
<tr>
<td>Detach Method</td>
<td>803</td>
</tr>
<tr>
<td>ServiceStart Method</td>
<td>804</td>
</tr>
<tr>
<td>Events</td>
<td>805</td>
</tr>
<tr>
<td>OnAttach Event</td>
<td>806</td>
</tr>
<tr>
<td>TIBCBackupRestoreService Class</td>
<td>807</td>
</tr>
<tr>
<td>Members</td>
<td>808</td>
</tr>
<tr>
<td>Properties</td>
<td>809</td>
</tr>
<tr>
<td>BackupFile Property</td>
<td>810</td>
</tr>
<tr>
<td>NBBackupLevel Property</td>
<td>811</td>
</tr>
<tr>
<td>NBBackupOptions Property</td>
<td>812</td>
</tr>
<tr>
<td>UseNBBackup Property</td>
<td>813</td>
</tr>
<tr>
<td>Verbose Property</td>
<td>814</td>
</tr>
<tr>
<td>TIBCBackupService Class</td>
<td>815</td>
</tr>
<tr>
<td>Members</td>
<td>816</td>
</tr>
<tr>
<td>Properties</td>
<td>817</td>
</tr>
<tr>
<td>BackupFile Property</td>
<td>818</td>
</tr>
<tr>
<td>BlockingFactor Property</td>
<td>819</td>
</tr>
<tr>
<td>Database Property</td>
<td>820</td>
</tr>
<tr>
<td>Options Property</td>
<td>821</td>
</tr>
<tr>
<td>Verbose Property</td>
<td>822</td>
</tr>
<tr>
<td>TIBCConfigParams Class</td>
<td>823</td>
</tr>
<tr>
<td>Members</td>
<td>824</td>
</tr>
<tr>
<td>Properties</td>
<td>825</td>
</tr>
<tr>
<td>BaseLocation Property</td>
<td>826</td>
</tr>
</tbody>
</table>

© 2021 Devart
LockFileLocation Property ........................................................................................................ 821
MessageFileLocation Property ................................................................................................ 821
SecurityDatabaseLocation Property .................................................................................... 821
TIBConfigService Class ......................................................................................................... 822
Members .................................................................................................................................. 822
Properties ............................................................................................................................... 825
Connection Property .............................................................................................................. 826
Database Property .................................................................................................................. 826
JournalInformation Property .................................................................................................. 827
Transaction Property .............................................................................................................. 827
Methods .................................................................................................................................. 828
ActivateShadow Method ........................................................................................................ 829
AlterJournal Method ............................................................................................................... 830
BringDatabaseOnline Method .................................................................................................. 830
CreateJournal Method ............................................................................................................ 831
CreateJournalArchive Method ............................................................................................... 831
DisableFlush Method ............................................................................................................. 832
DropJournal Method ............................................................................................................... 832
DropJournalArchive Method .................................................................................................. 833
FlushDatabase Method .......................................................................................................... 833
GetJournalInformation Method ............................................................................................. 834
ReclaimMemory Method ....................................................................................................... 834
ServiceStart Method .............................................................................................................. 835
SetAsyncMode Method .......................................................................................................... 835
SetDBSqlDialect Method ........................................................................................................ 836
SetFlushInterval Method ....................................................................................................... 836
SetGroupCommit Method ...................................................................................................... 837
SetLingerInterval Method ...................................................................................................... 837
SetPageBuffers Method ......................................................................................................... 838
SetReadOnly Method .............................................................................................................. 839
SetReclaimInterval Method ................................................................................................. 839
SetSweepInterval Method ...................................................................................................... 840
Shutdow nDatabase Method ................................................................................................. 841
SweepDatabase Method ......................................................................................................... 842
TIBControlAndQueryService Class ..................................................................................... 842
Members ................................................................................................................................. 843
Properties ............................................................................................................................... 844
BufferSize Property .............................................................................................................. 845
Eof Property ............................................................................................................................ 846
Methods .................................................................................................................................. 846
GetNextChunk Method ........................................................................................................... 847
GetNextLine Method .............................................................................................................. 847
TIBDatabaseInfo Class .......................................................................................................... 848
Members ................................................................................................................................. 848
Properties ............................................................................................................................... 848
DbName Property ................................................................................................................... 849
NoOfAttachments Property ................................................................................................. 849
TIBJournalInformation Class .............................................................................................. 850
Members ................................................................................................................................. 850
Properties ............................................................................................................................... 851
CheckpointInterval Property ............................................................................................... 852
CheckpointLength Property ................................................................................................. 853
Directory Property ................................................................................................................ 853
InterBase Data Access Components

TIBCSecurityService Class

Members

Properties

TIBCRestoreService Class

Members

Properties

TIBCLicenseMaskInfo Class

Members

Properties

.HasArchive Property
.HasJournal Property
.PageCache Property
.PageLength Property
.PageSize Property
.TimestampName Property

TIBCLicenseInfo Class

Members

Properties

CapabilityMask Property
.LicenseMask Property

TIBCLicensingService Class

Members

Properties

Action Property
.ID Property
.Key Property

Methods

AddLicense Method
.RemoveLicense Method

TIBCLimboTransactionInfo Class

Members

Properties

Action Property
.Advise Property
.HostSite Property
.ID Property
.MultiDatabase Property
.RemoteDatabasePath Property
.RemoteSite Property
.State Property

TIBCLogService Class

Members

Properties

BackupFile Property
.Database Property
.Options Property
.PageBuffers Property
.PageSize Property
.Verbose Property

TIBCSecurityService Class

Members

Properties

SecurityAction Property
.UserDatabase Property
User Info Property ................................................................. 887
User Infos Property (Indexer) ................................................ 887
User Infos Count Property ................................................... 888
Methods ........................................................................... 888
Add User Method ............................................................... 889
Delete User Method ........................................................... 890
Display User Method .......................................................... 890
Display Users Method ........................................................ 891
Enable EUA Method ............................................................ 891
Modify User Method ............................................................ 892
Suspend EUA Method .......................................................... 892
TIBCServerProperties Class ................................................ 893
Members ........................................................................... 893
Properties ......................................................................... 895
Add Alias Method ............................................................ 900
Delete Alias Method ......................................................... 901
Properties ......................................................................... 901
Add User Method ............................................................... 900
Delete User Method ........................................................... 889
Display User Method .......................................................... 890
Display Users Method ........................................................ 891
Enable EUA Method ............................................................ 891
Modify User Method ............................................................ 892
Suspend EUA Method .......................................................... 892
Methods ........................................................................... 899
Add Alias Method ............................................................ 900
Delete Alias Method ......................................................... 901
Add Alias Info Method ....................................................... 902
Fetch Config Params Method ................................................ 902
Fetch Database Info Method ................................................ 903
Fetch License Info Method .................................................. 903
Fetch License Mask Info Method ........................................ 904
Fetch Version Info Method .................................................. 904
Properties ......................................................................... 907
Database Property ............................................................. 908
Options Property ............................................................... 908
Table Names Property ......................................................... 909
TIBCStatisticalService Class ................................................ 909
Members ........................................................................... 909
Properties ......................................................................... 912
Properties ......................................................................... 912
Config Property ................................................................. 913
Session Name Property ....................................................... 914
Methods ........................................................................... 914
List Trace Sessions Method .................................................. 915
Resume Trace Method ........................................................ 916
Start Trace Method ............................................................. 916
Stop Trace Method .............................................................. 917
Suspend Trace Method ........................................................ 917
TIBCUserInfo Class ............................................................. 918
Members ........................................................................... 918
Properties ......................................................................... 919
Active User Property .......................................................... 920
Default Role Property ........................................................ 920
Description Property ......................................................... 921
First Name Property ............................................................. 921
InterBase Data Access Components

IBCAlerter

Classes

TIBCAlerter Class

Enumerations

TIBCValidateOption Enumeration

TIBCTransactionGlobalAction Enumeration

TIBCSecurityAction Enumeration

TIBCLicensingAction Enumeration

TIBCBackupOption Enumeration

TIBCRestoreOptions Set

TIBCNBackupOptions Set

TIBCSecurityAction Set

TIBCValidationService Class

Members

Properties

Connections Property

ServiceVersion Property

FixLimboTransactionErrors Method

Options Property

GlobalAction Property

Database Property

LimboTransactionCount Property

UserName Property

SystemUserName Property

PassWord Property

SQLRole Property

Properties

Members

Types

TIBCBackupOptions Set

TIBCLicensingAction Set

TIBCBackupOption Enumeration

TIBCRestoreOptions Set

TIBCStatsOptions Set

TIBCValidateOptions Set

14 IBCAlerter

Classes

TIBCAlerter Class

Members

Properties

AutoCommit Property

Connection Property

Events Property

Transaction Property
15 IBCArray

TCustomIBCArray Class

Methods
- SetItemAsSmallInt Method
- SetItemAsInteger Method
- SetItemAsFloat Method
- ReadArraySlice Method
- ReadArrayItem Method
- ReadArray Method
- GetItemValue Method
- GetItemAsSmallInt Method
- GetItemAsInteger Method
- GetItemAsString Method
- GetArrayInfo Method
- CreateTemporaryArray Method
- Assign Method

Properties
- ArrayDimensions Property
- ArrayHighBound Property (Indexer)
- ArrayID Property
- ArrayLowBound Property (Indexer)
- ArraySize Property
- AsString Property
- Cached Property
- CachedDimensions Property
- CachedHighBound Property (Indexer)
- CachedLowBound Property (Indexer)
- CachedSize Property
- ColumnName Property
- DbHandle Property
- IsNull Property
- Items Property
- ItemScale Property
- ItemSize Property
- Modified Property
- TableName Property
- TrHandle Property

Events
- OnEvent Event

Types
- TIBCAAlertEvent Procedure Reference
16 IBCClasses

Classes

TGDSDatabaseInfo Class

Members

Properties

Allocation Property
AttachmentID Property
BackoutCount Property
BaseLevel Property
CurLogFileName Property
CurLogPartitionOffset Property
CurrentMemory Property
DBFileName Property
DBImplementationClass Property
DBImplementationNo Property
DBSiteName Property
DeleteCount Property
ExpungeCount Property
Fetches Property
ForcedWrites Property
InsertCount Property
IsRemoteConnect Property
LogFile Property
Marks Property
MaxMemory Property
NoReserve Property
NumBuffers Property
NumWALBuffers Property
ODSMajorVersion Property
ODSMinorVersion Property
PageSize Property
PurgeCount Property
ReadIdxCount Property
ReadOnly Property
Reads Property
ReadSeqCount Property
SweepInterval Property
UpdateCount Property
UserNames Property
Version Property
WALAverageGroupCommitSize Property
WALAverageIOSize Property
WALBufferSize Property
WALCheckpointLength Property
WALCurCheckpointInterval Property
WALGroupCommitWaitUSecs Property
WALNumCommits Property
WALNumIO Property
WALPrvCheckpointFilename Property
WALPrvCheckpointPartOffset Property
Writes Property
<table>
<thead>
<tr>
<th>Classes</th>
<th>17 IBCCo mpo</th>
<th></th>
<th>18 IBCDa t yMap</th>
<th></th>
<th>19 IBErr or</th>
<th></th>
<th>20 IBCLoade r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Members</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>Properties</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>Cached Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>CharSetID Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>ConversionCharsetName Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>ConversionSubType Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>DbHandle Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>Handle Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>ID Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>MaxSegmentSize Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>NumSegments Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>Streamed Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>SubType Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>TrHandle Property</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>Methods</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>AllocBlob Method</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>CloseBlob Method</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>FreeBlob Method</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>IsInit Method</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>LengthBlob Method</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>ReadBlob Method</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>WriteBlob Method</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>Enumerations</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td></td>
</tr>
<tr>
<td>Routines</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>Reverse2 Function</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>Reverse4 Function</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>XSQLDA_LENGTH Function</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>Variables</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
<tr>
<td>IntegerPrecision Variable</td>
<td>TIBCCo mpo</td>
<td>IBCCo mpoManag er</td>
<td>IBCDa t yMap</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
<td>IBCErr or</td>
</tr>
</tbody>
</table>

© 2021 Devart
21 *IBCScript* .............................................................................................................................................. 1053

**Classes** .................................................................................................................................................. 1053

*IBCScript* Class ........................................................................................................................................ 1054

Members ................................................................................................................................................... 1054

Properties ................................................................................................................................................ 1056

AutoDDL Property ................................................................................................................................. 1058

Connection Property .............................................................................................................................. 1058

DataSet Property .................................................................................................................................. 1059

Params Property .................................................................................................................................... 1059

Transaction Property ............................................................................................................................. 1060

22 *IBCSQLMonitor* ..................................................................................................................................... 1060

**Classes** .................................................................................................................................................. 1061

*IBCSQLMonitor* Class ............................................................................................................................ 1061

Members ................................................................................................................................................... 1062

23 *IBDacVcl* ............................................................................................................................................... 1063

**Classes** .................................................................................................................................................. 1063

*IBConnectDialog* Class .......................................................................................................................... 1063

Members ................................................................................................................................................... 1064

Properties ................................................................................................................................................ 1066

Connection Property .............................................................................................................................. 1067

DatabaseLabel Property .......................................................................................................................... 1067

ProtocolLabel Property .......................................................................................................................... 1068

**Routines** ............................................................................................................................................... 1068

GetIBCDatabaseList Procedure ............................................................................................................. 1069

GetIBCServerList Procedure ................................................................................................................ 1070

24 *MemData* .............................................................................................................................................. 1070

**Classes** .................................................................................................................................................. 1072

*TAttribute* Class .................................................................................................................................... 1073

Members ................................................................................................................................................... 1073

Properties ................................................................................................................................................ 1074

AttributeName Property .......................................................................................................................... 1075

DataType Property .................................................................................................................................. 1076

Length Property ..................................................................................................................................... 1076

ObjectProperty ......................................................................................................................................... 1077

Offset Property ....................................................................................................................................... 1077

Owner Property ...................................................................................................................................... 1078

Scale Property ......................................................................................................................................... 1078

Size Property .......................................................................................................................................... 1079
TMemDataSet Class

Enumerations

TUpdateRecKind Enumeration
TLocateExOption Enumeration
TConnLostCause Enumeration
TCompressBlobMode Enumeration

TSharedObject Class

Properties

Methods

TCompressedBlob Class

Members

Properties

Compressed Property
CompressedSize Property

TDLObject Class

Members

TMemData Class

Members

TObjectType Class

Members

Properties

AttributeCount Property
Attributes Property(indexer)
DataType Property
Size Property

Methods

FindAttribute Method

TSharedObject Class

Members

Properties

RefCount Property

Methods

AddRef Method
Release Method

Types

TLocateExOptions Set
TUpdateRecKinds Set

Enumerations

TCompressBlobMode Enumeration
TConnLostCause Enumeration
TDANumericType Enumeration
TLocateExOption Enumeration
TSortType Enumeration
TUpdateRecKind Enumeration

25 MemDS

Classes

TMemDataSet Class

Members

Properties
### Classes

- **TVirtualDataSet Class**
  - **DoNotRaiseExceptionOnUaFail Variable**

### Members

### Events

- **OnUpdateError Event**
- **UpdateStatus Method**
- **UpdateResult Method**
- **SetRangeStart Method**
- **SetRangeEnd Method**
- **SaveToXML Method**
- **Prepare Method**
- **LocateEx Method**
- **Locate Method**
- **CommitUpdates Method**
- **DeferredPost Method**
- **EditRangeEnd Method**
- **EditRangeStart Method**
- **GetBlob Method**
- **SetRanged Property**
- **Prepared Property**
- **LocalUpdate Property**
- **LocalConstraints Property**
- **KeyExclusive Property**
- **CachedUpdates Property**
- **UpdatesPending Property**
- **UpdatesRecordTypes Property**

### Methods

- **ApplyRange Method**
- **ApplyUpdates Method**
- **CancelRange Method**
- **CancelUpdates Method**
- **CommitUpdates Method**
- **DeferredPost Method**
- **EditRangeEnd Method**
- **EditRangeStart Method**
- **GetBlob Method**
- **LocateEx Method**
- **Locate Method**
- **Prepare Method**
- **LocateEx Method**
- **Locate Method**
- **SetRangeStart Method**
- **SetRangeEnd Method**
- **UnPrepare Method**
- **UpdateResult Method**
- **UpdateStatus Method**
- **SaveToXML Method**
- **SaveToXML Method**
- **SaveToXML Method**
- **SetRange Method**
- **SetRangeStart Method**
- **SetRangeEnd Method**
- **UnPrepare Method**
- **Locate Method**

### Variables

- **DoNotRaiseExceptionOnUaFail Variable**
- **SendDataSetChangeEventAfterOpen Variable**

26 **VirtualDataSet**

### Classes

- **TCustomVirtualDataSet Class**
  - **Members**

### Types

© 2021 Devart
27 VirtualTable ........................................................................................................... 1162

Classes ....................................................................................................................... 1162

TVirtualTable Class ..................................................................................................... 1163

Members ..................................................................................................................... 1163

Properties ................................................................................................................. 1166

DefaultSortType Property ......................................................................................... 1168

Methods ..................................................................................................................... 1168

Assign Method........................................................................................................... 1170

LoadFromFile Method............................................................................................... 1171

LoadFromStream Method.......................................................................................... 1172
1 What's New

02-Mar-21 New Features in IBDAC 7.4:
- RAD Studio 10.4.2 Sydney is supported
- macOS 11 Big Sur is supported
- iOS 14 is supported
- Android 11 is supported
- Over-the-Wire (OTW) encryption is supported
- Performance of batch operations is improved
- Memory consumption in batch operations is reduced
- Performance of the FindFirst, FindNext, FindLast, and FindPrior methods is improved
- Automatic detection of computed fields when generating update statements is improved

26-Aug-20 New Features in IBDAC 7.3:
- Lazarus 2.0.10 and FPC 3.2.0 are supported
- Performance of Batch Insert, Update, and Delete operations is improved

01-Jun-20 New Features in IBDAC 7.2:
- RAD Studio 10.4 Sydney is supported
- Lazarus 2.0.8 is supported
- macOS 64-bit in Lazarus is supported

26-Nov-19 New Features in IBDAC 7.1:
- Android 64-bit is supported
- Lazarus 2.0.6 is supported
- Interbase 2020 is supported
- Now Trial edition for macOS and Linux is fully functional
- Improved performance when using pooling
- Handling of BLOB fields in batch operations is improved

22-Jul-19 New Features in IBDAC 7.0:
- macOS 64-bit is supported
- Release 2 for RAD Studio 10.3 Rio, Delphi 10.3 Rio, and C++Builder 10.3 Rio is now required
24-Jun-19 New Features in IBDAC 6.4:
- Lazarus 2.0.2 is supported
- Local connection is supported
- The URL-style connection string format is supported
- The TIBConnectionOptions.IPVersion property is added
- The TCustomIBCService.IPVersion property is added
- Improved performance when inserting data into a table having BLOB fields using Loader component
- The DefaultSortType property for TVirtualTable is added
- Performance of the SaveToFile/LoadFromFile methods of TVirtualTable is significantly increased

26-Nov-18 New Features in IBDAC 6.3:
- RAD Studio 10.3 Rio is supported
- Possibility to write large blobs by pieces is added
- Support of UPPER and LOWER functions for Unified SQL is added
- The boSkipData option for the BackupService component is added
- The roSkipData option for the RestoreService component is added
- TIBCQuery.OnGetBlobData event is added

09-Jul-18 New Features in IBDAC 6.2:
- Lazarus 1.8.4 is supported
- Support for System Encryption Password (SEP) is added
- Possibility to grant/revoke admin role for a Firebird user with the help of the IBCSecurityService component is added
- Performance of batch operations is improved
- Demo projects for IntraWeb 14 are added
- Now the "Data type is not supported" exception is not raised by the Query component when the DescribeParams property is set to True

19-Sep-17 New Features in IBDAC 6.1:
- 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
05-Apr-17 New Features in IBDAC 6.0:
- 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
- Possibility to manage batch operations using a transaction is added
- Possibility to obtain active transaction number using DBMonitor is added

25-Apr-16 New Features in IBDAC 5.7:
- RAD Studio 10.1 Berlin is supported
- Lazarus 1.6 and FPC 3.0.0 is supported
- Support for the BETWEEN statement in TDADataSet.Filter is added
- Data Type Mapping performance is improved
- Performance of TDALoader on loading data from TDataSet is improved

09-Sep-15 New Features in IBDAC 5.6:
- RAD Studio 10 Seattle is supported
- INSERT, UPDATE and DELETE batch operations are supported
- Now Trial for Win64 is a fully functional Professional Edition

14-Apr-15 New Features in IBDAC 5.5:
- RAD Studio XE8 is supported
- AppMethod is supported
- Firebird 3 is supported
- Firebird 3 BOOLEAN column type is supported
- The roMetadataOnly option in the RestoreService component is added

15-Sep-14 New Features in IBDAC 5.4:
- RAD Studio XE7 is supported
- Lazarus 1.2.4 is supported
- Demo projects for FastReport 5 are added
- The TCustomDADataset.GetKeyFieldNames method is added
- The ConstraintColumns metadata kind for the TDAMetadata component is added

29-Apr-14 New Features in IBDAC 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
• Now update queries inside TDataSet descendants have correct owner
• The TIBCDatasetOptions.MasterFieldsNullable property is added

25-Dec-13 New Features in IBDAC 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
• Performance is improved
• Compatibility of migrating floating-point fields from other components is improved

18-Sep-13 New Features in IBDAC 5.1:
• RAD Studio XE5 is supported
• Application development for Android is supported
• Lazarus 1.0.12 is supported
• Performance is improved
• Automatic checking for new versions is added
• Flexible management of conditions in the WHERE clause is added
• The possibility to use conditions is added
• Support of the IN keyword in the TDataSet.Filter property is added
• Like operator behaviour when used in the Filter property is now similar to TClientDataSet
• The possibility to use ranges is added
• The Ping method for the Connection component is added
• The AllowImplicitConnect option for the Connection component is added
• The SQLRecCount property for the Query and StoredProc components is added
• The ScanParams property for the Script component is added
• The RowsAffected property for the Script component is added
• Trusted authentication mode for Firebird is supported
• Migration from FIBPlus is added
• Now the TIBCTransaction.Params property values can be separated by a semicolon
• The ForceUsingDefaultPort global variable is added
• TIBCLoader.LoadFromDataSet is optimized for cases when a dataset record count is less than the RowsPerBatch value
25-Apr-13 New Features in IBDAC 5.0:
- Rad Studio XE4 is supported
- NEXTGEN compiler is supported
- Application development for iOS is 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
- InterBase XE3 ToGo Edition support for iOS device is added
- Additional database shutdown options for TIBCConfigService.ShutdownDatabase are added

12-Dec-12 New Features in IBDAC 4.6:
- Rad Studio XE3 Update 1 is now required
- C++Builder 64-bit for Windows is supported
- TIBCConnection.Port property that allows specifying the port number or the service name for connection is added

05-Sep-12 New Features in IBDAC 4.5:
- Rad Studio XE3 is supported
- Windows 8 is supported

23-Nov-11 New Features in IBDAC 4.1:
- Update 2 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
- Mac OS X and iOS in RAD Studio XE2 is supported
- FireMonkey support is improved
- Lazarus 0.9.30.2 and FPC 2.4.4 are supported
- Mac OS X in Lazarus is supported
- Linux x64 in Lazarus is supported
- FreeBSD in Lazarus is supported

15-Sep-11 New Features in InterBase 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
• TIBCDataSetOptions.SetDomainNames property to enable setting TIBCFieldDesc.DomainName for fields is added
• TIBCLoader.RowsPerBatch property to specify the number of INSERT queries to load in a single batch is added

28-Apr-11 New Features in InterBase Data Access Components 3.60:
• Lazarus 0.9.30 and FPC 2.4.2 is supported
• TIBCLoader.InsertMode property allowing the use of "UPDATE OR INSERT INTO" syntax for loading data is added
• Possibility to assign Handle to TIBCConnection is added

13-Sep-10 New Features in InterBase Data Access Components 3.50:
• Embarcadero RAD Studio XE supported

10-Sep-09 New Features in InterBase Data Access Components 3.10:
• Embarcadero RAD Studio 2010 supported

02-Apr-09 New Features in InterBase Data Access Components 3.00:
• **TIBCLoader** component

  serves for fast loading of data to the database. For Firebird 2.0 and higher it combines INSERT statements in one EXECUTE BLOCK statement to speed up loading.

• **InterBase services components**

  allow to backup and restore database, configure server parameters and security.

• Free Pascal under Linux supported
• Added NoPreconnect property to TIBCScript for executing CONNECT and CREATE
DATABASE commands

23-Oct-08 New Features in InterBase Data Access Components 2.70:
- Delphi 2009 and C++Builder 2009 supported
- Extended Unicode support for Delphi 2007 added (special Unicode build)
- Free Pascal 2.2 supported
- Powerful design-time editors implemented in Lazarus
- Completed with more comprehensive structured Help

23-May-08 New Features in InterBase Data Access Components 2.50:
- Added compatibility with UniDAC
- Improved support of default field values
- The new component for metadata receiving added

27-Sep-07 New Features in InterBase Data Access Components 2.20:
- CodeGear RAD Studio 2007 supported
- Added ability to treat integer fields as TBooleanField when the domain name contains "BOOLEAN"

12-Jun-07 New Features in InterBase Data Access Components 2.10:
- C++Builder 2007 supported

22-Mar-07 New Features in InterBase Data Access Components 2.00:

New functionality:
- Delphi 2007 for Win32 support
- Implemented Disconnected Model for working offline and automatically connecting and disconnecting
- Implemented Local Failover for detecting connection loss and implicitly re-executing some operations
- WideMemo field type in Delphi 2006 supported
• Added **DataSet Manager** to control project datasets
• New **TCRBatchMove** component for transferring data between all types of TDataSet descendants added
• Data export and import to/from XML supported
• Support for sending messages to DBMonitor from any point in your program added

**Support for more InterBase/Firebird server functionality:**
• **RETURNING clause in the INSERT SQL statement** (Firebird 2 server only) supported
• EXECUTE BLOCK syntax (Firebird 2 server only) supported
• Automatic updates by DB_Key unique field (Firebird 2 server only) supported
• **Default values in stored procedures** supported

**Extensions and improvements to existing functionality:**
• General performance improved
• **Master/detail** functionality extensions:
  o **Local master/detail** relationship support added
  o Support for master/detail relationships in **CachedUpdates** mode added
• **Connection pool** functionality improvements:
  o Efficiency significantly improved
  o New **API for draining the connection pool** added
• **TIBCScript** component improvements:
  o Support for executing **individual statements** in scripts added
  o Support for **executing huge scripts stored in files** with dynamic loading added
  o Ability to use standard ISQL tool syntax added
• Greatly increased **performance of applying updates** in **CachedUpdates** mode
• Working with **calculated and lookup fields** improvements:
  o Local **sorting** and filtering added
  o Record **location** speed increased
  o Improved working with lookup fields
• Ability to customize update commands by attaching external components to **TIBCUpdateSQL** objects added
• Ability to **include all fields** in automatically generated update SQLs added

**Usability improvements:**
• **Syntax highlighting** in design-time editors added
• Completely restructured and clearer **demo projects**
28-Aug-06 New Features in InterBase Data Access Components 1.10:

- Professional editions of Turbo Delphi, Turbo Delphi for .NET, Turbo C++ supported

© 1997-2021 Devart. All Rights Reserved.

2 General Information

This section contains general information about InterBase Data Access Components

- **Overview**
- **Features**
- **Requirements**
- **Compatibility**
- **Using Several DAC Products in One IDE**
- **Component List**
- **Hierarchy Chart**
- **Editions**
- **Licensing and Subscriptions**
- **Getting Support**
- **Frequently Asked Questions**

© 1997-2021 Devart. All Rights Reserved.

2.1 Overview

InterBase Data Access Components (IBDAC) is a library of components that provides access to InterBase and Firebird database servers. IBDAC directly uses InterBase client software to connect to server. The IBDAC library is designed to help programmers develop faster and cleaner InterBase database applications. IBDAC is a complete replacement for standard InterBase connectivity solutions. It presents an efficient alternative to the Borland Database Engine for access to InterBase and InterBase Express Components.

The IBDAC library is actively developed and supported by the Devart Team. If you have questions about IBDAC, email the developers at ibdac@devart.com or visit IBDAC online at https://www.devart.com/ibdac/.
Advantages of IBDAC Technology

IBDAC is a direct connectivity database wrapper built specifically for the InterBase server. IBDAC offers wide coverage of the InterBase feature set, and emphasizes optimized data access strategies.

Wide Coverage of InterBase Features

By providing access to the most advanced database functionality, IBDAC allows developers to harness the full capabilities of the InterBase server and optimize their database applications. IBDAC provides complete support for InterBase Blobs and Arrays, support for Unicode character data, InterBase events. View the full list of supported InterBase features in Features.

Optimized Code

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

Compatibility with other Connectivity Methods

The IBDAC interface retains compatibility with standard VCL data access components like BDE and IBX. Existing BDE- and IBX-based applications can be easily migrated to IBDAC and enhanced to take advantage of InterBase-specific features. Project migration can be automated with the BDE/IBX Migration Wizard. Find out more about Migration Wizard in Using Migration Wizard.

Development and Support

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

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

Key Features

- Direct access to server data. Does not require installation of other data provider layers
(such as BDE and ODBC)
- VCL, LCL and FireMonkey versions of the library available
- Full support of the latest versions of InterBase and Firebird database servers
- Support for all InterBase data types
- **Disconnected Model** with automatic connection control for working with data offline
- **Local Failover** for detecting connection loss and implicitly re-executing certain operations
- All types of local sorting and filtering, including by calculated and lookup fields
- Automatic data updating with TIBCQuery and TIBCTable components
- Unicode and national charsets support
- **InterBase Events support**
- Advanced script execution functionality with TIBCScript component
- **Support for using Macros in SQL**
- Easy migration from BDE and IBX with Migration Wizard
- Lets you use Professional Edition of Delphi and C++Builder to develop client/server applications
- Included annual IBDAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

The full list of IBDAC features can be found in Features.

**How does IBDAC work?**

IBDAC uses InterBase client software to connect to the server directly through the native InterBase interface, without using BDE or ODBC. It is designed to be lightweight. It consists of a minimal layer between InterBase server and your code. This extends functionality without sacrificing performance.

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

**IBDAC Connection**
2.2 Features

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

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

Network and connectivity:
• Disconnected Model with automatic connection control for working with data offline
• Local Failover for detecting connection loss and implicitly reexecuting certain operations

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

InterBase technology support:
• Support for fast record insertion with the TIBClassLoader component
• InterBase event support
• Comprehensive array data type support
• 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
• 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

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

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

Data access and data management automation:
• Automatic data updating with TIBCQuery and TIBCTable components
• Automatic record refreshing and locking
• Automatic query preparing
• Support for ftWideMemo field type in Delphi 2006 and higher

Extended data access functionality:
• Separate component for executing SQL statements
• Simplified access to table data with TIBCTable component
• BLOB compression support
• Support for using macros in SQL
• Ability to customize update commands by attaching external components to TIBCUpdateSQL objects
• Deferred detail DataSet refresh in master/detail relationships
• **MIDAS** technology support

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

**Script execution:**
• Advanced script execution features with **TIBCScript** component
• Support for executing individual statements in scripts
• Support for executing huge scripts stored in files with dynamic loading
• Ability to use standard ISQL syntax in scripts

**SQL execution monitoring:**
• Extended SQL tracing capabilities provided by **TIBCSQLMonitor** component and **DBMonitor application**
• Borland SQL Monitor support
• Ability to send messages to DBMonitor from any point in your program

**Visual extensions:**
• Includes source code of enhanced TCRDBGGrid 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, IBX and FibPlus components with **Migration Wizard**
• More convenient data source setup with the **TIBCDDataSource** component
• Syntax highlighting in design-time editors

**Resources:**
• Code documentation and guides in the CHM, PDF, and HXS formats
• Many helpful **demo** projects
Licensing and support:
- Included annual IBDAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

2.3 Requirements

Applications based on IBDAC require client software only (the gds32.dll or fbclient.dll libraries for InterBase or Firebird correspondingly). IBDAC dynamically loads the client library available on user systems. By default, IBDAC searches client libraries in directories specified in the PATH environment variable. The TIBCConnection.ClientLibrary property is used to specify the path to the client library. Starting with Delphi XE2 it is possible to develop 64-bit applications. To develop 64-bit applications, you should use 64-bit client libraries. For more information, please refer to the "Deployment" and "Database Specific Aspects of 64-bit Development" articles.

See Also
- FAQ: What software should be installed on a client computer so that my applications that use IBDAC can run?

2.4 Compatibility

Database Server Compatibility

<table>
<thead>
<tr>
<th>InterBase</th>
<th>Windows</th>
<th>macOS</th>
<th>Linux</th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versions since XE3 up to 2020</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Versions since XE</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Versions since 4.2</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### IDE Compatibility

IBDAC is compatible with the following IDEs:

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

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

- **Embarcadero RAD Studio 10.2 Tokyo**
  - Embarcadero Delphi 10.2 Tokyo for Windows
  - Embarcadero Delphi 10.2 Tokyo for macOS
  - Embarcadero Delphi 10.2 Tokyo for Linux
  - Embarcadero Delphi 10.2 Tokyo for iOS
  - Embarcadero Delphi 10.2 Tokyo for Android
  - Embarcadero C++Builder 10.2 Tokyo for Windows
  - Embarcadero C++Builder 10.2 Tokyo for macOS
  - Embarcadero C++Builder 10.2 Tokyo for iOS
  - Embarcadero C++Builder 10.2 Tokyo for Android
- Embarcadero RAD Studio 10.1 Berlin
  - Embarcadero Delphi 10.1 Berlin for Windows
  - Embarcadero Delphi 10.1 Berlin for macOS
  - Embarcadero Delphi 10.1 Berlin for iOS
  - Embarcadero Delphi 10.1 Berlin for Android
  - Embarcadero C++Builder 10.1 Berlin for Windows
  - Embarcadero C++Builder 10.1 Berlin for macOS
  - Embarcadero C++Builder 10.1 Berlin for iOS
  - Embarcadero C++Builder 10.1 Berlin for Android
- Embarcadero RAD Studio 10 Seattle
  - Embarcadero Delphi 10 Seattle for Windows
  - Embarcadero Delphi 10 Seattle for macOS
  - Embarcadero Delphi 10 Seattle for iOS
  - Embarcadero Delphi 10 Seattle for Android
  - Embarcadero C++Builder 10 Seattle for Windows
  - Embarcadero C++Builder 10 Seattle for macOS
  - Embarcadero C++Builder 10 Seattle for iOS
  - Embarcadero C++Builder 10 Seattle for Android
- Embarcadero RAD Studio XE8
  - Embarcadero Delphi XE8 for Windows
  - Embarcadero Delphi XE8 for macOS
  - Embarcadero Delphi XE8 for iOS
  - Embarcadero Delphi XE8 for Android
  - Embarcadero C++Builder XE8 for Windows
  - Embarcadero C++Builder XE8 for macOS
  - Embarcadero C++Builder XE8 for iOS
  - Embarcadero C++Builder XE8 for Android
- Embarcadero RAD Studio XE7
  - Embarcadero Delphi XE7 for Windows
  - Embarcadero Delphi XE7 for macOS
  - Embarcadero Delphi XE7 for iOS
  - Embarcadero Delphi XE7 for Android
  - Embarcadero C++Builder XE7 for Windows
  - Embarcadero C++Builder XE7 for macOS
  - Embarcadero C++Builder XE7 for iOS
  - Embarcadero C++Builder XE7 for Android
- Embarcadero RAD Studio XE6
- Embarcadero Delphi XE6 for Windows
- Embarcadero Delphi XE6 for macOS
- Embarcadero Delphi XE6 for iOS
- Embarcadero Delphi XE6 for Android
- Embarcadero C++Builder XE6 for Windows
- Embarcadero C++Builder XE6 for macOS
- Embarcadero C++Builder XE6 for iOS
- Embarcadero C++Builder XE6 for Android

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

- Embarcadero Delphi 2010
- Embarcadero C++Builder 2010
- CodeGear RAD Studio 2009 (Requires Update 3)
  - CodeGear Delphi 2009
  - CodeGear C++Builder 2009
- CodeGear RAD Studio 2007
  - CodeGear Delphi 2007
  - CodeGear C++Builder 2007
- Borland Developer Studio 2006
  - Borland Delphi 2006
  - Borland C++Builder 2006
- Borland Delphi 7
- Borland Delphi 6 (Requires Update Pack 2 – Delphi 6 Build 6.240)
- Borland C++Builder 6 (Requires Update Pack 4 – C++Builder 6 Build 10.166)
- Lazarus 2.0.10 and Free Pascal 3.2.0 for Windows, macOS, and Linux (32-bit and 64-bit)

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

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

Supported Target Platforms

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

Note that support for 64-bit Windows and macOS was introduced in RAD Studio XE2, and is not available in older versions of RAD Studio. Support for iOS is available since RAD Studio XE4, but support for iOS 64-bit is available since RAD Studio XE8. Support for Android is available since RAD Studio XE5. Support for Linux 64-bit is available since RAD Studio 10.2 Tokyo. Support for macOS 64-bit is available since RAD Studio 10.3 Rio. Support for Android 64-bit is available since RAD Studio 10.3.3 Rio. Support for macOS 32-bit and iOS 32-bit was removed in RAD Studio 10.4.
Supported GUI Frameworks
- FireMonkey (FMX)
- Visual Component Library (VCL)
- Lazarus Component Library (LCL)

Devart Data Access Components Compatibility

All DAC products are compatible with each other.

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

© 1997-2021
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.

© 1997-2021
Devart. All Rights Reserved.

2.6 Component List

This topic presents a brief description of the components included in the InterBase Data Access Components library. Click on the name of each component for more information. These components are added to the IBDAC page of the Component palette except for
**TCRBatchMove** and **TVirtualTable** components. **TCRBatchMove** and **TVirtualTable** components are added to the Data Access page of the Component palette.

## Basic IBDAC components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCConnection</td>
<td>Sets and controls connection to InterBase database.</td>
</tr>
<tr>
<td>TIBCTransaction</td>
<td>Provides discrete transaction control over database connections.</td>
</tr>
<tr>
<td>TIBCQuery</td>
<td>Uses SQL statements to retrieve data from InterBase table or tables. Single SELECT statement may be adequately used to generate missing INSERT, DELETE, UPDATE statements.</td>
</tr>
<tr>
<td>TIBCSQL</td>
<td>Executes SQL statements and stored procedures, which do not return rowsets.</td>
</tr>
<tr>
<td>TIBCTable</td>
<td>Lets you retrieve and update data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td>TIBCStoredProc</td>
<td>Executes stored procedures and functions, allows to edit cursor data returned as parameter.</td>
</tr>
<tr>
<td>TIBCUpdateSQL</td>
<td>Lets you tune update operations for a DataSet component.</td>
</tr>
<tr>
<td>TIBCDataSource</td>
<td>Provides an interface between an IBDAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TIBCScript</td>
<td>Executes sequences of SQL statements.</td>
</tr>
<tr>
<td>TIBCSQLMonitor</td>
<td>Use to monitor dynamic SQL execution in IBDAC based applications.</td>
</tr>
<tr>
<td>TIBCConnectDialog</td>
<td>Used to build custom prompts for username, password and server name.</td>
</tr>
<tr>
<td>TVirtualTable</td>
<td>Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.</td>
</tr>
<tr>
<td>TVirtualDataSet</td>
<td>Dataset that processes arbitrary non-tabular data.</td>
</tr>
</tbody>
</table>

## IBDAC Professional Edition components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCEncryptor</td>
<td>Represents data encryption and decryption in client application.</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TIBCLoader</td>
<td>Allows to load external data into the database table.</td>
</tr>
<tr>
<td>TIBCAlerter</td>
<td>Use to transfer messages between connections.</td>
</tr>
<tr>
<td>TIBCMetaData</td>
<td>Retrieves metadata on specified SQL object.</td>
</tr>
<tr>
<td>TIBCServerProperties</td>
<td>Returns database server information, including configuration parameters, and also version and license information.</td>
</tr>
<tr>
<td>TIBCConfigService</td>
<td>Configures database parameters.</td>
</tr>
<tr>
<td>TIBCLicensingService</td>
<td>Adds or removes InterBase software activation certificates.</td>
</tr>
<tr>
<td>TIBCLogService</td>
<td>Returns the contents of the interbase.log file from server.</td>
</tr>
<tr>
<td>TIBCStatisticalService</td>
<td>Shows database statistics.</td>
</tr>
<tr>
<td>TIBCValidationService</td>
<td>Validates a database and reconciles database transactions.</td>
</tr>
<tr>
<td>TIBCSecurityService</td>
<td>Used to manage user access to the InterBase server.</td>
</tr>
<tr>
<td>TIBCTraceService</td>
<td>Used for working with trace service added in Firebird 2.5.</td>
</tr>
<tr>
<td>TIBCBackupService</td>
<td>Used to backup a database.</td>
</tr>
<tr>
<td>TIBCRestoreService</td>
<td>Used to restore a database.</td>
</tr>
<tr>
<td>TCRBatchMove</td>
<td>Transfers data between all types of TDataSets descendants. This component is placed on the Data Access page of the Component palette.</td>
</tr>
</tbody>
</table>

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

2.7 **Hierarchy Chart**

Many IBDAC classes are inherited from standard VCL/LCL classes. The inheritance hierarchy chart for IBDAC is shown below. The IBDAC 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
   |   |   |—TIBCConnection
   |—TDataSet
   |   |—TMemDataSet
   |   |   |—TCustomDADataset
   |   |   |   |—TCustomIBCDataset
   |   |   |   |   |—TCustomIBCQuery
   |   |   |—TIBCQuery
   |   |   |—TIBCStoredProc
   |   |   |—TCustomIBCTable
   |   |   |   |—TIBCTable
   |   |—TDAMetaData
   |   |   |—TIBCMetaData
   |   |   |—TVirtualTable
   |—TDataSource
   |   |—TCRDataSource
   |   |   |—TIBCDataset
   |—DADeveloper
   |   |—IBCDataset
   |   |—TCRBatchMove
   |—TCustomConnectDialog
   |   |—TIBCConnectDialog
   |—TCustomDASQL
   |   |—TIBCSQL
   |—TCustomDASQLMonitor
   |   |—TIBCSQLMonitor
   |—TCustomDAUpdateSQL
   |   |—TIBCUpdateSQL
   |—TDALoader
   |   |—TIBCLoader
   |—TDAScript
```
2.8 Editions

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

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

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

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

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

**IBDAC Edition Matrix**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Standard</th>
<th>Professional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Desktop Application Development</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>macOS</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Linux</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Mobile Application Development</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iOS</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Android</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Data Access Components</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Components:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBConnection</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>TIBCQuery</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCSQL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCTable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCStoredProc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCUpdateSQL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCDataSource</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td>Available</td>
<td>Required</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>Script Executing</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>InterBase Data Access Components</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transactions managing</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fast data loading into the server</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Interbase &amp; Firebird Specific Components</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Messaging between sessions and applications</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Obtaining metadata about database objects</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Database server information return</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>InterBase Services</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Database parameters configurations</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Managing of activation certificates</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>The server log return</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Shows database statistics</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Database backup</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Database restore</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Validation of database</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>User access management</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Interacting with Firebird 2.5 trace service</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>DataBase Activity Monitoring</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Monitoring of per-component SQL execution</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Additional Components</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Advanced connection dialog</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Feature</td>
<td>Professional Edition</td>
<td>Community Edition</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Data encryption and decryption</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>TIBCEncryptor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data storing in memory table</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TVirtualTable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dataset that wraps arbitrary non-tabular data</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TVirtualDataSet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced DBGrid with extended functionality</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TCRDBGrid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Records transferring between datasets</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>TCRBatchMove</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Design-Time Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Professional Edition</th>
<th>Community Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhanced component and property editors</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Migration Wizard</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DataSet Manager</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Cross IDE Support

<table>
<thead>
<tr>
<th>Feature</th>
<th>Professional Edition</th>
<th>Community Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lazarus and Free Pascal Support</td>
<td>X</td>
<td>SRC</td>
</tr>
</tbody>
</table>

1 Available only in Professional Edition with Source Code.

© 1997-2021 Devart. All Rights Reserved.

2.9 **Licensing**

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

**INTRODUCTION**

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 IBDAC 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 IBDAC 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 IBDAC. You can distribute IBDAC only in compiled Executable Programs or Dynamic-Link Libraries with required run-time libraries and packages.

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

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

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

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

7. WARRANTY

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

8. SUBSCRIPTION AND SUPPORT

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

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

9. COPYRIGHT

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

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

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

Support Options

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

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

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

IBDAC Priority Support

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

To get help through the IBDAC 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, C++Builder you are using.
- Your IBDAC Registration number.
- Full IBDAC edition name and version number. You can find both of these from the IBDAC | IBDAC About menu in the IDE.
- Versions of the InterBase server and client you are using.
- A detailed problem description.
- If possible, a small test project that reproduces the problem. It is recommended to use Scott or SYS schema objects only. Please include definitions for all and avoid using third-party components.

© 1997-2021 Devart. All Rights Reserved.  Request Support  DAC Forum  Provide Feedback
2.11 Frequently Asked Questions

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

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

Installation and Deployment

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

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

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

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

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

The minimal configuration of client installation includes the following steps:

- Copy the client file gds32.dll to the folder available for executable unit of your program. For example, to the folder with your executable file, or to the Windows system folder. For more information, see description of the LoadLibrary function and the environment variable PATH.

- Add the "gds_db 3050/tcp" line to the services file in the %WinDir%\system32\drivers \etc directory.
  For Firebird version 1.0.0.338 and higher, both client and server use port 3050 by default. So, you do not need to modify the services file. You can also specify port number for the Firebird client in connection string - server/3050:c:\dir\data.gdb

- Copy file InterBase.msg (or firebird.msg for Firebird) to the folder available for executable unit of your program. File must belong to the same version as InterBase or Firebird.

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

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

2. Can I create components using IBDAC?

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

3. What licensing changes can I expect with IBDAC 2.00?

The basic IBDAC license agreement will remain the same. With IBDAC 2.00, the IBDAC Edition Matrix will be reorganized and a new IBDAC Subscription Program will be introduced.

4. What do the IBDAC 2.00 Edition Levels correspond to?

IBDAC 2.00 will come in three editions: Trial, Professional, and Professional with Sources. When you upgrade to the new version, your edition level will be automatically updated using the following Edition Correspondence Table.

<table>
<thead>
<tr>
<th>Old Edition Level</th>
<th>New Edition Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBDAC Standard Edition</td>
<td>IBDAC Professional Edition</td>
</tr>
<tr>
<td>IBDAC Professional Edition</td>
<td>IBDAC Professional Edition with Sources</td>
</tr>
<tr>
<td>IBDAC Trial Edition</td>
<td>IBDAC Trial Edition</td>
</tr>
</tbody>
</table>

The feature list for each edition can be found in the IBDAC documentation and on the IBDAC website.

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

After IBDAC 2.00, all upgrades to future versions are free to users with an active IBDAC
Subscription.

Users that have a registration for versions of IBDAC prior to IBDAC 2.00 will have to first upgrade to IBDAC 2.00 to jump in on the Subscription program.

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

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

Users with a valid IBDAC Subscription get the following benefits:

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

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

The IBDAC Subscription Program is available for registered users of IBDAC 2.00 and higher.

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

Yes, you can. IBDAC version licenses are perpetual.

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

An annual IBDAC Subscription is included when ordering or upgrading to any registered (non-Trial) edition of IBDAC 2.00 or higher.

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

9. Does this mean that if I upgrade to IBDAC 2 from IBDAC 1, I'll get an annual IBDAC Subscription for free?

Yes.

10. How do I upgrade to IBDAC 2.00?

To upgrade to IBDAC 2.00, you can get a Version Update from the IBDAC Ordering Page. For more information, please contact sales@devart.com.
Performance

1. How productive is IBDAC?

IBDAC uses low-level protocol to access the database server. This allows IBDAC to achieve high performance. From time to time we compare IBDAC with other products, and IBDAC always takes first place. For more information refer to online test results.

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

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

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

How To

1. How can I enable syntax highlighting in IBDAC component editors at design time?

To enable syntax highlighting for IBDAC, you should download and install the freeware SynEdit component set.

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

You can determine your IBDAC version number in several ways:

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

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

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

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

The values of these properties are templates for query statements, and they cannot be manually executed. Usually there is no need to fill these properties because the text of the query is generated automatically.
In special cases, you can set these properties to perform more complicated processing during a query. These properties are automatically processed by IBDAC during the execution of the Post, Delete, or RefreshRecord methods, and are used to construct the query to the server. Their values can contain parameters with names of fields in the underlying data source, which will be later replaced by appropriate data values.

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

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

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

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

5. Some questions about the visual part of IBDAC

The following questions usually arise from the same problem:

- I set the Debug property to True but nothing happens!
- While executing a query, the screen cursor does not change to an hour-glass.
- Even if I have LoginPromp set to True, the connect dialog does not appear.

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

General Questions

1. I would like to develop an application that works with InterBase Server. Which should I use - IBDAC or dbExpress?

dbExpress technology serves for providing a more or less uniform way to access different servers (SQL Server, MySQL, Oracle and so on). It is based on drivers that include server-specific features. Like any universal tool, in many specialized cases dbExpress providers lose some functionality. For example, the dbExpress design-time is quite poor and cannot be expanded.
IBDAC is a specialized set of components to access InterBase server with advanced design-time and component interface similar to BDE.

We tried to implement maximal InterBase support in IBDAC. dbExpress technology puts severe restrictions. For example, Unicode fields cannot be passed from the driver to dbExpress.

In some cases dbExpress is slower because data undergoes additional conversion to correspond to dbExpress standards.

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

2. Why use IBDAC instead of standard InterBase Express components?

There are many reasons why IBDAC is better than IBExpress. Some of them are enumerated here. For more information refer to IBDAC features list.

- Reliable user support - we help to solve common issues quickly using e-mail or dedicated forum.
- IBDAC is being constantly improved and enhanced, so you can be sure that the product is always up-to-date with the latest InterBase data access technology advances.
- Better support for BLOBs, Arrays and other advanced features of the databases.
- Automatic generation of SQL UPDATE, INSERT, DELETE, LOCK statements, so that you do not need to care about routine tasks.
- Ability to lock records automatically, which helps you build stable multiuser applications more easily.
- Unicode and national charsets support in all IBDAC components
- IBDAC shares the same troubleproof engine with the other famous DAC products - ODAC, MyDAC, and SDAC. So if you have worked with one of them, it will be easier for you to switch to another one if you ever need to integrate support for another database server in your application.

3. Are the IBDAC connection components thread-safe?

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

4. Behaviour of my application has changed when I upgraded IBDAC. How can I
restoring the old behaviour with the new version?

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

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

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

6. I cannot use INT64 fields as key fields in master-detail relationship.

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

7. Can IBDAC and BDE functions be used side-by-side in a single application?

Yes. There is no problem with using both IBDAC and BDE functions in the same application.
• Installing IBDAC.
• Working with the IBDAC demo projects.
• Compiling and deploying your IBDAC project.
• Using the IBDAC documentation.
• How to get help with IBDAC.

What is IBDAC?

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

Many IBDAC classes are based on VCL, LCL and FireMonkey classes and interfaces. IBDAC is a replacement for the Borland Database Engine and InterBase Express, provides native database connectivity, and is specifically designed as an interface to the InterBase and Firebird databases.

An introduction to IBDAC is provided in the Overview section.

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

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

Installing IBDAC

To install IBDAC, complete the following steps.

1. Choose and download the version of the IBDAC installation program that is compatible with your IDE. For instance, if you are installing IBDAC 2.00, you should use the following files:

   For BDS 2006 and Turbo - ibdac200d10*.exe
   For Delphi 7 - ibdac200d7*.exe

   For more information, visit the IBDAC download page.

2. Close all running IDE's.

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

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

To check that IBDAC has been installed properly, launch your IDE and make sure that an IBDAC page has been added to the Component palette and that an IBDAC menu was added to the Menu bar.
If you have bought IBDAC Professional Edition with Source Code, you will be able to download both the compiled version of IBDAC and the IBDAC source code. The installation process for the compiled version is standard, as described above. The IBDAC source code must be compiled and installed manually. Consult the supplied ReadmeSrc.html file for more details.

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

### Working with the IBDAC demo projects

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

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

**IbDacDemo Walkthrough**

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

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

\Program Files\Devart\IBDAC for Delphi 2006\Demos\Win32\IbDacDemo
\IbDacDemo.bdsproj

4. Select Run | Run or press F9 to compile and launch the demo project. *IbDacDemo* should start, and a full-screen IBDAC Demo window with a toolbar, an explorer panel, and a view panel will open. The explorer panel will contain a list of all the demo sub-projects included in *IbDacDemo*, 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 InterBase. However, you will not be able to actually retrieve data from InterBase or execute commands until you connect to the database.
5. Click on the "Connect" button in the IbDacDemo toolbar. A Connect dialog box will open. Enter the connection parameters you use to connect to your InterBase server and click "Connect" in the dialog box.

**Note:** For this step to work properly, you must have the InterBase Client installed.

Now you have a fully functional interface to your InterBase server. You will be able to go through the different demos, to browse tables, create and drop objects, and execute DSQL 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 the objects that will be used by IbDacDemo. If some of these objects already exist in the database you have connected to, the following error message will appear.

"An error has occurred:
unsuccessful metadata update Table DEPT already exists

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

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

Ignore this exception?"

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

7. Choose a demo that demonstrates an aspect of working with InterBase 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 InterBase tables, select the Table demo from the "Working with Components" folder. A simple InterBase 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 IbDacDemo 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 IBDAC components are used to connect to the database.

9. Click on the "Form as text" button in the IbDacDemo toolbar to view the code behind the interface to the demo. Try to find the places where IBDAC components are created on the
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 IBDAC components.

**General demos**

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

**InterBase-specific demos**

A collection of projects that demonstrate how to incorporate InterBase/Firebird features in database applications.

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

**Other IBDAC demo projects**

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

**Compiling and deploying your IBDAC project**

**Compiling IBDAC-based projects**

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

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

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

**Deploying IBDAC-based projects**
To deploy an application that uses IBDAC, you will need to make sure the target workstation has access to the following files:

- The InterBase Client software, if connecting in Client mode.
- The IBDAC bpl files, if compiling with runtime packages.

If you are evaluating deploying projects with IBDAC 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 IBDAC Trial Edition for C++Builder will only work if the C++Builder IDE is launched. More information about IBDAC Trial Edition limitations is provided here.

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

**Using the IBDAC documentation**

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

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

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

**How to get help with IBDAC**

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

- If you have any questions about IBDAC installation or licensing, consult the Licensing and FAQ sections.
- You can get community assistance and IBDAC technical support on the IBDAC Support Forum.
- To get help through the IBDAC Priority Support program, send an email to the IBDAC development team at ibdac@devart.com.
3.1 Installation

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

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

Before installing IBDAC ...

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

So before installing a new version of IBDAC, uninstall any previous version of IBDAC 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 ibdac@devart.com

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

Installed packages

Note: %IBDAC% denotes the path to your IBDAC installation directory.
Delphi/C++Builder Win32 project packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>dacXX.bpl</td>
<td>DAC run-time package</td>
<td>Windows\System32</td>
</tr>
<tr>
<td>dcldacXX.bpl</td>
<td>DAC design-time package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>dacvclXX.bpl*</td>
<td>DAC VCL support package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>ibdacXX.bpl</td>
<td>IBDAC run-time package</td>
<td>Windows\System32</td>
</tr>
<tr>
<td>dclibdacXX.bpl</td>
<td>IBDAC design-time package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>ibdacvclXX.bpl*</td>
<td>VCL support package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>crcontrolsXX.bpl</td>
<td>TCRDBGGrid component</td>
<td>Delphi\Bin</td>
</tr>
</tbody>
</table>

Additional packages for using IBDAC managers and wizards

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasetmanagerXX.bpl</td>
<td>DataSet Manager package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>oramigwizardXX.dll</td>
<td>IBDAC BDE\IBX Migration wizard</td>
<td>%IBDAC%\Bin</td>
</tr>
</tbody>
</table>

Environment Changes

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

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

Delphi

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

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

C++Builder

C++Builder 6:

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

C++Builder 2006, 2007:
- \$(BCB)\IBDAC\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)\IBDAC\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 IBDAC Installer performs C++Builder environment changes automatically for compiled versions of IBDAC.

Lazarus

The IBDAC installation program only copies IBDAC files. You need to install IBDAC packages to Lazarus IDE manually. Open %IBDAC%\Source\Lazarus1\dclibdac10.lpk (for Trial version %IBDAC%\Packages\dclibdac10.lpk) file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilt with IBDAC packages.

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

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

Installation of Additional Components and Add-ins

DBMonitor

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

© 1997-2021 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback

3.2 Migrating from BDE and IBX

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

Migration Wizard allows you to convert your BDE or IBX projects to IBDAC. This wizard
replaces BDE or IBX components in a specified project (.dfm and .pas files) with IBDAC components.

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

After the project conversion it will be reopened.

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

If some problems occur while making changes, you can restore your project from backup file. To do this, perform the following steps.
- Select **Migration Wizard** from the **IBDAC** menu
- Select Restore original files from backup and press the Next button.
- Select the backup file. By default it is RExpert.reu file in RBackup folder of your converted project. Press the Next button.
- Check your settings and press the Finish button to start the conversion operation.
- Press Yes in the dialog that appeared.

Your project will be restored to its previous state.

### 3.3 Connecting to InterBase and Firebird

This tutorial describes how to connect to InterBase and Firebird using the **TIBCConnection** component.
1. **Requirements**

2. **General Information**

3. **Creating a Connection**
   - 3.1 **Connecting at Design-Time**
     - 3.1.1 **Using TIBCConnection Editor**
     - 3.1.2 **Using Object Inspector**
   - 3.2 **Connecting at Runtime**

4. **Opening a Connection**

5. **Modifying a Connection**

6. **Closing a Connection**

7. **Additional Information**

8. **See Also**

### Requirements

This tutorial assumes that you have installed IBDAC and run the database server and the IDE. You need to know the server address, the port number (if you use a port other than the default port `3850`), the path to the database file (.gdb or .fdb), and the username and password. To connect at runtime, add the IBC unit to the `uses` clause for Delphi or include the `IBC.hpp` header file for C++ Builder.

### General Information

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

### Creating a Connection

#### Connecting at Design-Time

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

1. Find the TIBConnection component in the IBDAC category on the Tool Palette.
2. Double-click the component. A new object will appear on the form. If this is the first TIBConnection object in this unit, it will be named `IBCConnection1`.

Using TIBConnection Editor

1. Double-click the `IBCConnection1` object.
2. Specify the DNS name or IP address of the InterBase or Firebird server in the Server edit box.
3. If you use a port other than the default port 3050, specify it in the Port edit box.
4. Specify the database file path in the Database edit box, e.g., D:\InterBase\employee.gdb or D:\Firebird\employee.fdb.
5. Specify the username (sysdba by default) in the Username edit box.
6. Specify the password (masterkey by default) in the Password edit box.
7. If you have both InterBase and Firebird client libraries installed, specify the path to the client library — gds32.dll for InterBase or fbclient.dll for Firebird — in the Client library edit box. Otherwise, skip this step.

Using Object Inspector
1. Select the IBCConnection1 object on the form.
2. If you have both InterBase and Firebird client libraries installed, set the ClientLibrary property to gds32.dll for InterBase or fbclient.dll for Firebird. Otherwise, skip this step.
3. Set the Database property to the database file path, e.g., D:\InterBase\employee.gdb or D:\Firebird\employee.fdb.
4. Set the Password property to the password (masterkey by default).
5. If you use a port other than the default port 3050, set the Port property to your port.
6. Set the Server property to the DNS name or IP address of the InterBase or Firebird server.
7. Set the Username property to the username (sysdba by default).

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

Delphi

```delphi
var
    IBCConnection1: TIBCConnection;
begin
    IBCConnection1 := TIBCConnection.Create(nil);
    try
        // adds connection parameters
        // if Server is empty, a connection is established through the local process
        IBCConnection1.Server := 'server';
        IBCConnection1.Database := 'database';
        IBCConnection1.Username := 'username';
        IBCConnection1.Password := 'password';
        IBCConnection1.Port := 3050;
        // indicates the client lib for InterBase (for Firebird, use fbclient.dll)
        IBCConnection1.ClientLibrary := 'gds32.dll';
        // disables a login prompt
        IBCConnection1.LoginPrompt := False;
        // opens a connection
        IBCConnection1.Open;
    finally
```
Opening a Connection

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

**Delphi**

```delphi
IBConnection1.Open;
```

**C++ Builder**

```cpp
IBConnection1->Open;
```

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

**Delphi**

```delphi
IBConnection1.Connected := True;
```

**C++ Builder**

```cpp
IBConnection1->Connected = True;
```

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

Modifying a Connection

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

**Closing a Connection**

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

**Delphi**

```
IBCConnection1.Close;
```

or:

```
IBCConnection1.Connected := False;
```

**C++ Builder**

```
IBCConnection1->Close;
```

or:

```
IBCConnection1->Connected = False;
```

**Additional Information**

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

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

**See Also**

- `TIBCConnection`
- `Server`
- `Port`
- `Database`
- `Username`
- `Password`
- `LoginPrompt`
- `ConnectString`
3.4 Deleting Data From Tables

This tutorial describes how to delete data from a table using the TIBCQuery and TIBCTable components.

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

Requirements

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

General Information

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

Using the DataSet Functionality

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

Delphi

```delphi
var
```
Building DML Statements Manually

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

Parameterized DML Statements

Delphi

```delphi
var
  IBCQuery1: TIBCQuery;
begin
  IBCQuery1 := TIBCQuery.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCQuery1.Connection := IBCConnection1;
    // adds a statement to retrieve data
    IBCQuery1.SQL.Text := 'SELECT * FROM dept';
    // opens the dataset
    IBCQuery1.Open;
    // deletes the active record
    IBCQuery1.Delete;
  finally
    IBCQuery1.Free;
  end;
end;
```

C++Builder

```cpp
TIBCQuery* IBCQuery1 = new TIBCQuery(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCQuery1->Connection = IBCConnection1;
  // adds a statement to retrieve data
  IBCQuery1->SQL->Text = "SELECT * FROM dept";
  // opens the dataset
  IBCQuery1->Open();
  // deletes the active record
  IBCQuery1->Delete();
}
finally {
  IBCQuery1->Free();
}
```
Non-Parameterized DML Statements

Delphi

var  
  IBCQuery1: TIBCQuery;
begin    
  IBCQuery1 := TIBCQuery.Create(nil);
  try    
  // IBCConnection1 was set up earlier    
  IBCQuery1.Connection := IBCConnection1;
  // adds a statement to delete a record    
  IBCQuery1.SQL.Add('DELETE FROM dept WHERE deptno = 10;');    
  // executes the statement    
  IBCQuery1.Execute;
  finally    
  IBCQuery1.Free;
  end;
end;

C++Builder

TIBQuery* IBCQuery1 = new TIBCQuery(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCQuery1->Connection = IBCConnection1;
  // adds a statement to delete a record
  IBCQuery1->SQL->Add("DELETE FROM dept WHERE deptno = :deptno;");
  // executes the statement
  IBCQuery1->Execute();
} finally {
  IBCQuery1->Free();
}
IBCQuery1->Free();
}

Additional Information

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

3.5 Creating Database Objects

This tutorial describes how to create database objects in InterBase and Firebird using the TIBCSQL and TIBCScript components.

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

Requirements

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

General Information

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

Creating Tables
To create tables, the `TIBSQL` component is used in this tutorial.

**Design-Time**
- Find the `TIBSQL` component in the IBDAC category on the Tool Palette.
- Double-click the component. A new object will appear on the form. If this is the first `TIBSQL` object in this project, it will be named `IBCSQL1`. Note that the `Connection` property is automatically set to an existing connection.
- Double-click the `IBCSQL1` object.
- Enter the following statements:

```sql
CREATE TABLE dept (  
    deptno integer not null primary key,  
    dname varchar(14),  
    loc varchar(13),  
    primary key (deptno)  
);  
CREATE TABLE emp (  
    empno integer not null primary key,  
    ename varchar(10),  
    job varchar(9),  
    mgr integer,  
    hiredate timestamp,  
    sal integer,  
    comm integer,  
    deptno integer references dept (deptno)  
);  
```
- Click the **Execute** button to create two tables.

**Runtime**

The same tables created at runtime:

**Delphi**

```delphi
var  
    IBCSQL1: TIBSQL;begin  
    IBCSQL1 := TIBSQL.Create(nil);  
    try  
        // IBCConnection1 was set up earlier  
        IBCSQL1.Connection := IBCConnection1;  
        // adds statements to create tables  
        IBCSQL1.SQL.Add('CREATE TABLE dept (');  
        IBCSQL1.SQL.Add(' deptno integer not null primary key,');  
        IBCSQL1.SQL.Add(' dname varchar(14),' );  
        IBCSQL1.SQL.Add(' loc varchar(13)');  
```
Creating Stored Procedures

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

**Design-Time**

- Find the **TIBScript** component in the IBDAC category on the Tool Palette.
- Double-click the component. A new object will appear on the form. If this is the first
The TIBScript object in this project, it will be named `TIBCScript1`. Note that the Connection property is already set to an existing connection.

- Double-click the `TIBCScript1` object.
- Enter the following statement:

```sql
CREATE PROCEDURE TenMostHighPaidEmployees
RETURNS (salary integer)
AS
BEGIN
  FOR
    SELECT FIRST 10 emp.sal FROM emp ORDER BY emp.sal DESC INTO salary DO
      suspend;
  END;

CREATE FUNCTION GetEmpNumberInDept (pdeptno integer)
RETURNS integer
AS
BEGIN
  RETURN (SELECT COUNT(*) FROM emp WHERE deptno = :pdeptno);
END;
```

- Click the Execute button to create two stored procedures.

**Runtime**

The same stored procedures created at runtime:

**Delphi**

```delphi
var
  IBCScript1: TIBScript;
begin
  IBCScript1 := TIBScript.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCScript1.Connection := IBCConnection1;
    // adds statements to create procedures
    IBCScript1.SQL.Add('CREATE PROCEDURE TenMostHighPaidEmployees');
    IBCScript1.SQL.Add('RETURNS (salary integer)');
    IBCScript1.SQL.Add('AS');
    IBCScript1.SQL.Add('BEGIN');
    IBCScript1.SQL.Add('FOR');
    IBCScript1.SQL.Add('SELECT FIRST 10 emp.sal FROM emp ORDER BY emp.sal DESC INTO salary DO suspend;');
    IBCScript1.SQL.Add('END;');
    IBCScript1.SQL.Add('CREATE FUNCTION GetEmpNumberInDept (pdeptno integer)');
    IBCScript1.SQL.Add('RETURNS integer');
    IBCScript1.SQL.Add('AS');
  except
    .....
  end;
```
Getting Started

```
IBCScript1.SQL.Add('BEGIN');
IBCScript1.SQL.Add('  RETURN (SELECT COUNT(*) FROM emp WHERE deptno = :pdeptno);');
IBCScript1.SQL.Add('END');
// executes the statements
IBCScript1.Execute;
finally
    IBCScript1.Free;
end;
end;

C++Builder

TIBCScript* IBCScript1 = new TIBCScript(NULL);
try {
    // IBCConnection1 was set up earlier
    IBCScript1->Connection = IBCConnection1;
    // adds statements to create procedures
    IBCScript1->SQL->Add("CREATE PROCEDURE TenMostHighPaidEmployees");
    IBCScript1->SQL->Add("RETURNS (salary integer)");
    IBCScript1->SQL->Add("AS");
    IBCScript1->SQL->Add("BEGIN");
    IBCScript1->SQL->Add(" FOR");
    IBCScript1->SQL->Add(" SELECT FIRST 10 emp.sal FROM emp ORDER BY emp.sal");
    IBCScript1->SQL->Add("( suspend;");
    IBCScript1->SQL->Add("END");
    IBCScript1->SQL->Add(""");
    IBCScript1->SQL->Add("CREATE FUNCTION GetEmpNumberInDept ("");
    IBCScript1->SQL->Add(" pdeptno integer")");
    IBCScript1->SQL->Add("AS");
    IBCScript1->SQL->Add("BEGIN");
    IBCScript1->SQL->Add(" RETURN (SELECT COUNT(*) FROM emp WHERE deptno =");
    IBCScript1->SQL->Add("END");
    // executes the statements
    IBCScript1->Execute;
}
__finally {
    IBCScript1->Free();
}
```

Additional Information

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

This tutorial describes how to insert data into tables using the TIBCQuery and TIBCTable components.

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

Requirements

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

General Information

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

The dept table definition:

<table>
<thead>
<tr>
<th>deptno</th>
<th>dname</th>
<th>loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
</tr>
</tbody>
</table>
### The emp Table Definition

The `emp` table definition:

<table>
<thead>
<tr>
<th>emp no</th>
<th>ename</th>
<th>job</th>
<th>mgr</th>
<th>hiredate</th>
<th>sal</th>
<th>comm</th>
<th>dept no</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMIT H</td>
<td>CLE RK</td>
<td>7902</td>
<td>17-12-1980</td>
<td>800</td>
<td>NULL</td>
<td>20</td>
</tr>
<tr>
<td>7499</td>
<td>ALE N</td>
<td>SALE SMAN</td>
<td>7698</td>
<td>20-02-1981</td>
<td>1600</td>
<td>300</td>
<td>30</td>
</tr>
<tr>
<td>7521</td>
<td>WAR D</td>
<td>SALE SMAN</td>
<td>7698</td>
<td>22-02-1981</td>
<td>1250</td>
<td>500</td>
<td>30</td>
</tr>
<tr>
<td>7566</td>
<td>JONE S</td>
<td>MAN AGE R</td>
<td>7839</td>
<td>02-04-1981</td>
<td>2975</td>
<td>NULL</td>
<td>20</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>SALE SMAN</td>
<td>7698</td>
<td>28-09-1981</td>
<td>1250</td>
<td>1400</td>
<td>30</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE E</td>
<td>MAN AGE R</td>
<td>7839</td>
<td>01-05-1981</td>
<td>2850</td>
<td>NULL</td>
<td>30</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>MAN AGE R</td>
<td>7839</td>
<td>09-06-1981</td>
<td>2450</td>
<td>NULL</td>
<td>10</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANAL YST</td>
<td>7566</td>
<td>13-07-1987</td>
<td>3000</td>
<td>NULL</td>
<td>20</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>PRES IDE N</td>
<td>NULL</td>
<td>17-11-1981</td>
<td>5000</td>
<td>NULL</td>
<td>10</td>
</tr>
<tr>
<td>7844</td>
<td>TURN ER</td>
<td>SALE SMAN</td>
<td>7698</td>
<td>08-09-1981</td>
<td>1500</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>CLE RK</td>
<td>7788</td>
<td>13-07-1987</td>
<td>1100</td>
<td>NULL</td>
<td>20</td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>CLE RK</td>
<td>7698</td>
<td>03-12-1981</td>
<td>950</td>
<td>NULL</td>
<td>30</td>
</tr>
</tbody>
</table>
Inserting at Design-Time

- Find the TIBCQuery component in the IBDAC category on the Tool Palette.
- Double-click the component. A new object will appear on the form. If this is the first time that you create a TIBCQuery object in this application, the object will be named IBCQuery1. Note that the IBCQuery1.Connection property is automatically set to an existing connection.
- Double-click the IBCQuery1 object.
- Enter the following statement:

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

Inserting at Runtime

Using the DataSet Functionality

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

Delphi

```delphi
var
  IBCQuery1: TIBCQuery;
begin
  IBCQuery1 := TIBCQuery.Create(nil);
  try // IBCConnection1 was set up earlier
    IBCQuery1.ConnectionString := IBCConnection1;
    // adds a statement to retrieve data
    IBCQuery1.SQL.Text := 'SELECT * FROM dept';
    // opens the dataset
    IBCQuery1.Open;
    // adds a new empty record at the end of the dataset
    IBCQuery1.Append;
    // searches fields by their names and assigns new values
    IBCQuery1.FieldName('deptno').AsInteger := 10;
  except
    on E: Exception do
      Writeln(E.Classname, ': ', E.Message);
  end;
end.
```
IBCQuery1.FieldName('dname').AsString := 'ACCOUNTING';
IBCQuery1.FieldName('loc').AsString := 'NEW YORK';
// writes the modified record
IBCQuery1.Post;
finally
  IBCQuery1.Free;
end;
end;

C++Builder

TIBQuery* IBCQuery1 = new TIBQuery(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCQuery1->Connection = IBCConnection1;
  // adds a statement to retrieve data
  IBCQuery1->SQL->Text = "SELECT * FROM dept";
  // opens the dataset
  IBCQuery1->Open();
  // adds a new empty record at the end of the dataset
  IBCQuery1->Append();
  // searches fields by their names and assigns new values
  IBCQuery1->FieldName("deptno")->AsInteger = 10;
  IBCQuery1->FieldName("dname")->AsString = "ACCOUNTING";
  IBCQuery1->FieldName("loc")->AsString = "NEW YORK";
  // writes the modified record
  IBCQuery1->Post();
}__finally {
  IBCQuery1->Free();
}

Building DML Statements Manually

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

Parameterized DML Statements

Delphi

var
  IBCQuery1: TIBQuery;
begin
  IBCQuery1 := TIBQuery.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCQuery1.Connection := IBCConnection1;
    // adds a parameterized statement to insert data
    IBCQuery1.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (:deptno, ');
    // searches parameters by their names and assigns new values
    IBCQuery1.ParamByName('deptno').AsInteger := 10;
    IBCQuery1.ParamByName('dname').AsString := 'ACCOUNTING';
    IBCQuery1.ParamByName('loc').AsString := 'NEW YORK';
    // executes the statement
C++Builder

TIBCQuery* IBCQuery1 = new TIBCQuery(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCQuery1->Connection = IBCConnection1;
  // adds a parameterized statement to insert data
  IBCQuery1->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (:deptno, :dname, :loc)";
  // searches parameters by their names and assigns new values
  IBCQuery1->ParamByName("deptno")->AsInteger = 10;
  IBCQuery1->ParamByName("dname")->AsString = "ACCOUNTING";
  IBCQuery1->ParamByName("loc")->AsString = "NEW YORK";
  // executes the statement
  IBCQuery1->Execute();
} __finally {
  IBCQuery1->Free();
}

Delphi

var
  IBCQuery1: TIBCQuery;
begin
  IBCQuery1 := TIBCQuery.Create(nil);
  try // IBCConnection1 was set up earlier
    IBCQuery1.Connection := IBCConnection1;
    // adds a statement to insert a record
    IBCQuery1.SQL.Add("INSERT INTO dept(deptno, dname, loc) VALUES (10,"ACCOUNTING","NEW YORK")
    // executes the statement
    IBCQuery1.Execute;
  finally
    IBCQuery1.Free;
  end;
end;

C++Builder

TIBCQuery* IBCQuery1 = new TIBCQuery(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCQuery1->Connection = IBCConnection1;
  // adds the statement to insert a record
  IBCQuery1->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (10,"ACCOUNTING","NEW YORK")
  // executes the statement
  IBCQuery1->Execute();
}
__finally {
    IBCQuery1->Free();
}

Additional Information

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

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

3.7 Retrieving Data From Tables

This tutorial describes how to retrieve data from a table using the TIBCQuery and TIBCTable components.

1. Requirements
2. General Information
3. TIBCQuery
4. TIBCTable
5. Additional Information

Requirements

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

General Information

IBDAC provides the TIBCQuery and TIBCTable components for retrieving data from a table. This tutorial shows you how to retrieve data from the dept table.

© 2021 Devart
TIBCQuery

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

**Delphi**

```delphi
var
  IBCQuery1: TIBCQuery;
begin
  IBCQuery1 := TIBCQuery.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCQuery1.Connection := IBCConnection1;
    // adds a statement to retrieve data
    IBCQuery1.SQL.Text := 'SELECT * FROM dept';
    // opens the dataset
    IBCQuery1.Open;
    // shows the number of records in the dataset
    ShowMessage(IntToStr(IBCQuery1.RecordCount));
  finally
    IBCQuery1.Free;
  end;
end;
```

**C++Builder**

```cpp
TIBCQuery* IBCQuery1 = new TIBCQuery(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCQuery1->Connection = IBCConnection1;
  // adds a statement to retrieve data
  IBCQuery1->SQL->Text = "SELECT * FROM dept";
  // opens the dataset
  IBCQuery1->Open();
  // shows the number of records in the dataset
  ShowMessage(IntToStr(IBCQuery1->RecordCount));
} finally {
  IBCQuery1->Free();
}
```

TIBCTable

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

**Delphi**

```delphi
var
  IBCTable1: TIBCTable;
begin
  IBCTable1 := TIBCTable.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCTable1.Connection := IBCConnection1;
```

© 2021 Devart
// indicates the name of the table
IBCTable1.TableName := 'dept';
// opens the dataset
IBCTable1.Open;
// shows the number of records in the dataset
ShowMessage(IntToStr(IBCTable1.RecordCount));
finally
  IBCTable1.Free;
end;
end;

C++Builder

TIBCTable* IBCTable1 = new TIBCTable(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCTable1->Connection = IBCConnection1;
  // indicates the name of the table
  IBCTable1->TableName = "dept";
  // opens the dataset
  IBCTable1->Open();
  // shows the number of records in the dataset
  ShowMessage(IntToStr(IBCTable1->RecordCount));
}
finally {
  IBCTable1->Free();
}

Additional Information

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

3.8 Modifying Data in Tables

This tutorial describes how to modify data in tables using the TIBCQuery and TIBCTable
components.

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

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

General Information

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

10 ACCOUNTING NEW YORK
to change it to:

10 RESEARCH LOS ANGELES

Using the DataSet Functionality

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

Delphi

```delphi
var
  IBCQuery1: TIBCQuery;
beg
  IBCQuery1 := TIBCQuery.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCQuery1.Connection := IBCConnection1;
    // adds a statement to retrieve data
    IBCQuery1.SQL.Text := 'SELECT * FROM dept';
    // opens the dataset
    IBCQuery1.Open;
    // positions the cursor on the deptno=10 record
    IBCQuery1.FindKey([10]);
    // enables editing of data in the dataset
    IBCQuery1.Edit;
    // searches fields by their names and assigns new values
```
IBCQuery1.FieldByName('dname').AsString := 'RESEARCH';
IBCQuery1.FieldByName('loc').AsString := 'LOS ANGELES';
// writes the modified record
IBCQuery1.Post;
finally
  IBCQuery1.Free;
end;
end;

C++Builder

TIBCQuery* IBCQuery1 = new TIBCQuery(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCQuery1->Connection = IBCConnection1;
  // adds a statement to retrieve data
  IBCQuery1->SQL->Text = "SELECT * FROM dept";
  // opens the dataset
  IBCQuery1->Open();
  // positions the cursor on the deptno=10 record
  IBCQuery1->FindKey(ARRAYOFCONST((10)));
  // enables editing of data in the dataset
  IBCQuery1->Edit();
  // searches fields by their names and assigns new values
  IBCQuery1->FieldByName("dname")->AsString = "RESEARCH";
  IBCQuery1->FieldByName("loc")->AsString = "LOS ANGELES";
  // writes the modified record
  IBCQuery1->Post();
}__finally {
  q->Free();
}

Building DML Statements Manually

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

Parameterized DML Statements

Delphi

var
  IBCQuery1: TIBCQuery;
begin
  IBCQuery1 := TIBCQuery.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCQuery1.Connection := IBCConnection1;
    // adds a statement to update a record
    IBCQuery1.SQL.Add('UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = 10;
    // searches parameters by their names and assigns new values
    IBCQuery1.ParamByName('deptno').AsInteger := 10;
    IBCQuery1.ParamByName('dname').AsString := 'RESEARCH';
    IBCQuery1.ParamByName('loc').AsString := 'LOS ANGELES';
  except
  ...
// executes the statement
IBCQuery1.Execute;
finally
  IBCQuery1.Free;
end;
end;

C++Builder

TIBCQuery* IBCQuery1 = new TIBCQuery(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCQuery1->Connection = IBCConnection1;
  // adds a statement to update a record
  IBCQuery1->SQL->Add("UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = 10");
  // searches parameters by their names and assigns new values
  IBCQuery1->ParamByName("deptno")->AsInteger = 10;
  IBCQuery1->ParamByName("dname")->AsString = "RESEARCH";
  IBCQuery1->ParamByName("loc")->AsString = "LOS ANGELES";
  // executes the statement
  IBCQuery1->Execute();
} finally {
  IBCQuery1->Free();
}

Non-Parameterized DML Statements

Delphi

var
  IBCQuery1: TIBCQuery;
begin
  IBCQuery1 := TIBCQuery.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCQuery1.Connection := IBCConnection1;
    // adds the statement to update a record
    IBCQuery1.SQL.Add('UPDATE dept SET dname = 'RESEARCH', loc = 'LOS ANGELES' WHERE deptno = 10');
    // executes the statement
    IBCQuery1.Execute;
  finally
    IBCQuery1.Free;
  end;
end;

C++Builder

TIBCQuery* IBCQuery1 = new TIBCQuery(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCQuery1->Connection = IBCConnection1;
  // adds a statement to update a record
  IBCQuery1->SQL->Add("UPDATE dept SET dname = 'RESEARCH', loc = 'LOS ANGELES' WHERE deptno = 10");
  // executes the statement
  IBCQuery1->Execute();
}
Additional Information

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

3.9 Using Stored Procedures

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

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

Requirements

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

General Information

A stored procedure is a group of one or more SQL statements grouped as a logical unit and stored in the database. Stored procedures are intended to perform a specific task or a set of related tasks. They combine the ease and flexibility of the SQL language with the procedural functionality of a structured programming language. Complicated business rules and programming logic that may require execution of multiple SQL statements should be kept in stored procedures, which can be called by the client applications.
A stored function is similar to a stored procedure, but there are some differences: a function must return a value, whereas in a stored procedure it is optional; a function can have only input parameters, whereas a procedures can have input or output parameters; a function can be called from a procedure, whereas a procedure cannot be called from a function.

**Input parameters**

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

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

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

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

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

**Delphi**

```delphi
donete
var
  IBCStoredProc1: TIBCStoredProc;
begin
  IBCStoredProc1 := TIBCStoredProc.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCStoredProc1.Connection := IBCConnection1;
    // indicates the name of the stored procedure to call
    IBCStoredProc1.StoredProcedure := 'InsertDept';
    // constructs a statement based on the Params and StoredProcName // properties, and assigns it to the SQL property
    IBCStoredProc1.PreparedStatement;
    // searches parameters by their names and assigns new values
    IBCStoredProc1.ParamByName('p_deptno').AsInteger := 10;
    IBCStoredProc1.ParamByName('p_dname').AsString := 'ACCOUNTING';
    IBCStoredProc1.ParamByName('p_loc').AsString := 'NEW YORK';
    // executes the stored procedure
    IBCStoredProc1.Execute;
  finally
end;
```
C++Builder

TIBCStoredProc* IBCStoredProc1 = new TIBCStoredProc(NULL);
try {
    // IBCConnection1 was set up earlier
    IBCStoredProc1->Connection = IBCConnection1;
    // indicates the name of the stored procedure to call
    IBCStoredProc1->StoredProcName = "InsertDept";
    // constructs a statement based on the Params and StoredProcName properties, and assigns it to the SQL property
    IBCStoredProc1->PrepareSQL();
    // searches parameters by their names and assigns new values
    IBCStoredProc1->ParamByName("p_deptno") -> AsInteger = 10;
    IBCStoredProc1->ParamByName("p_dname") -> AsString = "ACCOUNTING";
    IBCStoredProc1->ParamByName("p_loc") -> AsString = "NEW YORK";
    // executes the stored procedure
    IBCStoredProc1->Execute();
}
finally {
    IBCStoredProc1->Free();
}

Output Parameters

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

CREATE PROCEDURE CountDept
RETURNS (cnt integer)
BEGIN
    SELECT count(*) FROM dept into cnt;
END;

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

Delphi

var
    IBCStoredProc1: TIBCStoredProc;
begin
    IBCStoredProc1 := TIBCStoredProc.Create(nil);
    try
        // IBCConnection1 was set up earlier
        IBCStoredProc1.Connection := IBCConnection1;
        // indicates the name of the stored procedure to call
Input/output parameters

A stored procedure that contains input and output parameters can both accept and return values. Programs can pass a value to the stored procedure, which does something under the hood, and passes the resulting value back to the calling program. The input value must be set before executing the stored procedure. The output value is returned after executing the stored procedure.

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

```sql
CREATE PROCEDURE GiveBonus(sal float) 
RETURNS (bonus float) 
AS
    bonus = sal * 1.05;
END;
```

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

var
  IBCStoredProc1: TIBCStoredProc;
begin
  IBCStoredProc1 := TIBCStoredProc.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCStoredProc1.Connection := IBCConnection1;
    // indicates the name of the stored procedure to call
    IBCStoredProc1.StoredProcName := 'GiveBonus';
    // constructs a statement based on the Params and StoredProcName
    // properties, and assigns it to the SQL property
    IBCStoredProc1.PrepareSQL;
    // searches a parameter by its name and assigns a new value
    IBCStoredProc1.ParamByName('sal').AsFloat := 500.5;
    // executes the stored procedure
    IBCStoredProc1.Execute;
    // shows the resulting value
    ShowMessage(IBCStoredProc1.ParamByName('sal').AsString);
  finally
    IBCStoredProc1.Free;
  end;
end;

C++Builder

TIBCStoredProc* IBCStoredProc1 = new TIBCStoredProc(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCStoredProc1->Connection = IBCConnection1;
  // indicates the name of the stored procedure to call
  IBCStoredProc1->StoredProcName = "GiveBonus";
  // constructs a statement based on the Params and StoredProcName
  // properties, and assigns it to the SQL property
  IBCStoredProc1->PrepareSQL();
  // searches a parameter by its name and assigns a new value
  IBCStoredProc1->ParamByName("sal")->AsFloat = 500.5;
  // executes the stored procedure
  IBCStoredProc1->Execute();
  // shows the resulting value
  ShowMessage(IBCStoredProc1->ParamByName("sal")->AsString);
}
__finally {
  IBCStoredProc1->Free();
}

Using Firebird Stored Functions

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

```
CREATE FUNCTION GiveBonusFunc(sal float)
RETURNS float
AS
```
This function can be executed as follows:

```sql
SELECT GiveBonusFunc(500.5);
```

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

**Delphi**

```delphi
var  IBCStoredProc1: TIBCStoredProc;
begin  IBCStoredProc1 := TIBCStoredProc.Create(nil);
  try    // IBCCConnection1 was set up earlier    IBCStoredProc1.Connection := IBCCConnection1;
    // indicates the name of the stored procedure to call    IBCStoredProc1.StoredProcName := 'GiveBonusFunc';
    // constructs a statement based on the Params and StoredProcName    // properties, and assigns it to the SQL property    IBCStoredProc1.PrepareSQL;
    // searches a parameter by its name and assigns a new value    IBCStoredProc1.ParamByName('sal').AsFloat := 500.5;
    // executes the stored procedure    IBCStoredProc1.Execute;
    // shows the resulting value    ShowMessage(IBCStoredProc1.ParamByName('result').AsString);
  finally    IBCStoredProc1.Free;
  end;
end;
```

**C++Builder**

```cpp
TIBCStoredProc* IBCStoredProc1 = new TIBCStoredProc(NULL);
try {    // IBCCConnection1 was set up earlier    IBCStoredProc1->Connection = IBCCConnection1;
    // indicates the name of the stored procedure to call    IBCStoredProc1->StoredProcName = "GiveBonusFunc";
    // constructs a statement based on the Params and StoredProcName    // properties, and assigns it to the SQL property    IBCStoredProc1->PrepareSQL();
    // searches a parameter by its name and assigns a new value    IBCStoredProc1->ParamByName("sal")->AsFloat = 500.5;
    // executes the stored procedure    IBCStoredProc1->Execute();
    // shows the resulting value    ShowMessage(IBCStoredProc1->ParamByName("result")->AsString);
} finally {    IBCStoredProc1->Free();
```
Note: To retrieve the result returned by the stored function using TIBCStoredProc, use the automatically created 'result' parameter.

Returning Result Sets

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

3.10 Using Stored Procedures with Result Sets

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

Requirements

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

General Information

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

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

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

Using the DataSet Functionality

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

**Delphi**

```delphi
var
  IBCStoredProc1: TIBCStoredProc;
begin
  IBCStoredProc := TIBCStoredProc.Create(nil);
  try
    // IBCConnection1 was set up earlier
    IBCStoredProc.Connection := IBCConnection1;
    // indicates the name of the stored procedure to call
    IBCStoredProc.StoredProcName := 'SelectDept';
    // constructs a statement based on the Params and StoredProcName
    // properties, and assigns it to the SQL property
    IBCStoredProc.PrepareSQL;
    // opens the dataset
    IBCStoredProc.Open;
    // adds a new empty record at the end of the dataset
    IBCStoredProc.Post;
    // searches fields by their names and assigns new values
    IBCStoredProc.FieldName('deptno').AsInteger := 50;
    IBCStoredProc.FieldName('dname').AsString := 'SALES';
    IBCStoredProc.FieldName('loc').AsString := 'NEW YORK';
    // writes the modified record
    IBCStoredProc.Post;
    // adds a new empty record in the current cursor position
    IBCStoredProc.Insert;
    IBCStoredProc.FieldName('deptno').AsInteger := 60;
    IBCStoredProc.FieldName('dname').AsString := 'ACCOUNTING';
    IBCStoredProc.FieldName('loc').AsString := 'LOS ANGELES';
    IBCStoredProc.Post;
    // positions the cursor on the deptno=10 record
    IBCStoredProc.FindKey([10]);
    // enables editing of data in the dataset
    IBCStoredProc.Edit;
    IBCStoredProc.FieldName('dname').AsString := 'RESEARCH';
    IBCStoredProc.FieldName('loc').AsString := 'LOS ANGELES';
    IBCStoredProc.Post;
  finally
    IBCStoredProc.Free;
  end;
end;
```

**C++Builder**

```cpp
TIBCS::TIBCS::TIBCSProc* TIBCS::TIBCSProc = new TIBCS::TIBCSProc(NULL);
try {
  // IBCConnection1 was set up earlier
  IBCConnection1->Connection = IBCConnection1;
  // indicates the name of the stored procedure to call
  IBCConnection1->StoredProcName = "SelectDept";
  // constructs a statement based on the Params and StoredProcName
```
3.11 Demo Projects

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

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

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

Where are the IBDAC demo projects located?

In most cases all the IBDAC demo projects are located in "%IbDac%\Demos\".
In Delphi 2007 under Windows Vista all the IBDAC demo projects are located in "My Documents\Devart\IbDac for Delphi 2007\Demos", for example "C:\Documents and Settings \All Users\Documents\Devart\IbDac for Delphi 2007\Demos\".

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

For most new IDEs with the structure will be as follows.

Demos

|--IbDacDemo [The main IBDAC demo project]
|--ThirdParty
  |-- [A collection of demo projects on integration with third-party components]
  |
|--Miscellaneous
  |-- [Some other demo projects on design technologies]

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

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

To explore an IBDAC demo project,

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

The executed version of the demo will contain a sample application written with IBDAC or a navigable list of samples and sample descriptions. To use each sample properly, you will need to connect to a working InterBase/Firebird server.
The included sample applications are fully functional. To use the demos, you have to first set up a connection to InterBase. You can do so by clicking on the "Connect" button.

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

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

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

**Demo project descriptions**

**IbDacDemo**

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

**Working with components**

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

**General demos**

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

**InterBase-specific demos**

A collection of samples that demonstrate how to incorporate InterBase/Firebird features in database applications.

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

**Working with Components**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alerter</td>
<td>Uses the <a href="#">TIBCAlerter</a> component to send messages between</td>
</tr>
</tbody>
</table>
connections using InterBase events.

ConnectDialog | Demonstrates how to customize the IBDAC connect dialog. Changes the standard IBDAC 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 IBDAC 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 TIBCQuery, which is one of the most useful IBDAC components. Includes many TIBCQuery usage scenarios. Demonstrates how to execute queries, how to edit data and export it to XML files.

Note: This is a very good introductory demo. We recommend starting here when first becoming familiar with IBDAC.

Sql | Uses TIBCSQL to execute SQL statements. Demonstrates how to prepare the statement and how to work with parameters in SQL.

StoredProc | Uses TIBCStoredProc to access an editable recordset retrieved by an InterBase stored procedure in the client application.

Table | Demonstrates how to use TIBCTable to work with data from a single table on the server without writing any SQL queries manually. Performs server-side data sorting and filtering and retrieves results for browsing and editing.

UpdateSQL | Demonstrates using the TIBCUpdateSQL component to customize update commands. Lets you optionally use TIBCSQL and TIBCQuery 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>Demonstrates how to perform the most important tasks of working with data in CachedUpdates mode, including highlighting uncommitted changes, managing transactions, and committing changes in a batch.</td>
</tr>
<tr>
<td>FilterAndIndex</td>
<td>Demonstrates IBDAC’s local storage functionality. This sample shows how to perform local filtering, sorting and locating by multiple fields, including by calculated and lookup fields.</td>
</tr>
<tr>
<td>MasterDetail</td>
<td>Uses IBDAC functionality to work with master/detail relationships.</td>
</tr>
</tbody>
</table>
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.

**Threads**

Demonstrates how IBDAC can be used in multithreaded applications. This sample allows you to set up several threads and test IBDAC's performance with multithreading.

### InterBase-specific Demos

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays</td>
<td>Demonstrates working with InterBase arrays. This sample lets you view and control how arrays are represented in dataset fields by the SparseArrays and ObjectView properties.</td>
</tr>
<tr>
<td>BlobPictures</td>
<td>Demonstrates working with InterBase BLOB data types. The sample shows how to get binary data from the table. Also it shows off some extended BLOB handling functionality like local caching control, deferred blob reading, getting blob subtype, and more.</td>
</tr>
<tr>
<td>DB.Key</td>
<td>Demonstrates using Firebird 2.0 RDB$DB.KEY field for building SQLInsert, SQLUpdate and SQLDelete properties.</td>
</tr>
<tr>
<td>LongStrings</td>
<td>Demonstrates IBDAC functionality for working with long string fields (fields that have more than 256 characters). Shows the different ways they can be displayed as memo fields and string fields.</td>
</tr>
<tr>
<td>TextBlobs</td>
<td>Demonstrates working with InterBase BLOB data types. The sample shows how to get text data from the table. Also it shows off some extended BLOB handling functionality like local caching control, deferred blob reading, blob compression, getting blob subtype, and more.</td>
</tr>
</tbody>
</table>

### Supplementary Demo Projects

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

<table>
<thead>
<tr>
<th>Location</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThirdParty</td>
<td>FastReport</td>
<td>Demonstrates how IBDAC can be used with FastReport components. This project consists of two parts. The first part is several packages that integrate IBDAC components into the...</td>
</tr>
<tr>
<td>Category</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>FastReport editor</td>
<td>The second part is a demo application that lets you design and preview reports with IBDAC technology in the FastReport editor.</td>
<td></td>
</tr>
<tr>
<td>InfoPower</td>
<td>Uses InfoPower components to display recordsets retrieved with IBDAC. This demo project displays an InfoPower grid component and fills it with the result of an IBDAC query. Shows how to link IBDAC data sources to InfoPower components.</td>
<td></td>
</tr>
<tr>
<td>IntraWeb</td>
<td>A collection of sample projects that show how to use IBDAC components as data sources for IntraWeb applications. Contains IntraWeb samples for setting up a connection, querying a database and modifying data, and working with CachedUpdates and MasterDetail relationships.</td>
<td></td>
</tr>
<tr>
<td>ReportBuilder</td>
<td>Uses IBDAC data sources to create a ReportBuilder report that takes data from InterBase database. Shows how to set up a ReportBuilder document in design-time and how to integrate IBDAC components into the Report Builder editor to perform document design in run-time.</td>
<td></td>
</tr>
<tr>
<td>CBuilder</td>
<td>A general demo project about creating IBDAC-based applications with C++Builder. Lets you execute SQL scripts and work with result sets in a grid. This is one of the two IBDAC demos for C++Builder.</td>
<td></td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FailOver</td>
<td>Demonstrates the recommended approach to working with unstable networks. This sample lets you perform transactions and updates in several different modes, simulate a sudden session termination, and view what</td>
<td></td>
</tr>
</tbody>
</table>
Getting Started

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 IBDAC. 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.

VirtualTableCB

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.

IbDacDemo

IbDacDemo

[Win32 version of the main IBDAC demo project - see above]

© 1997-2021 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback

3.12 Deployment

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

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

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

**Trial Limitation Warning**

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

<table>
<thead>
<tr>
<th>BPL File</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>dacXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>ibdacXX.bpl</td>
<td>always</td>
</tr>
</tbody>
</table>

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

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

**Deploying Windows applications built with run-time packages**

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

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

<table>
<thead>
<tr>
<th>BPL File</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>dacXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>ibdacXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>dacvclXX.bpl</td>
<td>if your application uses the IbDacVcl unit</td>
</tr>
<tr>
<td>ibdacvclXX.bpl</td>
<td>if your application uses the IbDacVcl unit</td>
</tr>
<tr>
<td>crcontrolsXX.bpl</td>
<td>if your application uses the CRDBGrid component</td>
</tr>
</tbody>
</table>

If you remove the names of these assemblies from the References list of your project, these files will not be required on the target computer.
4 Using IBDAC

This section describes basics of using InterBase Data Access Components

- Updating Data with IBDAC Dataset Components
- Master/Detail Relationships
- Automatic Key Field Value Generation
- Data Type Mapping
- Data Encryption
- Working in an Unstable Network
- Disconnected Mode
- Increasing Performance
- Macros
- DataSet Manager
- Connection Pooling
- BLOB Data Types
- Unicode Character Data
- ARRAY Data Type
- TIBCQuery Component
- DBMonitor
- Writing GUI Applications with IBDAC
- Compatibility with Previous Versions
- 64-bit Development with Embarcadero RAD Studio XE2
- Database Specific Aspects of 64-bit Development
- Demo Projects
- Deployment

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

4.1 Updating Data with IBDAC Dataset Components

Queries are often complex so posting result set modifications to the database becomes not a trivial task. IBDAC dataset components which descend from TCustomIBCQuery provide different means for reflecting local changes on the server.

The following components are used to execute SQL statements: TIBCQuery, TIBCStoredProc, TIBCTable.
If application requires result set from a single database table then use TIBCTable to query data. Setting only TableName property you may obtain data, modify it and then post changes back to the database.

TIBCQuery component may return recordsets from different tables and views all in a single query. There is often no reliable way to make automated update of the database having only original SQL statement or particularly only a name of the stored procedure. To solve this problem additional properties are provided: SQLInsert, SQLUpdate and SQLDelete. Set them with SQL statements which will perform corresponding data modifications on behalf of the original statement whenever insert, update or delete operation is called. You may also assign UpdateObject property with the TIBCUpdateSQL class instance which holds all updating SQL statements in one place.

TIBCQuery can generate SQL statements for the SQLInsert, SQLUpdate and SQLDelete properties based on the original SQL statement. To identify rows which have to be processed when modified data is applied to the database KeyFields property must be assigned with the names of key fields so that the records are uniquely identified.

For the more careful customization of data update operations you can use InsertObject, ModifyObject and DeleteObject properties of TIBCUpdateSQL component.

Set the Transaction property of your DataSet component to the transaction component with ReadCommitted/ReadOnly IsolationLevel property, and UpdateTransaction property to the transaction component with ReadCommitted IsolationLevel property for the optimal transaction using performance. Borland recommends to start the read-only transaction and commit it with CommitRetaining on InterBase 7.1. Using transactions in such a way minimizes server load.

In Firebird 2.0 and higher you can use RETURNING clause of INSERT statement to get the inserted values. It can be useful for getting back inserted values if they are changed by BEFORE INSERT trigger. To add returning clause to SQLInsert automatically set DMLRefresh property to True.

When you use returning clause of statement in the IBCSQL or IBCQuery component, additional out parameters appear after preparing or executing the statement. They contain returned values and have names like "RET_" + column_name. Column name is a name of the column of returned value.

Example:

```
INSERT INTO T1 (F1, F2)
VALUES (:F1, :F2)
RETURNING F1, F2
```
After executing or preparing such statement the following out parameters appear: RET_F1 and RET_F2. They will contain values of F1 and F2 fields.

IBDAC supports working with RDB$DB_KEY field in Firebird 2.0. RDB$DB_KEY is raw record position in database. DB_KEY provides DB_KEY field that is used when it is incuded in the SQL explicitly and KeyFields property is not set. This field is represented with TIBCDbKeyField class. It will be used for building SQLInsert, SQLUpdate and SQLDelete properties. It can speed up your work because DB_KEY is even faster than PK.

**Note:** By default Db_Key field initialized with Visible = False. You should explicitly create Db_Key field to display it.

### See Also
- TIBCQuery
- TIBCStoredProc
- TIBCTable
- TIBCDbKeyField
- TCustomIBCDataSet.DMLRefresh
- TCustomIBCDataSet.UpdateTransaction

© 1997-2021 Devart. All Rights Reserved.
procedure TForm1.Form1Create(Sender: TObject);
var
  Master, Detail: TIBCQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TIBCQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  // create detail dataset
  Detail := TIBCQuery.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.

**Note:** To make this example first you should place TIBCConnection and TIBCTransaction components on the form, assign them to each other by setting TIBCConnection.Transaction and TIBCTransaction.Connection properties and provide connection parameters for connection to the sample database 'Employee'.

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: TIBCQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TIBCQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  // create detail dataset
  Detail := TIBCQuery.Create(Self);
  Detail.SQL.Text := 'SELECT * FROM Employee';
In this code example MD relationship is set up using `MasterFields` and `DetailFields` properties. Also note that there are no WHERE clause in detail dataset SQL.

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

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

See Also

- `TCustomDADataSet.Options`
- `TMemDataSet.CachedUpdates`

4.3 Automatic Key Field Value Generation

When editing dataset it is often convenient not to fill key field(s) values manually but generate them automatically. In the most common way an application developer generates a primary key value basing on a previously created generator. There are three ways to do it.

First, application independent way - developer creates an AFTER INSERT trigger that fills the field value. But there he faces the problem with getting a value inserted by the trigger back to dataset.
Second way is custom key field value generation. A developer can fill a key field value in TCustomIBCQuery.BeforePost event handler. But in that case he should manually execute a query and retrieve the generator value. So this way may be useful only if some special value processing is needed.

Third way, using Generator is the most simple one. A developer only needs to specify two properties - and key field values are generated automatically. There is no need to create a trigger or perform custom BeforePost processing.

Example:

```pascal
...IBCQuery.SQL.Text := 'SELECT DepNo, DepName, Location FROM Department';
IBCQuery.KeyFields := 'DEPT_NO';  // key field
IBCQuery.Generator := 'DeptGenerator'; // generator that will generate values...
```

See also

- [KeyGenerator](#)

© 1997-2021 Devart. All Rights Reserved.

### 4.4 Data Type Mapping

**Overview**

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

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

**Data Type Mapping Rules**

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

Here is the example with the numeric type in the following table of an InterBase or Firebird
database:

```sql
CREATE TABLE NUMERIC_TYPES(
    ID INTEGER NOT NULL PRIMARY KEY,
    VALUE4 NUMERIC(5, 2),
    VALUE5 NUMERIC(10, 4),
    VALUE6 NUMERIC(15, 6)
)
```

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

The above in the form of a table:

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

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

```pascal
var
    DBType: Word;
    MinPrecision: Integer;
    MaxPrecision: Integer;
    MinScale: Integer;
    MaxScale: Integer;
    FieldType: TFieldType;
begin
    DBType := ibcNumeric;
    MinPrecision := 10;
    MaxPrecision := r1Any;
    MinScale := 1;
    MaxScale := 4;
    FieldType := ftBCD;
    IBCConnection.DataTypeMap.AddDBTypeRule(DBType, MinPrecision, MaxPrecision, MinScale, MaxScale, FieldType);
end;
```

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

```pascal
// rule for numeric(5,2)
```
IBCConnection.DataTypeMap.AddDBTypeRule(ibcNumeric, 0, 9, rlAny, ftFloat, 4, ftBCD);// rule for numeric(10,4)
IBCConnection.DataTypeMap.AddDBTypeRule(ibcNumeric, 10, rlAny, 4, rlAny, ftFMTBCD);

Rules order

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

For example, there is a table in an InterBase or Firebird database:

```sql
CREATE TABLE NUMERIC_TYPES
(
  ID INTEGER NOT NULL PRIMARY KEY,
  VALUE4 NUMERIC(5, 2),
  VALUE5 NUMERIC(10, 4),
  VALUE6 NUMERIC(15, 6)
)
```

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

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

If rules are set in the following way:

```delphi
IBCConnection.DataTypeMap.Clear;
IBCConnection.DataTypeMap.AddDBTypeRule(ibcNumeric, 0, 9, rlAny, 4, ftFloat);  // rule for numeric(10,4)
IBCConnection.DataTypeMap.AddDBTypeRule(ibcNumeric, 10, rlAny, 4, rlAny, ftBCD);  // rule for numeric(15,6)
```

it will lead to the following result:

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

But if rules are set in the following way:
IBConnection.DataTypeMap.Clear;
IBConnection.DataTypeMap.AddDBTypeRule(ibcNumeric, 0, rlAny, 0, rlAny, 0, rlAny, 0, rlAny, 0, rlAny, 4, ftFMTBCD);
IBConnection.DataTypeMap.AddDBTypeRule(ibcNumeric, 0, rlAny, 9, rlAny, ftFloat);

it will lead to the following result:

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

This happens because the rule

IBConnection.DataTypeMap.AddDBTypeRule(ibcNumeric, 0, rlAny, 0, rlAny, ftFMTBCD);

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

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

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

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

**Defining rules for Connection and Dataset**

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

For example, such table is created in SQL Server:

```sql
CREATE TABLE PERSON
(
    ID INTEGER NOT NULL PRIMARY KEY,
```
FIRSTNAME VARCHAR(20),
LASTNAME VARCHAR(30),
GENDER_CODE VARCHAR(1),
BIRTH_DTTM TIMESTAMP
)

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:

IBCConnection.DataTypeMap.Clear;
IBCConnection.DataTypeMap.AddDBTypeRule(ibcTimestamp, 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:

IBCQuery.DataTypeMap.Clear;
IBCQuery.DataTypeMap.AddDBTypeRule(ibcTimestamp, 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:

IBCConnection.DataTypeMap.Clear;
IBCConnection.DataTypeMap.AddDBTypeRule(ibcTimestamp, ftDate);
IBCQuery.DataTypeMap.Clear;
IBCQuery.DataTypeMap.AddDBTypeRule(ibcTimestamp, ftDateTime);

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

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

Rules for a particular field

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

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

CREATE TABLE ITEM
(
ID INTEGER NOT NULL PRIMARY KEY,
NAME CHAR(50),
GUID CHAR(38)
)

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

IBCQuery.DataTypeMap.Clear;
IBCQuery.DataTypeMap.AddDBTypeRule(ibcChar, 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:

IBCQuery.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.

Rules for a particular character set

In InterBase/Firebird, character fields (CHAR, VARCHAR), as well as BLOB fields, may have a charset different from the one of the database. For example, a table may be created in the following way:

CREATE TABLE TEST
(
   ID INTEGER NOT NULL PRIMARY KEY,
   FIELD1 VARCHAR(10) CHARACTER SET UTF8,
   FIELD2 VARCHAR(10) CHARACTER SET WIN1251
)

To set different mapping-by-charset rules for fields with the same type (e.g., map UTF-8 VARCHAR fields as ftString, and fields encoded in WIN1251 as ftWideString), you can use the following IBDAC methods:

IBCConnection.DataTypeMap.Clear;
IBCConnection.DataTypeMap.AddCharsetRule(ibcVarchar, 'UTF8', ftString);
IBCConnection.DataTypeMap.AddCharsetRule(ibcVarchar, 'WIN1251', ftWideString);

Ignoring conversion errors

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

For example:

<table>
<thead>
<tr>
<th>Database value</th>
<th>Destination field type</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>'text value'</td>
<td>ftInteger</td>
<td>String cannot be converted to Integer</td>
</tr>
<tr>
<td>1000000</td>
<td>ftSmallint</td>
<td>Value is out of range</td>
</tr>
<tr>
<td>15,1</td>
<td>ftInteger</td>
<td>Cannot convert float to integer</td>
</tr>
</tbody>
</table>

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

`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.

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

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

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

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

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

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

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

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

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

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

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

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

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

**Example.**

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

```
CREATE TABLE EMP (  EMPNO INTEGER NOT NULL PRIMARY KEY,  ENAME VARCHAR(2000) CHARACTER SET OCTETS COLLATE OCTETS,  HIREDATE VARCHAR(200) CHARACTER SET OCTETS COLLATE OCTETS,  SAL VARCHAR(200) CHARACTER SET OCTETS COLLATE OCTETS,  FOTO BLOB SUB_TYPE 0)
```

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.

```
IBCQuery.SQL.Text := 'SELECT * FROM EMP';
IBCQuery.Encryption.Encryptor := IBCEncryptor;
IBCQuery.Encryption.Fields := ['ENAME', 'HIREDATE', 'SAL', 'FOTO'];
IBCQuery.Open;
```

4.6 Working in an Unstable Network

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

- `TCustomDAConnection.Options.LocalFailover = True`
- `TCustomDAConnection.Options.DisconnectedMode = True`
- `TDataset.CachedUpdates = True`
- `TCustomDADataset.FetchAll = True`
- `TCustomDADataset.Options.LocalMasterDetail = True`
- `AutoCommit = True`

It is recommended to use ReadCommitted or ReadOnly `IsolationLevel` of `TCustomIBCDataSet.Transaction`. Use `TCustomIBCDataSet.UpdateTransaction` for update operations. If connection has at least one opened transaction, which is not `ReadCommittedReadOnly`, FailOver does not execute. All `ReadCommittedReadOnly` transaction are restored with FailOver operation. In Disconnected mode you can work with one `ReadWrite` transaction, but it is not recommended.

The following settings are recommended for working with BLOB and array fields in an unstable network.

- `TCustomIBCDataSet.Options.DeferredBlobRead = False`
- `TCustomIBCDataSet.Options.DeferredArrayRead = False`
- `TCustomIBCDataSet.Options.CacheArrays = True`
- `TCustomIBCDataSet.Options.CacheBlobs = True`
- `TCustomIBCDataSet.Options.StreamedBlob = False`

These settings allow to work with Blobs and Arrays without an active connection.

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 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), IBDAC can implicitly perform the following operations:

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

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

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

**See Also**
- FailOver demo
- Working with Disconnected Mode
- TCustomDAConnection.Options
- TCustomDAConnection.Pooling

4.7 **Disconnected Mode**

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

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

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

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

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

It is recommended to use ReadCommited or ReadOnly isolationLevel of **IsolationLevel** of **TCustomIBCDataSet.Transaction**. Use **TCustomIBCDataSet.UpdateTransaction** for update operations. If connection has at least one opened transaction, which is not ReadCommitedReadOnly, FailOver does not execute. All ReadCommitedReadOnly transaction are restored with FailOver operation. In Disconnected mode you can work with one ReadWrite transaction, but it is not recommended.

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.IbDac.TIBCQuery.LockMode**
- **TCustomDACConnection.Pooling**
- **TCustomDACConnection.Connect**
- **TCustomDACConnection.Disconnt**
- **Working in unstable network**
4.8 Batch Operations

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

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

**Batch operations execution**

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

```sql
```

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

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

After the query execution, one record will be inserted into the BATCH_TEST table.
When using Batch operations, the query and its parameters remain unchanged. However, parameter values will be enclosed in an array:

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

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

How to implement a Batch operation in the code?

**Batch INSERT operation sample**

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

```pascal
var
  i: Integer;
begin
  // describe the SQL query
  IBCQuery1.SQL.Text := 'INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, ...' + Now;
  // define the parameter types passed to the query:
  IBCQuery1.Params[0].DataType := ftInteger;
  IBCQuery1.Params[1].DataType := ftInteger;
  IBCQuery1.Params[2].DataType := ftFloat;
  IBCQuery1.Params[3].DataType := ftString;
  IBCQuery1.Params[4].DataType := ftDateTime;
  // specify the array dimension:
  IBCQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to IBCQuery1.Params.ValueCount - 1 do begin
    IBCQuery1.Params[0][i].AsInteger := i + 1;
    IBCQuery1.Params[1][i].AsInteger := i + 2000 + 1;
    IBCQuery1.Params[2][i].AsFloat := (i + 1) / 12;
    IBCQuery1.Params[3][i].AsString := 'Values ' + IntToStr(i + 1);
    IBCQuery1.Params[4][i].AsDateTime := Now;
  end;
  // insert 1000 rows into the BATCH_TEST table
  IBCQuery1.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:

Query1.Execute(1000);

2×500 rows:

// insert first 500 rows
IBCQuery1.Execute(500, 0);
// insert next 500 rows
IBCQuery1.Execute(500, 500);

500 rows, then 300, and finally 200:

// insert 500 rows
Query1.Execute(500, 0);
// insert next 300 rows starting from 500
IBCQuery1.Execute(300, 500);
// insert next 200 rows starting from 800
Query1.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
  IBCQuery1.SQL.Text := 'UPDATE BATCH_TEST SET F_INTEGER=:F_INTEGER, F_FLOAT=:F_FLOAT,
  F_DATETIME=:F_DATE, F_STRING=:F_STRING, F_BOOLEAN=:F_BOOLEAN;
  // define parameter types passed to the query:
  IBCQuery1.Params[0].DataType := ftInteger;
  IBCQuery1.Params[1].DataType := ftFloat;
  IBCQuery1.Params[2].DataType := ftString;
  IBCQuery1.Params[3].DataType := ftDateTime;
  IBCQuery1.Params[4].DataType := ftInteger;
  // specify the array dimension:
  IBCQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to 1000 - 1 do begin
    IBCQuery1.Params[0][i].AsInteger := i - 2000 + 1;
    IBCQuery1.Params[1][i].AsFloat := (i + 1) / 100;
    IBCQuery1.Params[2][i].AsString := 'New Values ' + IntToStr(i + 1);
    IBCQuery1.Params[3][i].AsDateTime := Now;
    IBCQuery1.Params[4][i].AsInteger := i + 1;
  end;
  // update 1000 rows in the BATCH_TEST table
  IBCQuery1.Execute(1000);
end;
Batch DELETE operation sample

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

```pascal
var
  i: Integer;
begin
  // describe the SQL query
  IBCQuery1.SQL.Text := 'DELETE FROM BATCH_TEST WHERE ID=:ID';
  // define parameter types passed to the query:
  IBCQuery1.Params[0].DataType := ftInteger;
  // specify the array dimension
  IBCQuery1.Params.ValueCount := 1000;
  // populate the arrays with parameter values
  for i := 0 to 999 - 1 do
    IBCQuery1.Params[0][i].AsInteger := i + 1;
  // delete 1000 rows from the BATCH_TEST table
  IBCQuery1.Execute(1000);
end;
```

Performance comparison

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

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>25 000 records</th>
<th>Standard Operation (sec.)</th>
<th>Batch Operation (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>25 000 records</td>
<td>55.4</td>
<td>3.03</td>
</tr>
<tr>
<td>Update</td>
<td>25 000 records</td>
<td>81.9</td>
<td>3.58</td>
</tr>
<tr>
<td>Delete</td>
<td>25 000 records</td>
<td>61.3</td>
<td>0.91</td>
</tr>
</tbody>
</table>

The less, the better.

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

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

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

```pascal
for i := 0 to 9999 do begin
  IBCQuery1.Params.ParamByName('ID')[i].AsInteger := i + 1;
  IBCQuery1.Params.ParamByName('F_INTEGER')[i].AsInteger := i + 2000 + 1;
end;
```
IBCQuery1.Params.ParamByName('F_FLOAT')[i].AsFloat := (i + 1) / 12;
IBCQuery1.Params.ParamByName('F_STRING')[i].AsString := 'Values ' + IntToStr(i + 1);
IBCQuery1.Params.ParamByName('F_DATE')[i].AsDateTime := Now;
end;

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

4.9 Increasing Performance

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

Connect

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

Execute

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

To execute SQL statements a TIBCSQL component is more preferable than TIBCQuery. 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. Sometimes using
**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.

Specifying update SQL statements in a dataset improves performance because of omitting SQL statements generation and automatic preparation of internal updating datasets that are created for every kind of update SQL statements.
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.

4.10 Macros

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

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

Then modify SQL statement to include macro names into desired insertion points. Prefix each name with & ("at") sign to let IBDAC 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.

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

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

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

```plaintext
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.
4.11 DataSet Manager

DataSet Manager window

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

Using the DataSet Manager window, you can:

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

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

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

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

Opening the DataSet Manager window

You can display the DataSet Manager window by clicking DataSet Manager on the Tools menu. 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.
4.12 Connection Pooling

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

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

To use connection pooling set the Pooling property of the TCustomDAConnection component to True. Also you should set the PoolingOptions of the TCustomDAConnection. These options include MinPoolSize, MaxPoolSize, Validate, ConnectionLifeTime. Connections belong to the same pool if they have identical values for the following parameters: MinPoolSize, MaxPoolSize, Validate, ConnectionLifeTime, Server, Username, Password, Database, TIBConnectionOptions_CHARSET, TIBConnectionOptions.UseUnicode, TIBConnectionOptions_ROLE, SQLDialect, Params. When a connection component disconnects from the database the connection actually remains active and is placed into the pool. When this or another connection component connects to the database it takes a connection from the pool. Only when there are no connections in the pool, new connection is established.

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

The pool frees connections that are held in the pool during a long time. If no new connections are placed to the pool it becomes empty after approximately 4 minutes. This pool behaviour is intended to save resources when the count of connections in the pool exceeds the count that
is needed by application. If you set the `PoolingOptions.MinPoolSize` property to a non-zero value, this prevents the pool from freeing all pooled connections. When connection count in the pool decreases to `MinPoolSize` value, remaining connection will not be freed except if they are broken.

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

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

To force freeing of a connection when the `TCustomDAConnection` component disconnects, the `RemoveFromPool` method of `TCustomDAConnection` can be used. You can also free all connection in the pool by using the class procedures `Clear` or `AsyncClear` of TIBCCConnectionPoolManager. 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`

© 1997-2021 Devart. All Rights Reserved. Request Support  DAC Forum  Provide Feedback
4.13 BLOB Data Types

BLOB field can be used to store large amounts of data of different types. For BLOB data type only BLOB IDs (pointers to data) are stored in table columns; actual BLOB data is stored separately. When accessing a BLOB column, it is the ID which is returned, not the value itself. InterBase supports two types of blobs, stream and segmented. Segmented BLOBs are usual InterBase BLOBs and are stored in chunks. Stream BLOBs are stored as a continuous array of data bytes.

IBDAC components provide following features for working with BLOBs:

- Working with BLOBs the same way like with another fields.
- Fetching BLOB fields on demand.
- Streamed BLOBs support.
- Server side access to BLOB fields.
- Setting BLOB Parameter Buffer (BPB) for using BLOB subtype conversion.
- Compressing BLOB data.

IBDAC components support InterBase BLOB datatype. You can retrieve values of BLOB fields using TIBCQuery component the same way as you do for another fields.

It is possible to control the way BLOB objects are handled while the application fetches records from the database. BLOBs can be fetched either with other fields to the application or on demand. This is determined by DeferredBlobRead option in TCustomIBCDataSet component. Setting TCustomIBCDataSet.Options.DeferredBlobRead to false allows to reduce traffic over the network since BLOBs are only transferred on demand and to use less memory on the client side because returned record sets do not hold contents of BLOB fields.

IBDAC components support InterBase streamed BLOBs. To enable streamed BLOBs handling set TCustomIBCDataSet.Options.StreamedBlobs to True. Setting this option to True makes all edited BLOBs to be saved as streamed BLOBs and all streamed BLOBs to be handled as streamed. Otherwise streamed BLOBs are handled as usual segmented BLOBs and all edited BLOBs are saved as segmented BLOBs. Setting this option to True also allows to use benefits of server side access to BLOB fields.

Set TCustomIBCDataSet.Options.CacheBlobs to False to access streamed BLOB values on server side without caching BLOBs on the client side. Only requested portions of data are fetched in that case. Setting CacheBlobs to False may bring up the following benefits for time-critical applications: reduced traffic over the network since only required data are fetched, less memory is needed on the client side because BLOB data are not cached on the client side. This feature is available only for streamed BLOBs and only if StreamedBlobs option is set to
True. This option doesn't make sense if DefferedBlobRead is set to False because all BLOB values are fetched to the dataset in that case.

IBDAC components provides easy usage of InterBase BLOB subtype conversion, using BLOB filters. It allows to set BLOB parameter buffer (BPB), that is needed whenever a filter will be used when writing to or reading from a BLOB. BPB contains source and target subtype and charset (for text BLOBs). These parameters are set in TIBCBlob component by setting TIBCBlob.CharsetID, TIBCBlob.ConversionCharsetID, TIBCBlob.ConversionSubType, TIBCBlob.SubType properties. In retrieval operations, when you set them before reading BLOB, SubType and Charset properties are considered actual subtype and charset of database BLOB data. Application will get data converted to ConversionSubType subtype and ConversionCharset charset.

In upload operations, SubType and Charset properties mean actual subtype and charset of BLOB data in the application. ConversionSubType and ConversionCharset properties must contain desired subtype and charset to save BLOB to database with.

To avoid unwanted conversions do not set these properties, or make sure that Charset equals to ConversionCharset and SubType equals to ConversionSubType.

Note that if there is no filter for pair of source and target subtype, no conversion is provided and BLOB data remains unconverted.

Executing TIBCQuery or TIBCSQL with BLOB parameter requires live transaction after execute. To execute such statement you should explicitly start the transaction or set AutoCommit property to False.

Use P:Devart.Dac.TDADatasetOptions.CompressBlobMode for managing BLOB compression. BLOBs can be stored compressed on the client side, on the server side (in database) or on the both sides. By default it has value cbNone, that means no compression is provided. Use cbClient value to store compressed blobs on client side. This saves client memory. BLOB data is stored unchanged in database, other application can read these BLOBs as usual. If cbServer value is used, BLOB data is stored compressed in database. It's decompressed on the client side. This saves server disk space and network traffic. Other application can't process compressed BLOB data as usual. To use compressed BLOB data both on the client and server side use cbClientServer value. To use cbClient, cbServer, cbClientServer and cbNone constants you should add MemData unit to uses clause.

Note: Internal compression functions are available in Delphi 2007, Borland Developer Studio 2006, Delphi 2005, and Delphi 7. To use BLOB compression under Delphi 6 and C++Builder you should use your own compression functions. To use them set CompressProc and
UncompressProc variables declared in MemUtils unit.

- Pascal

```pascal
type
TCompressProc = function(dest: IntPtr; destLen: IntPtr; const source: IntPtr; sourceLen: longint): longint;
TUncompressProc = function(dest: IntPtr; destlen: IntPtr; source: IntPtr; sourceLne: longint): longint;

var
CompressProc: TCompressProc;
UncompressProc: TUncompressProc;
```

You can compress and decompress a single BLOB. To do it set
P:Devart.Dac.TCompressedBlob.Compressed property. Set it to True to compress BLOB data and to False to decompress BLOB data.

Note that using compression and decompression operations will raise CPU usage and can reduce application performance.

See Also
- TIBCBlob
- TCompressedBlob
- TCustomDADataset.Options
- TCustomIBCDataSet.Options
- TDAParam.ParamType

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

4.14 Unicode Character Data

Symbolic information in InterBase can be retrieved for the user as a different character encoding according to the query. InterBase supports number of encoding formats including Unicode. IBDAC components support UTF-8 Unicode (Unicode_FSS) encoding formats for data fields.

IBDAC allows to represent string data using string and WideString types. You can use TIBConnection.Options.UseUnicode property to enable this behaviour. This property value affects fields of queries and stored procedures. TIBConnection.Options.UseUnicode property has no influence to the parameters types of which were set manually.

Suppose that SIMPLE_TYPES table is created as:
CREATE TABLE SIMPLE_TYPES (  ID INTEGER NOT NULL,  F_CHAR CHAR(250),  F_VARCHAR VARCHAR2(300),)

Suppose we open the following SELECT statement in a dataset  
SELECT a.* FROM SIMPLE_TYPES a

If `TIBConnection.Options.UseUnicode` is set to False you get the next fields list after dataset is opened:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>TIntegerField</td>
</tr>
<tr>
<td>F_CHAR</td>
<td>TStringField</td>
</tr>
<tr>
<td>F_VARCHAR</td>
<td>TStringField</td>
</tr>
</tbody>
</table>

When you set `TIBConnection.Options.UseUnicode` to True the string fields type changes:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>TIntegerField</td>
</tr>
<tr>
<td>F_CHAR</td>
<td>TWideStringField</td>
</tr>
<tr>
<td>F_VARCHAR</td>
<td>TWideStringField</td>
</tr>
</tbody>
</table>

Fields of TWideStringField type hold rows in UTF-16 Unicode format. To get a value of the fields you can use TWideStringField.Value property. You can use FlatBuffers, LongString, FieldsAsString, TrimFixedChar options of `TCustomIBCDataSet` which are compatible with `TIBConnection.Options.UseUnicode`.

To use Unicode values as parameters, previously you need to set a value of data type field to ftWideString or ftFixedWideChar for the fields of VARCHAR or CHAR types accordingly. Otherwise after the execution of AsWideString or AsString operation data type field will be ftString by default.

```pascal
var  WS: WideString;
begin...
  with IBCQuery1 do begin
    Close;
    SQL.Text:=      'SELECT * from SIMPLE_TYPES '+      'WHERE '+      '  F_CHAR = :F_CHAR';    Params[0].DataType := ftFixedWideChar;    Params[0].AsWideString := WS;    Open;
  ...
```
Note that if table field has charset NONE or OCTETS, no charset conversion is provided and such fields are always fetched and stored as usual TStringField. You can set charset when creating or modifying table.

If parameter has Unicode data type value, assigning value by using AsString property converts String to WideString. And vice versa, if parameter doesn't have Unicode data type value, assigning value by AsWideString property converts WideString into String.

BLOB data type supports string data in UTF-8 Unicode encoding. You can set \texttt{TIBConnection.Options.UseUnicode} property to True and get TMemoField of ftBlob blob type. BLOB must have subtype 1 (text) and Unicode_FSS charset to use this feature.

\textbf{See Also}

- \texttt{TIBConnection.Options}
- \texttt{TCustomIBCDataSet.Options}

\section*{4.15 \textbf{ARRAY Data Type}}

Some problems appear when you need to use large arrays in dataset. As IBDAC creates one field for each array item great number of TField objects is created. As a result of it the performance decreases. So IBDAC has the limitation and creates fields for first 1000 items. However, you can access all items with TIBCArray object.

If such table is created

\begin{verbatim}
CREATE TABLE IBDAC_ARRAYS (    ID             INTEGER NOT NULL,    CHAR_ARRAY     CHAR(10) [1:5],    INTEGER_ARRAY  INTEGER [1:2,1:5],    FLOAT_ARRAY    FLOAT [0:8,0:2];
)
\end{verbatim}

If ComplexArrayFields is False you can access array item using TIBCArrayField

\begin{verbatim}
Value := TIBCArrayField(IBCQuery1.FieldByName('CHAR_ARRAY')).AsArray.GetItemAsString([5]);
\end{verbatim}

If ComplexArrayFields is True, to access array items you can call FieldByName method. For example

\begin{verbatim}
Value := Query.FieldByName('CHAR_ARRAY[1]').AsString;
\end{verbatim}
If ObjectField property is True this code is right

```pascal
Value := TADTField(Query.FieldByName('INTEGER_ARRAY[1]')).Fields[0].AsInteger;
```

Using TIBCDataset.GetArray you can access to array items through TIBCArray object

```pascal
Value := Query.GetArray('FLOAT_ARRAY').GetItemAsFloat([5, 2]);
```

It is possible to control the way Array objects are handled while the application fetches records from the database. Arrays can be fetched either with other fields to the application or on demand. This is determined by `DeferredArrayRead` option in TCustomIBCDataSet component. Setting `DeferredArrayRead` to False allows to reduce traffic over the network since arrays are only transferred on demand and to use less memory on the client side because returned record sets do not hold contents of the array fields.

Set `TCustomIBCDataSet.Options.CacheArrays` to False to access array values on server side without caching arrays on the client side. Only requested portions of data are fetched in that case. Setting `CacheArrays` to False may bring up the following benefits for time-critical applications: reduced traffic over the network since only required data are fetched, less memory is needed on the client side because array data are not cached on client side.

You can use array type for parameters of SQL statements. You need to assign `dTIBCArray` to `TIBCParam.DataType` and use `TIBCParam.AsArray` property to access array items.

For example:

```pascal
var
  IBCSQL: TIBCSQL;
  ...
  IBCSQL.SQL.Text := 'insert into IBDAC ARRAYS (ID, CHAR_ARRAY) VALUES(:ID, :CHAR_ARRAY)';
  with IBCSQL.ParamByName('ID').AsInteger := 50;
  IBCSQL.ParamByName('CHAR_ARRAY').AsArray do begin
    TableName := 'IBDAC ARRAYS';
    ColumnName := 'CHAR_ARRAY';
    DbHandle := IBCSQL.Connection.Handle;
    TrHandle := IBCSQL.Transaction.Handle;
    GetArrayInfo;
    SetItemAsString([1], 'AA');
    SetItemAsString([2], 'BB');
    SetItemAsString([3], 'CC');
  end;
  IBCSQL.Execute;
```

See Also
- `TIBCArray`
4.16 **TIBCQuery Component**

Important feature of TIBCQuery is that TIBCQuery is able to generate DML statements for updating data on the server itself. Remember that TIBCQuery is able to generate statements for updating only one table. By default TIBCQuery uses the first table from FROM clause. You should assign table name to UpdatingTable property if another table is used for updating. If you need more complex SQL statements than generated by TIBCQuery, use SQLInsert, SQLUpdate, SQLDelete properties to assign any SQL statements. It is not obligatory to assign all the properties, not assigned are still generated automatically.

**See Also**

- **TIBCQuery**

4.17 **DBMonitor**

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

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 **TIBCSQLMonitor** component onto the form;
- turn **moDBMonitor** option on;
- set to True the Debug property for components you want to trace;
4.18 Writing GUI Applications with IBDAC

IBDAC 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 IbDacVcl unit in your application. This feature is needed for writing console applications.

_Delphi and C++Builder_

By default IBDAC does not require Forms, Controls and other GUI related units. Only TIBConnectDialog and TIBErrorHandler require the Forms unit.

4.19 Compatibility with Previous Versions

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

If the manually assigned `RefreshSQL` property contains only "WHERE" clause, IBDAC 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 IBDAC 2.00.0.4. To restore the old behavior, set the `BaseSQLOldBehavior` variable to True.

**MemDS.SendDataSetChangeEventAfterOpen:**

Starting with IBDAC 2.20.0.11, 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 IBDAC 2.20.0.12, if the OnUpdateRecord event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged.

To restore the old behaviour, set DoNotRaiseExcetionOnUaFail to True.

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

**Extended type**

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

<table>
<thead>
<tr>
<th>TYPE</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended</td>
<td>10 bytes</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

The following two additional types are introduced to ensure compatibility in the process of developing 32- and 64-bit applications:

Extended80 – whose size in 32-bit application is 10 bytes; however, this type provides the same precision as its 8-byte equivalent in 64-bit applications.

Extended80Rec – can be used to perform low-level operations on an extended precision floating-point value. For example, the sign, the exponent, and the mantissa can be changed separately. It enables you to perform memory-related operations with 10-bit floating-point variables, but not extended-precision arithmetic operations.

**Pointer and Integers**

The major difference between 32- and 64-bit platforms is the volume of the used memory and, correspondingly, the size of the pointer that is used to address large memory volumes.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>4 bytes</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>
At the same time, the size of the Integer type remains the same for both platforms:

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

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

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

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>TYPE</th>
<th>Delphi Version</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>NativeInt</td>
<td>D5</td>
<td>N/A</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D6</td>
<td>N/A</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D7</td>
<td>8 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2005</td>
<td>8 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2006</td>
<td>8 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2007</td>
<td>8 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2009</td>
<td>4 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2010</td>
<td>4 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>Delphi XE</td>
<td>4 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>Delphi XE2</td>
<td>4 or 8 bytes</td>
</tr>
</tbody>
</table>

Out parameters

Some WinAPIs have OUT parameters of the SIZE_T type, which is equivalent to NativeInt in
Delphi XE2. The problem is that if you are developing only a 32-bit application, you won't be able to pass Integer to OUT, while in a 64-bit application, you will not be able to pass Int64; in both cases you will have to pass NativeInt.

For example:

```delphi
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:**

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

**Wrong:**

```delphi
SendMessage(hWnd, WM_SETTEXT, 0, Integer(@MyCharArray));
```

Replace SetWindowLong/GetWindowLog with SetWindowLongPtr/GetWindowLongPtr for GWLP_HINSTANCE, GWLP_ID, GWLP_USERDATA, GWLP_HWNDDPARENT 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:**

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

**Wrong:**

```delphi
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:

\textbf{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: \url{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: \url{http://docwiki.embarcadero.com/RADStudio/en/Converting_32-bit_Delphi_Applications_to_64-bit_Windows}

\textbf{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.

\textbf{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 \textbf{Include remote debug symbols} project option is enabled.
To enable it, perform the following steps:

1. Open Project Options (in the main menu Project->Options).
2. In the Target combobox, select **Debug configuration - 64-bit Windows platform**. If there is no such option in the combobox, right click "Target Platforms" in Project Manager and select **Add platform**. After adding the 64-bit Windows platform, the **Debug configuration - 64-bit Windows platform** option will be available in the Target combobox.
3. Select **Linking** in the left part of the Project Options form.
4. enable the **Include remote debug symbols** option.

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

To enable remote debugging, perform the following steps:

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

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

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

For more information about working with Platform Assistant Server, please refer to [http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Running_the_Platform_Assistant_on_Windows](http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Running_the_Platform_Assistant_on_Windows)
4.21 Database Specific Aspects of 64-bit Development

InterBase and FireBird Connectivity Aspects

To work with InterBase and Firebird, IBDAC uses theirs 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 \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 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.

5 Reference

This page shortly describes units that exist in IBDAC.

Units

<table>
<thead>
<tr>
<th>Unit Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAccess</td>
<td>This unit contains base classes for accessing databases.</td>
</tr>
<tr>
<td>CRBatchMove</td>
<td>This unit contains implementation of the TCRBatchMove component.</td>
</tr>
<tr>
<td>CREncryption</td>
<td>This unit contains base classes for data encryption.</td>
</tr>
<tr>
<td>CRGrid</td>
<td>This unit contains the TCRDBGrid component.</td>
</tr>
<tr>
<td>CRVio</td>
<td>This unit contains the TIPVersion enumeration.</td>
</tr>
<tr>
<td>DAAlerter</td>
<td>This unit contains the base class for the TIBCAlerter component.</td>
</tr>
<tr>
<td>DADump</td>
<td>This unit contains the base</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>class for the TIBCDump component.</td>
<td></td>
</tr>
<tr>
<td>DALoader</td>
<td>This unit contains the base class for the TIBCLoader component.</td>
</tr>
<tr>
<td>DAScript</td>
<td>This unit contains the base class for the TIBCScript component.</td>
</tr>
<tr>
<td>DASQLMonitor</td>
<td>This unit contains the base class for the TIBCSQLMonitor component.</td>
</tr>
<tr>
<td>DBAccess</td>
<td>This unit contains base classes for most of the components.</td>
</tr>
<tr>
<td>Devart.Dac.DataAdapter</td>
<td>This unit contains implementation of the DADDataAdapter class.</td>
</tr>
<tr>
<td>Devart.IbDac.DataAdapter</td>
<td>This unit contains implementation of the IBCDataAdapter class.</td>
</tr>
<tr>
<td>IBC</td>
<td>This unit contains main components of IBDAC.</td>
</tr>
<tr>
<td>IBCAdmin</td>
<td>This unit contains implementation of components, used for InterBase/Firebird server administration.</td>
</tr>
<tr>
<td>IBCAlerter</td>
<td>This unit contains implementation of the TIBCAlerter component.</td>
</tr>
<tr>
<td>IBCArray</td>
<td>Description is not available at the moment.</td>
</tr>
<tr>
<td>IBCClasses</td>
<td>IBCClasses unit defines the following data type constants: dtDbKey dtFixedChar dtFixedWideChar</td>
</tr>
<tr>
<td>IBCConnectionPool</td>
<td>This unit contains the TIBCConnectionPoolManager class for managing connection pool.</td>
</tr>
<tr>
<td>IBCDataTypeMap</td>
<td>Description is not available at the moment.</td>
</tr>
</tbody>
</table>
IBCError

IBCError unit implements the EIBCErr class.

IBCLoader

This unit contains implementation of the TIBCLoader component.

IBCScript

This unit contains implementation of the TIBCScript component.

IBCSQLMonitor

This unit contains implementation of the TIBCSQLMonitor component.

IbDacVcl

This unit contains the visual constituent of IBDAC.

MemData

This unit contains classes for storing data in memory.

MemDS

This unit contains implementation of the TMemDataSet class.

VirtualDataSet

This unit contains implementation of the TVirtualDataSet component.

VirtualTable

This unit contains implementation of the TVirtualTable component.

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.1 CRAccess

This unit contains base classes for accessing databases.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRCursor</td>
<td>A base class for classes that work with database cursors.</td>
</tr>
</tbody>
</table>

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
**Enumerations**

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

**5.1.1 Classes**

Classes in the **CRAccess** unit.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRCursor</td>
<td>A base class for classes that work with database cursors.</td>
</tr>
</tbody>
</table>

**5.1.1.1 TCRCursor Class**

A base class for classes that work with database cursors.

For a list of all members of this type, see **TCRCursor** members.
Syntax

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

Remarks

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

Inheritance Hierarchy

```
TSharedObject
TCRCursor
```

© 1997-2021 Devart. All Rights Reserved.

5.1.1.1.1 Members

**TCRCursor** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>(inherited from <strong>TSharedObject</strong>) Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>(inherited from <strong>TSharedObject</strong>) Inches the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Release</td>
<td>(inherited from <strong>TSharedObject</strong>) Decrements the reference count.</td>
</tr>
</tbody>
</table>
5.1.2 Types

Types in the CRAccess unit.

Types

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

5.1.2.1 TBeforeFetchProc Procedure Reference

This type is used for the TCustomDADataset.BeforeFetch event.

Unit

CRAccess

Syntax

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

Parameters

Cancel

True, if the current fetch operation should be aborted.

5.1.3 Enumerations

Enumerations in the CRAccess unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRIsolationLevel</td>
<td>Specifies how to handle transactions containing database modifications.</td>
</tr>
<tr>
<td>TCRTransactionAction</td>
<td>Specifies the transaction</td>
</tr>
</tbody>
</table>
5.1.3.1 TCRIsolationLevel Enumeration

Specifies how to handle transactions containing database modifications.

Unit

CRAccess

Syntax

TCRIsolationLevel = (ilReadCommitted, ilReadUnCommitted, ilRepeatableRead, ilIsolated, ilSnapshot, ilCustom);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ilCustom</td>
<td>The parameters of the transaction are set manually in the Params property.</td>
</tr>
<tr>
<td>ilIsolated</td>
<td>The most restricted level of transaction isolation. Database server isolates data involved in current transaction by putting additional processing on range locks. Used to put aside all undesired effects observed in the concurrent accesses to the same set of data, but may lead to a greater latency at times of a congested database environment.</td>
</tr>
<tr>
<td>ilReadCommitted</td>
<td>Sets isolation level at which transaction cannot see changes made by outside transactions until they are committed. Only dirty reads (changes made by uncommitted transactions) are eliminated by this state of the isolation level. The default value.</td>
</tr>
<tr>
<td>ilReadUncommitted</td>
<td>The most unrestricted level of the transaction isolation. All types of data access interferences are possible. Mainly used for browsing database and to receive instant data with prospective changes.</td>
</tr>
<tr>
<td>ilRepeatableRead</td>
<td>Prevents concurrent transactions from modifying data in the current uncommitted transaction. This level eliminates dirty reads.</td>
</tr>
</tbody>
</table>
as well as nonrepeatable reads (repeatable reads of the same data in one transaction before and after outside transactions may have started and committed).

| ilSnapshot | Uses row versioning. Provides transaction-level read consistency. A data snapshot is taken when the snapshot transaction starts, and remains consistent for the duration of a transaction. |

### 5.1.3.2 **TCRTransactionAction Enumeration**

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

**Unit**

`CRAccess`

**Syntax**

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

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>taCommit</td>
<td>Transaction is committed.</td>
</tr>
<tr>
<td>taRollback</td>
<td>Transaction is rolled back.</td>
</tr>
</tbody>
</table>

### 5.1.3.3 **TCursorState Enumeration**

Used to set cursor state

**Unit**

`CRAccess`

**Syntax**

```
TCursorState = (csInactive, csOpen, csParsed, csPrepared, csBound, ...)
```
Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>csBound</td>
<td>Parameters bound</td>
</tr>
<tr>
<td>csExecuted</td>
<td>Statement successfully executed</td>
</tr>
<tr>
<td>csExecuteFetchAll</td>
<td>Set before FetchAll</td>
</tr>
<tr>
<td>csExecuting</td>
<td>Statement is set before executing</td>
</tr>
<tr>
<td>csFetched</td>
<td>Fetch finished or canceled</td>
</tr>
<tr>
<td>csFetching</td>
<td>Set on first</td>
</tr>
<tr>
<td>csFetchingAll</td>
<td>Set on the FetchAll start</td>
</tr>
<tr>
<td>csInactive</td>
<td>Default state</td>
</tr>
<tr>
<td>csOpen</td>
<td>statement open</td>
</tr>
<tr>
<td>csParsed</td>
<td>Statement parsed</td>
</tr>
<tr>
<td>csPrepared</td>
<td>Statement prepared</td>
</tr>
</tbody>
</table>

5.2 CRBatchMove

This unit contains implementation of the TCRBatchMove component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMove</td>
<td>Transfers records between datasets.</td>
</tr>
</tbody>
</table>

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMoveProgressEvent</td>
<td>This type is used for the TCRBatchMove.OnBatchMoveProgress event.</td>
</tr>
</tbody>
</table>

Enumerations
### 5.2.1 Classes

Classes in the **CRBatchMove** unit.

#### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMove</td>
<td>Transfers records between datasets.</td>
</tr>
</tbody>
</table>

#### TCRBatchMove Class

Transfers records between datasets.

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

**Unit**

**CRBatchMove**

**Syntax**

```pascal
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 IBDAC page.

---

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbortOnKeyViol</td>
<td>Used to specify whether the batch operation should be terminated immediately after key or integrity violation.</td>
</tr>
<tr>
<td>AbortOnProblem</td>
<td>Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.</td>
</tr>
<tr>
<td>ChangedCount</td>
<td>Used to get the number of records changed in the destination dataset.</td>
</tr>
<tr>
<td>CommitCount</td>
<td>Used to set the number of records to be batch moved before commit occurs.</td>
</tr>
<tr>
<td>Destination</td>
<td>Used to specify the destination dataset for the batch operation.</td>
</tr>
<tr>
<td>FieldMappingMode</td>
<td>Used to specify the way fields of destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.</td>
</tr>
</tbody>
</table>
### 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Performs the batch operation.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnBatchMoveProgress</td>
<td>Occurs when providing feedback to the user about</td>
</tr>
</tbody>
</table>
### 5.2.1.2 Properties

Properties of the **TCRBatchMove** class.

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

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

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AbortOnKeyViol</strong></td>
<td>Used to specify whether the batch operation should be terminated immediately after key or integrity violation.</td>
</tr>
<tr>
<td><strong>AbortOnProblem</strong></td>
<td>Used to specify whether the batch operation should be terminated immediately after key or integrity violation.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>CommitCount</strong></td>
<td>Used to set the number of records to be batch moved before commit occurs.</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>Used to specify the destination dataset for the batch operation.</td>
</tr>
<tr>
<td><strong>FieldMappingMode</strong></td>
<td>Used to specify the way fields of destination and source datasets will be mapped to each other if the <code>TCRBatchMove.Mappings</code> list is empty.</td>
</tr>
<tr>
<td><strong>Mappings</strong></td>
<td>Used to set field matching between source and destination datasets for the batch operation.</td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td>Used to set the type of the batch operation that will be executed after calling the <code>TCRBatchMove.Execute</code> method.</td>
</tr>
<tr>
<td><strong>RecordCount</strong></td>
<td>Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Used to specify the source dataset for the batch operation.</td>
</tr>
</tbody>
</table>

See Also
- [TCRBatchMove Class](#)
- [TCRBatchMove Class Members](#)
5.2.1.1.2.1  AbortOnKeyViol Property

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

Class
TCRBatchMove

Syntax

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

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

5.2.1.1.2.2  AbortOnProblem Property

Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.

Class
TCRBatchMove

Syntax

```plaintext
property AbortOnProblem: boolean default True;
```

Remarks
Use the AbortOnProblem property to specify whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit the specified Destination.
5.2.1.2.3 ChangedCount Property

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

Class

TCRBatchMove

Syntax

```pascal
property ChangedCount: Integer;
```

Remarks

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

5.2.1.2.4 CommitCount Property

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

Class

TCRBatchMove

Syntax

```pascal
property CommitCount: integer default 0;
```

Remarks

Use the CommitCount property to set the number of records to be batch moved before the commit occurs. If it is set to 0, the operation will be chunked to the number of records to fit 32 Kb.
5.2.1.1.2.5 Destination Property

Used to specify the destination dataset for the batch operation.

Class

TCRBatchMove

Syntax

```plaintext
property Destination: TDataSet;
```

Remarks

Specifies the destination dataset for the batch operation.

5.2.1.1.2.6 FieldMappingMode Property

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

Class

TCRBatchMove

Syntax

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

Remarks

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

5.2.1.1.2.7 KeyViolCount Property

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

TCRBatchMove

**Syntax**

```property
KeyViolCount: Integer;
```

**Remarks**

Use the KeyViolCount property to get the number of records that could not be replaced, added, deleted from the destination dataset because of integrity or key violations.

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

**See Also**

- `AbortOnKeyViol`

© 1997-2021 Devart. All Rights Reserved.

5.2.1.1.2.8 Mappings Property

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

**Class**

TCRBatchMove

**Syntax**

```property
Mappings: TStrings;
```

**Remarks**

Use the Mappings property to set field matching between the source and destination datasets for the batch operation. By default fields matching is based on their position in the datasets. To map the column ColName in the source dataset to the column with the same name in the destination dataset, use:

ColName

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

\[
\text{DestColName} = \text{SourceColName}
\]

5.2.1.1.2.9  Mode Property

Used to set the type of the batch operation that will be executed after calling the \texttt{Execute} method.

Class

\texttt{TCRBatchMove}

Syntax

\begin{verbatim}
property Mode: \texttt{TCRBatchMode} default bmAppend;
\end{verbatim}

Remarks

Use the Mode property to set the type of the batch operation that will be executed after calling the \texttt{Execute} method.

5.2.1.1.2.10  MovedCount Property

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

Class

\texttt{TCRBatchMove}

Syntax

\begin{verbatim}
property MovedCount: Integer;
\end{verbatim}

Remarks

Use the MovedCount property to get the number of records that were read from the source...
dataset during the batch operation. This number includes records that caused key or integrity violations or were trimmed.

5.2.1.1.2.11 ProblemCount Property

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

Class

TCRBatchMove

Syntax

```pascal
property ProblemCount: Integer;
```

Remarks

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

If AbortOnProblem is True, then ProblemCount will never exceed one, because the operation aborts when the problem occurs.

See Also

- AbortOnProblem

5.2.1.1.2.12 RecordCount Property

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

Class

TCRBatchMove

Syntax

```pascal
property RecordCount: Integer default 0;
```
Remarks

Determines the maximum number of records in the source dataset, that will be applied to the destination dataset. If it is set to 0, all records in the source dataset will be applied to the destination dataset, starting from the first record. If RecordCount is greater than 0, up to the RecordCount records are applied to the destination dataset, starting from the current record in the source dataset. If RecordCount exceeds the number of records left in the source dataset, batch operation terminates after reaching last record.

© 1997-2021 Devart. All Rights Reserved.

5.2.1.2.13 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.

© 1997-2021 Devart. All Rights Reserved.

5.2.1.3 Methods

Methods of the TCRBatchMove class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Performs the batch operation.</td>
</tr>
</tbody>
</table>
5.2.1.3.1 Execute Method

Performs the batch operation.

**Class**

TCRBatchMove

**Syntax**

```pascal
procedure Execute;
```

**Remarks**

Call the Execute method to perform the batch operation.

5.2.1.4 Events

Events of the TCRBatchMove class.

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

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnBatchMoveProgress</td>
<td>Occurs when providing feedback to the user about the batch operation in progress is needed.</td>
</tr>
</tbody>
</table>

**See Also**

- TCRBatchMove Class
- TCRBatchMove Class Members
5.2.1.4.1 OnBatchMoveProgress Event

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

Class

TCRBatchMove

Syntax

```pascal
property OnBatchMoveProgress: TCRBatchMoveProgressEvent;
```

Remarks

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

5.2.2 Types

Types in the CRBatchMove unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMoveProgressEvent</td>
<td>This type is used for the TCRBatchMove.OnBatchMoveProgress event.</td>
</tr>
</tbody>
</table>

5.2.2.1 TCRBatchMoveProgressEvent Procedure Reference

This type is used for the TCRBatchMove.OnBatchMoveProgress event.
CRBatchMove

Syntax

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

Parameters

- **Sender**
  - An object that raised the event.
- **Percent**
  - Percentage of the batch operation progress.

5.2.3 Enumerations

Enumerations in the CRBatchMove unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMode</td>
<td>Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.</td>
</tr>
<tr>
<td>TCRFieldMappingMode</td>
<td>Used to specify the way fields of the destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.</td>
</tr>
</tbody>
</table>

5.2.3.1 TCRBatchMode Enumeration

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

Unit
**CRBatchMove**

**Syntax**

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

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmAppend</td>
<td>Appends the records from the source dataset to the destination dataset. The default mode.</td>
</tr>
<tr>
<td>bmAppendUpdate</td>
<td>Replaces records in the destination dataset with the matching records from the source dataset. If there is no matching record in the destination dataset, the record will be appended to it.</td>
</tr>
<tr>
<td>bmDelete</td>
<td>Deletes records from the destination dataset if there are matching records in the source dataset.</td>
</tr>
<tr>
<td>bmUpdate</td>
<td>Replaces records in the destination dataset with the matching records from the source dataset.</td>
</tr>
</tbody>
</table>

---

5.2.3.2 **TCRFieldMappingMode** Enumeration

Used to specify the way fields of the destination and source datasets will be mapped to each other if the `TCRBatchMove.Mappings` list is empty.

**Unit**

**CRBatchMove**

**Syntax**

```
TCRFieldMappingMode = (mmFieldIndex, mmFieldName);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>mmFieldIndex</td>
<td>Specifies that the fields of the destination dataset will be mapped to the fields of the source dataset by field index.</td>
</tr>
<tr>
<td>mmFieldName</td>
<td>Mapping is performed by field names.</td>
</tr>
</tbody>
</table>

---

© 1997-2021 Devart. All Rights Reserved.

[Request Support]  [DAC Forum]  [Provide Feedback]
5.3 **CREncryption**

This unit contains base classes for data encryption.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCREncryptor</td>
<td>The class that performs data encryption and decryption in a client application using various encryption algorithms.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCREncDataHeader</td>
<td>Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td>TCREncryptionAlgorithm</td>
<td>Specifies the algorithm of data encryption.</td>
</tr>
<tr>
<td>TCRHashAlgorithm</td>
<td>Specifies the algorithm of generating hash data.</td>
</tr>
<tr>
<td>TCRInvalidHashAction</td>
<td>Specifies the action to perform on data fetching when hash data is invalid.</td>
</tr>
</tbody>
</table>

---

5.3.1 **Classes**

Classes in the **CREncryption** unit.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCREncryptor</td>
<td>The class that performs data encryption and decryption in a client application using</td>
</tr>
</tbody>
</table>
5.3.1.1 TCREncryptor Class

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

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

Unit

CREncryption

Syntax

TCREncryptor = class (TComponent);

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataHeader</td>
<td>Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td>EncryptionAlgorithm</td>
<td>Specifies the algorithm of data encryption.</td>
</tr>
<tr>
<td>HashAlgorithm</td>
<td>Specifies the algorithm of generating hash data.</td>
</tr>
<tr>
<td>InvalidHashAction</td>
<td>Specifies the action to perform on data fetching when hash data is invalid.</td>
</tr>
<tr>
<td>Password</td>
<td>Used to set a password that is used to generate a key for</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetKey</td>
<td>Sets a key, using which data is encrypted.</td>
</tr>
</tbody>
</table>

Properties of the TCREncryptor class.

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

Published

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

See Also

- TCREncryptor Class
- TCREncryptor Class Members
5.3.1.1.2.1 DataHeader Property

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

Class

TCREncryptor

Syntax

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

Remarks

Use DataHeader to specify whether the additional information is stored with the encrypted data. Default value is `ehTagAndHash`.

© 1997-2021 Devart. All Rights Reserved.

5.3.1.1.2.2 EncryptionAlgorithm Property

Specifies the algorithm of data encryption.

Class

TCREncryptor

Syntax

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

Remarks

Use EncryptionAlgorithm to specify the algorithm of data encryption. Default value is `eaBlowfish`.

© 1997-2021 Devart. All Rights Reserved.

5.3.1.1.2.3 HashAlgorithm Property

Specifies the algorithm of generating hash data.

Class

© 2021 Devart
TCREncryptor

Syntax

```plaintext
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.

Class

TCREncryptor

Syntax

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

Remarks

Use InvalidHashAction to specify the action to perform on data fetching when hash data is invalid. This property is used only if hash is stored with the encrypted data (the DataHeader property is set to ehTagAndHash). Default value is ihFail.

If the DataHeader property is set to ehTagAndHash, then on data fetching from a server the hash check is performed for each record. After data decryption its hash is calculated and compared with the hash stored in the field. If these values don’t coincide, it means that the stored data is incorrect, and depending on the value of the InvalidHashAction property one of the following actions is performed:

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

5.3.1.2.5 Password Property

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

Class

**TCREncryptor**

Syntax

```plaintext
property Password: string stored False;
```

Remarks

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

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

5.3.1.3 Methods

Methods of the **TCREncryptor** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SetKey</strong></td>
<td>Sets a key, using which data is encrypted.</td>
</tr>
</tbody>
</table>

See Also

- **TCREncryptor Class**
- **TCREncryptor Class Members**
Reserved.

5.3.1.3.1 SetKey Method

Sets a key, using which data is encrypted.

Class

TCREncryptor

Syntax

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

Parameters

Key
   Holds bytes that represent a key.
Offset
   Offset in bytes to the position, where the key begins.
Count
   Number of bytes to use from Key.

Remarks

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

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

© 1997-2021
Devart. All Rights Reserved.

5.3.2 Enumerations

Enumerations in the CREncryption unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCREncDataHeader</td>
<td>Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td>TCREncryptAlgorithm</td>
<td>Specifies the algorithm of</td>
</tr>
</tbody>
</table>
5.3.2.1 TCREncDataHeader Enumeration

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

**Unit**

**CREncryption**

**Syntax**

TCREncDataHeader = (ehTagAndHash, ehTag, ehNone);

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ehNone</td>
<td>No additional information is stored.</td>
</tr>
<tr>
<td>ehTag</td>
<td>GUID and the random initialization vector are stored with the encrypted data.</td>
</tr>
<tr>
<td>ehTagAndHash</td>
<td>Hash, GUID, and the random initialization vector are stored with the encrypted data.</td>
</tr>
</tbody>
</table>

5.3.2.2 TCREncryptionAlgorithm Enumeration

Specifies the algorithm of data encryption.

**Unit**

**CREncryption**

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

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>eaAES128</td>
<td>The AES encryption algorithm with key size of 128 bits is used.</td>
</tr>
<tr>
<td>eaAES192</td>
<td>The AES encryption algorithm with key size of 192 bits is used.</td>
</tr>
<tr>
<td>eaAES256</td>
<td>The AES encryption algorithm with key size of 256 bits is used.</td>
</tr>
<tr>
<td>eaBlowfish</td>
<td>The Blowfish encryption algorithm is used.</td>
</tr>
<tr>
<td>eaCast128</td>
<td>The CAST-128 encryption algorithm with key size of 128 bits is used.</td>
</tr>
<tr>
<td>eaRC4</td>
<td>The RC4 encryption algorithm is used.</td>
</tr>
<tr>
<td>eaTripleDES</td>
<td>The Triple DES encryption algorithm is used.</td>
</tr>
</tbody>
</table>

5.3.2.3 TCRHashAlgorithm Enumeration

Specifies the algorithm of generating hash data.

Unit

CREncryption

Syntax

TCRHashAlgorithm = (haSHA1, haMD5);

Values

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

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

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

Unit

CREncryption

Syntax

TCRInvalidHashAction = (ihFail, ihSkipData, ihIgnoreError);

Values

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

5.4 CRGrid

This unit contains the TCRDBGrid component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRDBGrid</td>
<td>Extends the functionality of the TDBGrid component.</td>
</tr>
</tbody>
</table>

5.4.1 Classes

Classes in the CRGrid unit.
5.4.1.1 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);

5.4.1.1.1 Members

TCRDBGrid class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.</td>
</tr>
<tr>
<td>Filtered</td>
<td>Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.</td>
</tr>
</tbody>
</table>
**LevelDelimiterChar**  
Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.

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

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

**OptionsMenu**  
TCRDBGrid has local menu situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User can change content of this menu using OptionsMenu property.

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivateFilterEdit</td>
<td>Use ActivateFilterEdit at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.</td>
</tr>
<tr>
<td>ActivateSearchEdit</td>
<td>Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.</td>
</tr>
<tr>
<td>AdjustColumns</td>
<td>Call AdjustColumns method at run-time to stretch all columns in a grid so that</td>
</tr>
</tbody>
</table>
they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.

**ApplyFilter**

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

**CalcTitleLevel**

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

**ClearFilters**

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

**ClearSorting**

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

**DataChanged**

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

**GetGridSize**

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

**GetTitleLevel**

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

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnGetCellParams</td>
<td>Write an OnGetCellParams event handler to provide custom background colors while the grid redraws individual cells.</td>
</tr>
</tbody>
</table>
5.4.1.1.2 Properties

Properties of the TCRDBGrid class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptionsMenu</td>
<td>TCRDBGrid has local menu is situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User can change content of this menu using OptionsMenu property.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.</td>
</tr>
<tr>
<td>Filtered</td>
<td>Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.</td>
</tr>
<tr>
<td>LevelDelimiterChar</td>
<td>Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.</td>
</tr>
<tr>
<td>OnMemoClick</td>
<td>Write an OnMemoClick event handler to provide</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>custom processing of Memo fields instead of built-in Memo field editor.</td>
<td></td>
</tr>
<tr>
<td>OptionsEx</td>
<td>OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.</td>
</tr>
</tbody>
</table>

**See Also**
- [TCRDBGrid Class](#)
- [TCRDBGrid Class Members](#)

5.4.1.1.2.1 Columns Property

To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.

**Class**

TCRDBGrid

**Syntax**

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

**Remarks**

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

5.4.1.1.2.2 Filtered Property

Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.
Class
TCRDBGrid

Syntax

**property** Filtered: boolean **default** True;

Remarks
Conditional statements of all columns are combined to form a single filter.

5.4.1.1.2.3 LevelDelimiterChar Property

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

Class
TCRDBGrid

Syntax

**property** LevelDelimiterChar: char **default** '|';

5.4.1.1.2.4 OnMemoClick Property

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

Class
TCRDBGrid

Syntax

**property** OnMemoClick: TOnMemoClick;

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
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 fin onto entire grid area.
- **dgeSummary** - displays status bar with summary information for every column. See TCRColumn.SummaryMode property on description of available options for individual columns. dgeSummary is mutually exclusive with dgeRecordCount option.
5.4.1.2.6 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;

5.4.1.3 Methods

Methods of the TCRDBGrid class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivateFilterEdit</td>
<td>Use ActivateFilterEdit at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.</td>
</tr>
<tr>
<td>ActivateSearchEdit</td>
<td>Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.</td>
</tr>
<tr>
<td>AdjustColumns</td>
<td>Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ApplyFilter</strong></td>
<td>Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.</td>
</tr>
<tr>
<td><strong>CalcTitleLevel</strong></td>
<td>Call CalcTitleLevel method to set Top and Bottom values of aRect parameter record depending on Level parameter value.</td>
</tr>
<tr>
<td><strong>ClearFilters</strong></td>
<td>Call CalcFilters method to clear all filter statements for grid columns.</td>
</tr>
<tr>
<td><strong>ClearSorting</strong></td>
<td>Call ClearSorting method to discard sorting previously applied to the grid data.</td>
</tr>
<tr>
<td><strong>DataChanged</strong></td>
<td>Call DataChanged method to update displayed data and status information for the grid.</td>
</tr>
<tr>
<td><strong>GetGridSize</strong></td>
<td>Call GetGridSize method to obtain width of a client area for the grid component.</td>
</tr>
<tr>
<td><strong>GetTitleLevel</strong></td>
<td>GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.</td>
</tr>
</tbody>
</table>

**See Also**
- TCRDBGrid Class
- TCRDBGrid Class Members
5.4.1.1.3.2 ActivateSearchEdit Method

Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.

5.4.1.3.3 AdjustColumns Method

Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.
**procedure** AdjustColumns;

**Remarks**

Visual feedback is immediate if OptionsEx property includes dgeLocalFilter option.

**Note:** Since filtering is performed at session level then other data-aware controls may also be updated.

© 1997-2021
Devart. All Rights Reserved.

### 5.4.1.3.4 ApplyFilter Method

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

**Class**

TCRDBGrid

**Syntax**

**procedure** ApplyFilter;

**Remarks**

Visual feedback is immediate if OptionsEx property includes dgeLocalFilter option.

**Note:** Since filtering is performed at session level then other data-aware controls may also be updated.

© 1997-2021
Devart. All Rights Reserved.

### 5.4.1.3.5 CalcTitleLevel Method

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

**Class**

TCRDBGrid

**Syntax**
procedure CalcTitleLevel(Level: integer; var aRect: TRect);

Parameters
Level
aRect

Class
TCRDBGrid

Syntax
procedure ClearFilters;

Class
TCRDBGrid

Syntax
procedure ClearSorting;

Class
TCRDBGrid

Syntax
procedure DataChanged;

© 1997-2021 Devart. All Rights Reserved.
5.4.1.3.9  GetGridSize Method

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

Class

TCRDBGrid

Syntax

```delphi
function GetGridSize: integer;
```

Remarks

Client area accommodates cell values only without column grid lines.

5.4.1.3.10  GetTitleLevel Method

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

Class

TCRDBGrid

Syntax

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

Parameters

Level

© 1997-2021 Devart. All Rights Reserved.
5.4.1.1.4 Events

Events of the TCRDBGrid class.

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

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnGetCellParams</td>
<td>Write an OnGetCellParams event handler to provide custom background colors while the grid redraws individual cells.</td>
</tr>
</tbody>
</table>

See Also

- TCRDBGrid Class
- TCRDBGrid Class Members

5.4.1.4.1 OnGetCellParams Event

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

Class

TCRDBGrid

Syntax

```property
OnGetCellParams: TGetCellParamsEvent;
```

5.5 CRVio

This unit contains the TIPVersion enumeration.

Enumerations
### 5.5.1 Enumerations

Enumerations in the **CRVio** unit.

#### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIPVersion</td>
<td>Specifies Internet Protocol version.</td>
</tr>
</tbody>
</table>

#### TIPVersion Enumeration

Specifies Internet Protocol version.

**Unit**

**CRVio**

**Syntax**

\[
\text{TIPVersion} = (\text{ivIPv4}, \text{ivIPv6}, \text{ivIPBoth});
\]

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivIPBoth</td>
<td>Specifies that either IPv6 or IPv4 Internet Protocol version is used</td>
</tr>
<tr>
<td>ivIPv4</td>
<td>Specifies that the IPv4 Internet Protocol version is used</td>
</tr>
<tr>
<td>ivIPv6</td>
<td>Specifies that the IPv6 Internet Protocol version is used</td>
</tr>
</tbody>
</table>

**Remarks**

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

See Also
- TIBConnectionOptions.IPVersion

5.6 **DAAlerter**

This unit contains the base class for the TIBCAlerter component.

### Classes

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

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAlarterErrorEvent</td>
<td>This type is used for the TDAAlerter.OnError event.</td>
</tr>
</tbody>
</table>

5.6.1 **Classes**

Classes in the **DAAlerter** unit.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAAlerter</td>
<td>A base class that defines functionality for database event notification.</td>
</tr>
</tbody>
</table>
5.6.1.1  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
TDAAlerter

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.

© 1997-2021 Devart. All Rights Reserved.

5.6.1.1.1  Members

TDAAlerter class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if TDAAlerter waits for messages.</td>
</tr>
<tr>
<td>AutoRegister</td>
<td>Used to automatically register events whenever connection opens.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify the connection for TDAAlerter.</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SendEvent</strong></td>
<td>Sends an event with Name and content Message.</td>
</tr>
<tr>
<td><strong>Start</strong></td>
<td>Starts waiting process.</td>
</tr>
<tr>
<td><strong>Stop</strong></td>
<td>Stops waiting process.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnError</strong></td>
<td>Occurs if an exception occurs in waiting process</td>
</tr>
</tbody>
</table>

5.6.1.1.2 Properties

Properties of the **TDAAlerter** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong></td>
<td>Used to determine if TDAAlerter waits for messages.</td>
</tr>
<tr>
<td><strong>AutoRegister</strong></td>
<td>Used to automatically register events whenever connection opens.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify the connection for TDAAlerter.</td>
</tr>
</tbody>
</table>

See Also

- [TDAAlerter Class](#)
- [TDAAlerter Class Members](#)
5.6.1.2.1 Active Property

Used to determine if TDAlerter waits for messages.

Class

TDAlerter

Syntax

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

Remarks

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

See Also

- Start
- Stop

© 1997-2021 Devart. All Rights Reserved.

5.6.1.2.2 AutoRegister Property

Used to automatically register events whenever connection opens.

Class

TDAlerter

Syntax

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

Remarks

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

© 1997-2021 Devart. All Rights Reserved.
### 5.6.1.1.2.3 Connection Property

Used to specify the connection for TDAAlerter.

**Class**

TDAAlerter

**Syntax**

```property
connection: TCustomDAConnection;
```

**Remarks**

Use the Connection property to specify the connection for TDAAlerter.

**See Also**

- TIBConnection

© 1997-2021 Devart. All Rights Reserved.

### 5.6.1.1.3 Methods

Methods of the TDAAlerter class.

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

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendEvent</td>
<td>Sends an event with Name and content Message.</td>
</tr>
<tr>
<td>Start</td>
<td>Starts waiting process.</td>
</tr>
<tr>
<td>Stop</td>
<td>Stops waiting process.</td>
</tr>
</tbody>
</table>

**See Also**

- TDAAlerter Class
- TDAAlerter Class Members

© 1997-2021 Devart. All Rights Reserved.
5.6.1.3.1 SendEvent Method

Sends an event with Name and content Message.

Class
TDAAlerter

Syntax

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

Parameters

EventName
Holds the event name.

Message
Holds the content Message of the event.

Remarks

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

5.6.1.3.2 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
5.6.1.1.3.3  Stop Method

Stops waiting process.

Class

**TDAAlerter**

Syntax

```pascal
procedure Stop;
```

Remarks

Call Stop method to end waiting process.

See Also

- **Start**

5.6.1.4  Events

Events of the **TDAAlerter** class.

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

Public

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

See Also

- **TDAAlerter Class**
- **TDAAlerter Class Members**
5.6.1.4.1 OnError Event

Occurs if an exception occurs in waiting process

Class

TDAAlerter

Syntax

```pascal
property OnError: TAlerrorErrorEvent;
```

Remarks

The OnError event occurs if an exception occurs in waiting process. Alerter stops in this case. The exception can be accessed using the E parameter.

5.6.2 Types

Types in the DAAlerter unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAlerrorErrorEvent</td>
<td>This type is used for the TDAAlerter.OnError event.</td>
</tr>
</tbody>
</table>

5.6.2.1 TAlerrorErrorEvent Procedure Reference

This type is used for the TDAAlerter.OnError event.

Unit

DAAlerter

Syntax
TAlertErrorEvent = procedure (Sender: TDAAlert; E: Exception) of object;

Parameters

Sender
An object that raised the event.

E
Exception object.

5.7 DADump

This unit contains the base class for the TIBCDump component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDADump</td>
<td>A base class that defines functionality for descendant classes that dump database objects to a script.</td>
</tr>
<tr>
<td>TDADumpOptions</td>
<td>This class allows setting up the behaviour of the TDADump class.</td>
</tr>
</tbody>
</table>

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDABackupProgressEvent</td>
<td>This type is used for the TDADump.OnBackupProgress event.</td>
</tr>
<tr>
<td>TDARestoreProgressEvent</td>
<td>This type is used for the TDADump.OnRestoreProgress event.</td>
</tr>
</tbody>
</table>
5.7.1 Classes

Classes in the DADump unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDADump</td>
<td>A base class that defines functionality for descendant classes that dump database objects to a script.</td>
</tr>
<tr>
<td>TDADumpOptions</td>
<td>This class allows setting up the behaviour of the TDADump class.</td>
</tr>
</tbody>
</table>

5.7.1.1 TDADump Class

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

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

Unit

DADump

Syntax

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

Remarks

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

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

**TDADump** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Used to specify a connection object that will be used to connect to a data store.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of a TDADump component.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to set or get the dump script.</td>
</tr>
<tr>
<td>TableNames</td>
<td>Used to set the names of the tables to dump.</td>
</tr>
</tbody>
</table>

**Methods**

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

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

© 1997-2021 Devart. All Rights Reserved.

## Properties

Properties of the **TDADump** class.

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

## Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object that will be used to connect to a data store.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of a TDADump component.</td>
</tr>
</tbody>
</table>
## Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Debug</strong></td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to set or get the dump script.</td>
</tr>
<tr>
<td><strong>TableNames</strong></td>
<td>Used to set the names of the tables to dump.</td>
</tr>
</tbody>
</table>

### See Also
- **TDADump Class**
- **TDADump Class Members**

### 5.7.1.2.1 Connection Property

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

### Class
- **TDADump**

### Syntax

```property
Connection: TCustomDAConnection;
```

### Remarks

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

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

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

### See Also
5.7.1.1.2.2 Debug Property

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

Class

TDADump

Syntax

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

Remarks

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

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

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

See Also

- TCustomDADataset.Debug
- TCustomDASQL.Debug
property Options: TDADumpOptions;

Remarks

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

Descriptions of all options are in the table below.

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

5.7.1.1.2.4 SQL Property

Used to set or get the dump script.

Class

TDADump

Syntax

property SQL: TStrings;

Remarks

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

See Also

- Restore
- Backup
5.7.1.1.2.5 TableNames Property

Used to set the names of the tables to dump.

Class

TDADump

Syntax

```plaintext
property TableNames: string;
```

Remarks

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

See Also

- Backup

5.7.1.3 Methods

Methods of the TDADump class.

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

Public

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

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

### 5.7.1.3.1 Backup Method

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

### Class

[TDADump](#)

### Syntax

```plaintext
procedure Backup;
```

### Remarks

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

### See Also
- [SQL](#)
- [Restore](#)
- [BackupToFile](#)
- [BackupToStream](#)
- [BackupQuery](#)
5.7.1.3.2 BackupQuery Method

Dumps the results of a particular query.

Class
TDADump

Syntax
```
procedure BackupQuery(const Query: string);
```

Parameters
- **Query**
  Holds a query used for data selection.

Remarks
Call the BackupQuery method to dump the results of a particular query. Query must be a valid select statement. If this query selects data from several tables, only data of the first table in the from list will be dumped.

See Also
- Restore
- Backup
- BackupToFile
- BackupToStream

5.7.1.3.3 BackupToFile Method

Dumps database objects to the specified file.

Class
TDADump

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

Parameters
**FileName**
Holds the file name to dump database objects to.

**Query**
Your query to receive the data for dumping.

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

**See Also**
- [RestoreFromStream](#)
- [Backup](#)
- [BackupToStream](#)

---

5.7.1.3.4 BackupToStream Method

Dumps database objects to the stream.

**Class**
TDADump

**Syntax**
```delphi
procedure BackupToStream(Stream: TStream; const Query: string = '');
```

**Parameters**
- **Stream**
  Holds the stream to dump database objects to.
- **Query**
  Your query to receive the data for dumping.

**Remarks**
Call the BackupToStream method to dump database objects to the stream.

**See Also**
- [RestoreFromStream](#)
- [Backup](#)
5.7.1.1.3.5 Restore Method

Executes a script contained in the SQL property.

Class
TDADump

Syntax
```
procedure Restore;
```

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

See Also
- RestoreFromFile
- RestoreFromStream
- Backup
- SQL

5.7.1.1.3.6 RestoreFromFile Method

Executes a script from a file.

Class
TDADump

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

Parameters

`FileName`
Holds the file name to execute a script from.
Remarks

Call the RestoreFromFile method to execute a script from the specified file.

See Also

- Restore
- RestoreFromFile
- BackupToFile

5.7.1.1.3.7 RestoreFromStream Method

Executes a script received from the stream.

Class

TDADump

Syntax

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

Parameters

- **Stream**
  
  Holds a stream to receive a script to be executed.

Remarks

Call the RestoreFromStream method to execute a script received from the stream.

See Also

- Restore
- RestoreFromFile
- BackupToStream

© 1997-2021

Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
5.7.1.1.4 Events

Events of the **TDADump** class.

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

### Published

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

**See Also**
- **TDADump Class**
- **TDADump Class Members**
### TDADump

#### Syntax

```pascal
property OnBackupProgress: TDABackupProgressEvent;
```

#### Remarks

The OnBackupProgress event occurs several times during the dumping process of the `Backup`, `M:Devart.Dac.TDADump.BackupToFile(System.String)`, or `M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream)` method execution and indicates its progress. `ObjectName` parameter indicates the name of the currently dumping database object. `ObjectNum` shows the number of the current database object in the backup queue starting from zero. `ObjectCount` shows the quantity of database objects to dump. `Percent` parameter shows the current percentage of the current table data dumped, not the current percentage of the entire dump process.

#### See Also

- `Backup`
- `BackupToFile`
- `BackupToStream`

---

5.7.1.1.4.2 OnError Event

Occurs when InterBase raises some error on `Restore`.

#### Class

TDADump

#### Syntax

```pascal
property OnError: TOnErrorEvent;
```

#### Remarks

The OnError event occurs when InterBase 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`.
5.7.1.4.3 OnRestoreProgress Event

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

Class
TDADump

Syntax

```plaintext
property OnRestoreProgress: TDARestoreProgressEvent;
```

Remarks

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

See Also
- Restore
- RestoreFromFile
- RestoreFromStream

5.7.1.2 TDADumpOptions Class

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

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

Unit
DADump

Syntax
5.7.1.2.1 Members

TDADumpOptions class overview.

Properties

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

5.7.1.2.2 Properties

Properties of the TDADumpOptions class.

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

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddDrop</td>
<td>Used to add drop statements to a script before creating statements.</td>
</tr>
<tr>
<td>CompleteInsert</td>
<td>Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.</td>
</tr>
</tbody>
</table>
InterBase Data Access Components

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenerateHeader</td>
<td>Used to add a comment header to a script.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used for TDADump to quote all database object names in generated SQL statements.</td>
</tr>
</tbody>
</table>

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

---

### 5.7.1.2.2.1 AddDrop Property

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

**Class**

**TDADumpOptions**

**Syntax**

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

**Remarks**

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

---

### 5.7.1.2.2.2 CompleteInsert Property

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

**Class**

**TDADumpOptions**
Syntax

property CompleteInsert: boolean default False;

Remarks

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

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

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

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

5.7.1.2.2.3 GenerateHeader Property

Used to add a comment header to a script.

Class

TDADumpOptions

Syntax

property GenerateHeader: boolean default True;

Remarks

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

© 1997-2021

Request Support      DAC Forum      Provide Feedback

5.7.1.2.2.4 QuoteNames Property

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

Class

TDADumpOptions
**Syntax**

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

**Remarks**

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

**5.7.2 Types**

Types in the **DADump** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDABackupProgressEvent</td>
<td>This type is used for the <strong>TDADump.OnBackupProgress</strong> event.</td>
</tr>
<tr>
<td>TDARestoreProgressEvent</td>
<td>This type is used for the <strong>TDADump.OnRestoreProgress</strong> event.</td>
</tr>
</tbody>
</table>

**5.7.2.1 TDABackupProgressEvent Procedure Reference**

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

**Unit**

**DADump**

**Syntax**

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

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

**ObjectName**
The name of the currently dumping database object.

**ObjectNum**
The number of the current database object in the backup queue starting from zero.

**ObjectCount**
The quantity of database objects to dump.

**Percent**
The current percentage of the current table data dumped.

### 5.7.2.2 TDARestoreProgressEvent Procedure Reference

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

**Unit**
DADump

**Syntax**

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

**Parameters**

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

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

### 5.8 DALoader

This unit contains the base class for the TIBCLoader component.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

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

Classes in the **DALoader** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAColumn</td>
<td>Represents the attributes for column loading.</td>
</tr>
<tr>
<td>TDAColumns</td>
<td>Holds a collection of <strong>TDAColumn</strong> objects.</td>
</tr>
<tr>
<td>TDALoader</td>
<td>This class allows loading external data into database.</td>
</tr>
<tr>
<td>TDALoaderOptions</td>
<td>Allows loading external data into database.</td>
</tr>
</tbody>
</table>

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAPutDataEvent</td>
<td>This type is used for the <strong>TDALoader.OnPutData</strong> event.</td>
</tr>
<tr>
<td>TGetColumnDataEvent</td>
<td>This type is used for the <strong>TDALoader.OnGetColumnData</strong> event.</td>
</tr>
<tr>
<td>TLoaderProgressEvent</td>
<td>This type is used for the <strong>TDALoader.OnProgress</strong> event.</td>
</tr>
</tbody>
</table>
5.8.1.1 TDAColumn Class

 Represents the attributes for column loading.

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

 Unit

 DALoader

 Syntax

 TDAColumn = class(TCollectionItem);

 Remarks

 Each TDALoader uses TDAColumns to maintain a collection of TDAColumn objects. TDAColumn object represents the attributes for column loading. Every TDAColumn object corresponds to one of the table fields with the same name as its TDAColumn.Name property.

 To create columns at design-time use the column editor of the TDALoader component.

 See Also

 • TDALoader
 • TDAColumns

 © 1997-2021
 Devart. All Rights Reserved.

 5.8.1.1.1 Members

 TDAColumn class overview.

 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldType</td>
<td>Used to specify the types of values that will be loaded.</td>
</tr>
<tr>
<td>Name</td>
<td>Used to specify the field name of loading table.</td>
</tr>
</tbody>
</table>

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

Properties of the TDAColumn class.

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

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldType</td>
<td>Used to specify the types of values that will be loaded.</td>
</tr>
<tr>
<td>Name</td>
<td>Used to specify the field name of loading table.</td>
</tr>
</tbody>
</table>

See Also
- TDAColumn Class
- TDAColumn Class Members

5.8.1.2.1 FieldType Property

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

Class
TDAColumn

Syntax

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

Remarks

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

Used to specify the field name of loading table.

Class

TDAColumn

Syntax

```
property Name: string;
```

Remarks

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

See Also

- FieldType

5.8.1.2 TDAColumns Class

Holds a collection of TDAColumn objects.

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

Unit

DALoader

Syntax

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

Remarks

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

See Also

- TDALoader
• **TDAColumn**

© 1997-2021
Devart. All Rights Reserved.

### 5.8.1.2.1 Members

**TDAColumns** class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Items</strong></td>
<td>Used to access individual columns.</td>
</tr>
</tbody>
</table>

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Items</strong></td>
<td>Used to access individual columns.</td>
</tr>
</tbody>
</table>

**See Also**

• **TDAColumns Class**
• **TDAColumns Class Members**

© 1997-2021
Devart. All Rights Reserved.
TDAColumns

Syntax

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

Parameters

- **Index**
  
  Holds the Index of TDAColumn to refer to.

Remarks

Use the Items property to access individual columns. The value of the Index parameter corresponds to the Index property of TDAColumn.

See Also

- TDAColumn

© 1997-2021 Devart. All Rights Reserved.

5.8.1.3 TDALoader Class

This class allows loading external data into database.

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

Unit

DALoader

Syntax

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

Remarks

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

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
5.8.1.3.1 Members

**TDALoader** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Used to add a TDAColumn object for each field that will be loaded.</td>
</tr>
<tr>
<td>Connection</td>
<td>property. Used to specify TCustomDAConnection in which TDALoader will be executed.</td>
</tr>
<tr>
<td>TableName</td>
<td>Used to specify the name of the table to which data will be loaded.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateColumns</td>
<td>Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName.</td>
</tr>
<tr>
<td>Load</td>
<td>Starts loading data.</td>
</tr>
<tr>
<td>LoadFromDataSet</td>
<td>Loads data from the specified dataset.</td>
</tr>
<tr>
<td>PutColumnData</td>
<td>Overloaded. Puts the value of individual columns.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnGetColumnData</td>
<td>Occurs when it is needed to put column values.</td>
</tr>
<tr>
<td>OnProgress</td>
<td>Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.</td>
</tr>
</tbody>
</table>
OnPutData | Occurs when putting loading data by rows is needed.

5.8.1.3.2 Properties

Properties of the TDALoader class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Used to add a TDAColumn object for each field that will be loaded.</td>
</tr>
<tr>
<td>Connection</td>
<td>property. Used to specify TCustomDACConnection in which TDALoader will be executed.</td>
</tr>
<tr>
<td>TableName</td>
<td>Used to specify the name of the table to which data will be loaded.</td>
</tr>
</tbody>
</table>

See Also
- TDALoader Class
- TDALoader Class Members

5.8.1.3.2.1 Columns Property

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

Class

TDALoader

Syntax
property Columns: TDAColumns stored IsColumnsStored;

Remarks
Use the Columns property to add a TDAColumn object for each field that will be loaded.

See Also
• TDAColumns

Class
TDALoader

Syntax
property Connection: TCustomDAConnection;

Remarks
Use the Connection property to specify TCustomDAConnection in which TDALoader will be executed. If Connection is not connected, the Load method calls TCustomDAConnection.Connect.

See Also
• TCustomDAConnection

5.8.1.3.2.2 Connection Property

5.8.1.3.2.3 TableName Property

Used to specify the name of the table to which data will be loaded.

Class
TDALoader
Syntax

```plaintext
property TableName: string;
```

Remarks

Set the TableName property to specify the name of the table to which data will be loaded. Add TDAColumn objects to `Columns` for the fields that are needed to be loaded.

See Also

- TDAColumn
- TCustomDACConnection.GetTableNames

© 1997-2021 Devart. All Rights Reserved.

### 5.8.1.3.3 Methods

Methods of the `TDALoader` class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateColumns</td>
<td>Creates <code>TDAColumn</code> objects for all fields of the table with the same name as <code>TDALoader.TableName</code>.</td>
</tr>
<tr>
<td>Load</td>
<td>Starts loading data.</td>
</tr>
<tr>
<td>LoadFromDataSet</td>
<td>Loads data from the specified dataset.</td>
</tr>
<tr>
<td>PutColumnData</td>
<td>Overloaded. Puts the value of individual columns.</td>
</tr>
</tbody>
</table>

See Also

- `TDALoader Class`
- `TDALoader Class Members`

© 1997-2021 Devart. All Rights Reserved.
5.8.1.3.3.1  CreateColumns Method

Creates TDAColumn objects for all fields of the table with the same name as TableName.

Class

TDALoader

Syntax

```pascal
procedure CreateColumns;
```

Remarks

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

See Also

- TDAColumn
- TableName

5.8.1.3.3.2  Load Method

Starts loading data.

Class

TDALoader

Syntax

```pascal
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
5.8.1.3.3.3 LoadFromDataSet Method

Loads data from the specified dataset.

**Class**

TDALoader

**Syntax**

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

**Parameters**

*DataSet*

Holds the dataset to load data from.

**Remarks**

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

© 1997-2021 Devart. All Rights Reserved.

5.8.1.3.3.4 PutColumnData Method

Puts the value of individual columns.

**Class**

TDALoader

**Overload List**

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

© 2021 Devart
### PutColumnData

**Syntax**

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

**Parameters**

- **Col**
  - Holds the index of a loading column. The first column has index 0.

- **Row**
  - Holds the number of loading row. Row starts from 1.

- **Value**
  - Holds the column value.

**Remarks**

Call the PutColumnData method to put the value of individual columns. The Col parameter indicates the index of loading column. The first column has index 0. The Row parameter indicates the number of the loading row. Row starts from 1.

This overloaded method works faster because it searches the right index by its index, not by the index name.

The value of a column should be assigned to the Value parameter.

**See Also**

- `TDALoader.OnPutData`
Puts the value of individual columns by the column name.

Class

**TDALoader**

Syntax

```pascal
procedure PutColumnData(const ColName: string; Row: integer; const Value: variant); overload;
```

Parameters

- **ColName**
  - Holds the name of a loading column.

- **Row**
  - Holds the number of loading row. Row starts from 1.

- **Value**
  - Holds the column value.

5.8.1.3.4 Events

Events of the **TDALoader** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnGetColumnData</td>
<td>Occurs when it is needed to put column values.</td>
</tr>
<tr>
<td>OnProgress</td>
<td>Occurs if handling data loading progress of the <strong>TDALoader.LoadFromDataSet</strong> method is needed.</td>
</tr>
<tr>
<td>OnPutData</td>
<td>Occurs when putting loading data by rows is needed.</td>
</tr>
</tbody>
</table>

See Also

- [TDALoader Class](#)
- [TDALoader Class Members](#)
5.8.1.3.4.1 OnGetColumnData Event

Occurs when it is needed to put column values.

Class

**TDALoader**

Syntax

```property
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
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;
```
5.8.1.3.4.2 OnProgress Event

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

Class
TDALoader

Syntax

property OnProgress: TLoaderProgressEvent;

Remarks
Add a handler to this event if you want to handle data loading progress of the LoadFromDataSet method.

See Also
• LoadFromDataSet

5.8.1.3.4.3 OnPutData Event

Occurs when putting loading data by rows is needed.

Class
TDALoader

Syntax

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. It is more effective way to load data in comparison with using OnGetColumnData. OnPutData event handler must send column data by TDALoader.PutColumnData method. TDALoader will flush data to Oracle when it is needed.

See Also
- TDALoader.PutColumnData
- Load
- OnGetColumnData

5.8.1.4 TDALoaderOptions Class

Allows loading external data into database.

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

Unit
DALoader

Syntax
TDALoaderOptions = class(TPersistent);

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseBlankValues</td>
<td>Forces IBDAC to fill the buffer with null values after loading a row to the database.</td>
</tr>
</tbody>
</table>
5.8.1.4.2 Properties

Properties of the **TDALoaderOptions** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UseBlankValues</strong></td>
<td>Forces IBDAC to fill the buffer with null values after loading a row to the database.</td>
</tr>
</tbody>
</table>

See Also

- **TDALoaderOptions Class**
- **TDALoaderOptions Class Members**

5.8.1.4.2.1 **UseBlankValues Property**

Forces IBDAC 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 IBDAC to fill the buffer with null values after loading a row to the database.
5.8.2 Types

Types in the **DALoader** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAPutDataEvent</td>
<td>This type is used for the <strong>TDALoader.OnPutData</strong> event.</td>
</tr>
<tr>
<td>TGetColumnDataEvent</td>
<td>This type is used for the <strong>TDALoader.OnGetColumnData</strong> event.</td>
</tr>
<tr>
<td>TLoaderProgressEvent</td>
<td>This type is used for the <strong>TDALoader.OnProgress</strong> event.</td>
</tr>
</tbody>
</table>

5.8.2.1 TDAPutDataEvent Procedure Reference

This type is used for the **TDALoader.OnPutData** event.

**Unit**

**DALoader**

**Syntax**

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

**Parameters**

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

5.8.2.2 TGetColumnDataEvent Procedure Reference

This type is used for the **TDALoader.OnGetColumnData** event.

**Unit**
**DALoader**

**Syntax**

```
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.

© 1997-2021 Devart. All Rights Reserved.

5.8.2.3 **TLoaderProgressEvent Procedure Reference**

This type is used for the `TDALoader.OnProgress` event.

**Unit**

`DALoader`

**Syntax**

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

**Parameters**

- **Sender**: An object that raised the event.
- **Percent**: Percentage of the load operation progress.
5.9 DAScript

This unit contains the base class for the TIBCScript component.

Classes

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

Types

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

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TErrorAction</td>
<td>Indicates the action to take when the OnError handler exits.</td>
</tr>
</tbody>
</table>
5.9.1 **Classes**

Classes in the DAScript unit.

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

**5.9.1.1 TDAScript Class**

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

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

**Unit**

DAScript

**Syntax**

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

**Remarks**

Often it is necessary to execute several SQL statements one by one. This can be performed using a lot of components such as TCustomDASQL descendants. Usually it isn't the best solution. With only one TDAScript descendent 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, (for example, CREATE TRIGGER or CREATE PROCEDURE) only slash. Note that slash must be the first character in line.

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

See Also
- TCustomDASQL

---

5.9.1.1.1 Members

**TDAScript** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify the connection in which the script will be executed.</td>
</tr>
<tr>
<td><strong>DataSet</strong></td>
<td>Refers to a dataset that holds the result set of query execution.</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>Used to display the script execution and all its parameter values.</td>
</tr>
<tr>
<td><strong>Delimiter</strong></td>
<td>Used to set the delimiter string that separates script statements.</td>
</tr>
<tr>
<td><strong>EndLine</strong></td>
<td>Used to get the current statement last line number in a script.</td>
</tr>
<tr>
<td><strong>EndOffset</strong></td>
<td>Used to get the offset in the last line of the current statement.</td>
</tr>
<tr>
<td><strong>EndPos</strong></td>
<td>Used to get the end position of the current statement.</td>
</tr>
<tr>
<td><strong>Macros</strong></td>
<td>Used to change SQL script text in design- or run-time easily.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to get or set script text.</td>
</tr>
<tr>
<td><strong>StartLine</strong></td>
<td>Used to get the current statement start line number in a script.</td>
</tr>
<tr>
<td><strong>StartOffset</strong></td>
<td>Used to get the offset in the</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BreakExec</td>
<td>Stops script execution.</td>
</tr>
<tr>
<td>ErrorOffset</td>
<td>Used to get the offset of the statement if the Execute method raised an exception.</td>
</tr>
<tr>
<td>Execute</td>
<td>Executes a script.</td>
</tr>
<tr>
<td>ExecuteFile</td>
<td>Executes SQL statements contained in a file.</td>
</tr>
<tr>
<td>ExecuteNext</td>
<td>Executes the next statement in the script and then stops.</td>
</tr>
<tr>
<td>ExecuteStream</td>
<td>Executes SQL statements contained in a stream object.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
</tbody>
</table>

Events

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

Properties of the **TDAScript** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify the connection in which the script will be executed.</td>
</tr>
<tr>
<td><strong>DataSet</strong></td>
<td>Refers to a dataset that holds the result set of query execution.</td>
</tr>
<tr>
<td><strong>EndLine</strong></td>
<td>Used to get the current statement last line number in a script.</td>
</tr>
<tr>
<td><strong>EndOffset</strong></td>
<td>Used to get the offset in the last line of the current statement.</td>
</tr>
<tr>
<td><strong>EndPos</strong></td>
<td>Used to get the end position of the current statement.</td>
</tr>
<tr>
<td><strong>StartLine</strong></td>
<td>Used to get the current statement start line number in a script.</td>
</tr>
<tr>
<td><strong>StartOffset</strong></td>
<td>Used to get the offset in the first line of the current statement.</td>
</tr>
<tr>
<td><strong>StartPos</strong></td>
<td>Used to get the start position of the current statement in a script.</td>
</tr>
<tr>
<td><strong>Statements</strong></td>
<td>Contains a list of statements obtained from the SQL property.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Debug</strong></td>
<td>Used to display the script execution and all its</td>
</tr>
</tbody>
</table>
### Connection Property

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

**Class**  
**TDAScript**

**Syntax**

```property
connection: TCustomDAConnection;
```

**Remarks**

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

Set at design-time by selecting from the list of provided `TCustomDAConnection` objects.

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

**See Also**

- `TCustomDAConnection`
5.9.1.1.2.2 DataSet Property

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

Class

**TDAScript**

Syntax

```property` DataSet: TCustomDADataset;```

Remarks

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

See Also

- `ExecuteNext`
- `Execute`

5.9.1.1.2.3 Debug Property

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

Class

**TDAScript**

Syntax

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

Remarks

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

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

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

5.9.1.1.2.4 Delimiter Property

Used to set the delimiter string that separates script statements.

Class

TDAScript

Syntax

```plaintext
property Delimiter: string stored IsDelimiterStored;
```

Remarks

Use the Delimiter property to set the delimiter string that separates script statements. By default it is semicolon (;). You can use slash (/) to separate statements that can contain semicolon (for example, CREATE TRIGGER or CREATE PROCEDURE) if the Delimiter property's default value is semicolon. Note that slash must be the first character in line.

5.9.1.1.2.5 EndLine Property

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

Class

TDAScript

Syntax

```plaintext
property EndLine: Int64;
```

Remarks

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

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

Class
TDAScript

Syntax

```property
EndOffset: Int64;
```

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

5.9.1.1.2.7 EndPos Property

Used to get the end position of the current statement.

Class
TDAScript

Syntax

```property
EndPos: Int64;
```

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

5.9.1.1.2.8 Macros Property

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

Class
TDAScript
Syntax

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

Remarks

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

See Also

- TMacro
- MacroByName

© 1997-2021
Devart. All Rights Reserved.

5.9.1.2.9 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.

© 1997-2021
Devart. All Rights Reserved.

5.9.1.2.10 StartLine Property

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

Class
**TDAScript**

**Syntax**

```pascal
property StartLine: Int64;
```

**Remarks**

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

© 1997-2021 Devart. All Rights Reserved.

5.9.1.2.11 StartOffset Property

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

**Class**

**TDAScript**

**Syntax**

```pascal
property StartOffset: Int64;
```

**Remarks**

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

© 1997-2021 Devart. All Rights Reserved.

5.9.1.2.12 StartPos Property

Used to get the start position of the current statement in a script.

**Class**

**TDAScript**

**Syntax**

```pascal
property StartPos: Int64;
```

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

Contains a list of statements obtained from the SQL property.

**Class**

TDAScript

**Syntax**

```
property Statements: TDAStatements;
```

**Remarks**

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

For example, consider the following script:

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

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

**Example**

You can use the Statements property in the following way:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
var
  i: integer;
begin
  with Script do begin
    ```
for i := 0 to Statements.Count - 1 do
  if Copy(Statements[i].SQL, 1, 6) <> 'CREATE' then
    Statements[i].Execute;
  end;
end;

See Also

- TDAScript

© 1997-2021
Devart. All Rights Reserved.

5.9.1.1.3 Methods

Methods of the **TDAScript** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BreakExec</td>
<td>Stops script execution.</td>
</tr>
<tr>
<td>ErrorOffset</td>
<td>Used to get the offset of the statement if the Execute method raised an exception.</td>
</tr>
<tr>
<td>Execute</td>
<td>Executes a script.</td>
</tr>
<tr>
<td>ExecuteFile</td>
<td>Executes SQL statements contained in a file.</td>
</tr>
<tr>
<td>ExecuteNext</td>
<td>Executes the next statement in the script and then stops.</td>
</tr>
<tr>
<td>ExecuteStream</td>
<td>Executes SQL statements contained in a stream object.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
</tbody>
</table>

See Also

- TDAScript Class
- TDAScript Class Members

© 1997-2021
Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
5.9.1.3.1 BreakExec Method

Stops script execution.

**Class**

`TDAScript`

**Syntax**

```pascal
procedure BreakExec; virtual;
```

**Remarks**

Call the BreakExec method to stop script execution.

© 1997-2021 Devart. All Rights Reserved.

5.9.1.3.2 ErrorOffset Method

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

**Class**

`TDAScript`

**Syntax**

```pascal
function ErrorOffset: Int64;
```

**Return Value**

Offset of an error.

**Remarks**

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

**See Also**

- `OnError`

© 1997-2021 Devart. All Rights Reserved.
5.9.1.3.3 Execute Method

Executes a script.

Class

TDAScript

Syntax

```
procedure Execute; virtual;
```

Remarks

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

See Also

- ExecuteNext
- OnError
- ErrorOffset

5.9.1.3.4 ExecuteFile Method

Executes SQL statements contained in a file.

Class

TDAScript

Syntax

```
procedure ExecuteFile(const FileName: string);
```

Parameters

`FileName`

Holds the file name.

Remarks

Call the ExecuteFile method to execute SQL statements contained in a file. Script doesn't load full content into memory. Reading and execution is performed by blocks of 64k size.
Therefore, it is optimal to use it for big files.

5.9.1.1.3.5 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 InterBase raises an error, the OnError event occurs.

See Also

- Execute
- OnError
- ErrorOffset

5.9.1.1.3.6 ExecuteStream Method

Executes SQL statements contained in a stream object.

Class

TDAScript

Syntax

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

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

Remarks

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

5.9.1.1.3.7 FindMacro Method

Finds a macro with the specified name.

Class
TDAScript

Syntax

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

Parameters

Name
Holds the name of a macro to search for.

Return Value
TMacro object if a match is found, nil otherwise.

Remarks

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

See Also

- TMacro
- Macros
- MacroByName
5.9.1.3.8 MacroByName Method

Finds a macro with the specified name.

Class
TDAScript

Syntax

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

Parameters

Name
Holds the name of a macro to search for.

Return Value

TMacro object if a match is found.

Remarks

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

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

To set a value to a macro, use the TMacro.Value property.

See Also

- TMacro
- Macros
- FindMacro

© 1997-2021 Devart. All Rights Reserved.

5.9.1.4 Events

Events of the TDAScript class.

For a complete list of the TDAScript class members, see the TDAScript Members topic.
5.9.1.1.4.1 AfterExecute Event

Occurs after a SQL script execution.

Class
TDAScript

Syntax

```
property AfterExecute: TAfterStatementExecuteEvent;
```

Remarks

Occurs after a SQL script has been executed.

See Also

- Execute

© 1997-2021
Devart. All Rights Reserved.
5.9.1.1.4.2 BeforeExecute Event

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

Class
TDAScript

Syntax

```property
BeforeExecute: TBeforeStatementExecuteEvent;
```

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

5.9.1.1.4.3 OnError Event

Occurs when InterBase raises an error.

Class
TDAScript

Syntax

```property
OnError: TOnErrorEvent;
```

Remarks
Occurs when InterBase raises an error.

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

See Also
- ErrorOffset

© 1997-2021 Devart. All Rights Reserved.
5.9.1.2 TDAStatement Class

This class has attributes and methods for controlling single SQL statement of a script.

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

Unit

DAScript

Syntax

TDAStatement = class(TCollectionItem);

Remarks

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

See Also
- TDAScript
- TDAStatements

© 1997-2021 Devart. All Rights Reserved.

5.9.1.2.1 Members

TDAStatement class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndLine</td>
<td>Used to determine the number of the last statement line in a script.</td>
</tr>
<tr>
<td>EndOffset</td>
<td>Used to get the offset in the last line of the statement.</td>
</tr>
<tr>
<td>EndPos</td>
<td>Used to get the end position of the statement in a script.</td>
</tr>
<tr>
<td>Omit</td>
<td>Used to avoid execution of a statement.</td>
</tr>
<tr>
<td>Params</td>
<td>Contains parameters for an</td>
</tr>
</tbody>
</table>
Reference 254

### Properties

**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.

**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

### Methods

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

© 1997-2021 Devart. All Rights Reserved.

[Request Support] [DAC Forum] [Provide Feedback]

5.9.1.2.2 Properties

Properties of the **TDASStatement** class.

For a complete list of the **TDASStatement** class members, see the [TDASStatement Members] topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndLine</td>
<td>Used to determine the number of the last statement line in a script.</td>
</tr>
<tr>
<td>EndOffset</td>
<td>Used to get the offset in the last line of the statement.</td>
</tr>
<tr>
<td>EndPos</td>
<td>Used to get the end position of the statement in a script.</td>
</tr>
<tr>
<td>Omit</td>
<td>Used to avoid execution of a statement.</td>
</tr>
<tr>
<td>Params</td>
<td>Contains parameters for an</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Script</td>
<td>Used to determine the TDAScript object the SQL Statement belongs to.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to get or set the text of an SQL statement.</td>
</tr>
<tr>
<td>StartLine</td>
<td>Used to determine the number of the first statement line in a script.</td>
</tr>
<tr>
<td>StartOffset</td>
<td>Used to get the offset in the first line of a statement.</td>
</tr>
<tr>
<td>StartPos</td>
<td>Used to get the start position of the statement in a script.</td>
</tr>
</tbody>
</table>

See Also
- TDAScript Class
- TDAScript Class Members

### 5.9.1.2.2.1 EndLine Property

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

**Class**

**TDAScript**

**Syntax**

```
property EndLine: integer;
```

**Remarks**

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

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

Class
TDASatement

Syntax

property EndOffset: integer;

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

© 1997-2021
Devart. All Rights Reserved.

5.9.1.2.2.3 EndPos Property

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

Class
TDASatement

Syntax

property EndPos: integer;

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

© 1997-2021
Devart. All Rights Reserved.

5.9.1.2.2.4 Omit Property

Used to avoid execution of a statement.

Class
TDASatement
Syntax

```plaintext
property Omit: boolean;
```

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

Class
`TDAScript`  

Syntax

```plaintext
property Params: TDAParams;
```

Remarks
Contains parameters for an SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically. Params is a zero-based array of parameter records. Index specifies the array element to access.

See Also
- `TDAParam`

Class
`TDAScript`
Property Script:

Syntax

```
property script: TDAScript;
```

Remarks

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

5.9.1.2.2.7 SQL Property

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

Class

TDAScript

Syntax

```
property SQL: string;
```

Remarks

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

5.9.1.2.2.8 StartLine Property

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

Class

TDAScript

Syntax

```
property StartLine: integer;
```

Remarks

Use the StartLine property to determine the number of the first statement line in a script.
5.9.1.2.9 StartOffset Property

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

Class

TDAStatement

Syntax

property StartOffset: integer;

Remarks

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

5.9.1.2.10 StartPos Property

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

Class

TDAStatement

Syntax

property StartPos: integer;

Remarks

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

5.9.1.3 Methods

Methods of the TDAStatement class.

For a complete list of the TDAStatement class members, see the TDAStatement Members
5.9.1.2.3.1 Execute Method

Executes a statement.

Class
TDAStructure

Syntax

```plaintext
procedure Execute;
```

Remarks
Use the Execute method to execute a statement.

5.9.1.3 TDAStructures Class

Holds a collection of TDAStructure objects.

For a list of all members of this type, see TDAStructures members.
TDASTatements = class(TCollection);

Remarks

Each TDASTatements holds a collection of TDASTatement objects. TDASTatements maintains an index of the statements in its Items array. The Count property contains the number of statements in the collection. Use TDASTatements class to manipulate script SQL statements.

See Also

- TDAScript
- TDASTatement

© 1997-2021 Devart. All Rights Reserved.

5.9.1.3.1 Members

TDASTatements class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to access separate script statements.</td>
</tr>
</tbody>
</table>

5.9.1.3.2 Properties

Properties of the TDASTatements class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to access separate script statements.</td>
</tr>
</tbody>
</table>

See Also
5.9.1.3.2.1 Items Property (Indexer)

Used to access separate script statements.

Class

TDAStatements

Syntax

property Items[Index: Integer]: TDAStatement; default;

Parameters

Index
Holdstheindexvalue.

Remarks

Use the Items property to access individual script statements. The value of the Index parameter corresponds to the Index property of TDAStatement.

See Also

- TDAStatement

5.9.2 Types

Types in the DAScript unit.

Types

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

© 2021 Devart
### 5.9.2.1 TAfterStatementExecuteEvent Procedure Reference

This type is used for the **TDAScript.AfterExecute** event.

**Unit**

**DAScript**

**Syntax**

```plaintext
TAfterStatementExecuteEvent = procedure (Sender: TObject; SQL: string) of object;
```

**Parameters**

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

- **SQL**
  - Holds the passed SQL statement.

### 5.9.2.2 TBeforeStatementExecuteEvent Procedure Reference

This type is used for the **TDAScript.BeforeExecute** event.

**Unit**

**DAScript**

**Syntax**

```plaintext
TBeforeStatementExecuteEvent = procedure (Sender: TObject; var SQL: string; var Omit: boolean) of object;
```

**Parameters**

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

**SQL**
Holds the passed SQL statement.

**Omit**
True, if the statement execution should be skipped.

### 5.9.3 enumerations

Enumerations in the **DAScript** unit.

#### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TErrorAction</strong></td>
<td>Indicates the action to take when the OnError handler exits</td>
</tr>
</tbody>
</table>
5.9.3.1 **TErrorAction Enumeration**

Indicates the action to take when the OnError handler exits.

**Unit**

**DAScript**

**Syntax**

\[
\text{TErrorAction} = (\text{eaAbort, eaFail, eaException, eaContinue});
\]

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>eaAbort</td>
<td>Abort execution without displaying an error message.</td>
</tr>
<tr>
<td>eaContinue</td>
<td>Continue execution.</td>
</tr>
<tr>
<td>eaException</td>
<td>In Delphi 6 and higher exception is handled by the Application.HandleException method.</td>
</tr>
<tr>
<td>eaFail</td>
<td>Abort execution and display an error message.</td>
</tr>
</tbody>
</table>

5.10 **DASQLMonitor**

This unit contains the base class for the TIBCSQLMonitor component.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomDASQLMonitor</td>
<td>A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.</td>
</tr>
<tr>
<td>TDBMonitorOptions</td>
<td>This class holds options for dbMonitor.</td>
</tr>
</tbody>
</table>
Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlags</td>
<td>Represents the set of TDATraceFlag.</td>
</tr>
<tr>
<td>TMonitorOptions</td>
<td>Represents the set of TMonitorOption.</td>
</tr>
<tr>
<td>TOnSQLEvent</td>
<td>This type is used for the TCustomDASQLMonitor.OnSQL event.</td>
</tr>
</tbody>
</table>

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlag</td>
<td>Use TraceFlags to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
<tr>
<td>TMonitorOption</td>
<td>Used to define where information from SQLMonitor will be displayed.</td>
</tr>
</tbody>
</table>

5.10.1 Classes

Classes in the DASQLMonitor unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomDASQLMonitor</td>
<td>A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.</td>
</tr>
<tr>
<td>TDBMonitorOptions</td>
<td>This class holds options for dbMonitor.</td>
</tr>
</tbody>
</table>
5.10.1.1 TCustomDASQLMonitor Class

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

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

Unit

DASQLMonitor

Syntax

TCustomDASQLMonitor = class(TComponent);

Remarks

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

In applications use descendants of TCustomDASQLMonitor.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to activate monitoring of SQL.</td>
</tr>
<tr>
<td>DBMonitorOptions</td>
<td>Used to set options for dbMonitor.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to include the desired properties for TCustomDASQLMonitor.</td>
</tr>
<tr>
<td>TraceFlags</td>
<td>Used to specify which</td>
</tr>
</tbody>
</table>
database operations the monitor should track in an application at runtime.

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSQL</td>
<td>Occurs when tracing of SQL activity on database components is needed.</td>
</tr>
</tbody>
</table>

Properties of the `TCustomDASQLMonitor` class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to activate monitoring of SQL.</td>
</tr>
<tr>
<td>DBMonitorOptions</td>
<td>Used to set options for dbMonitor.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to include the desired properties for <code>TCustomDASQLMonitor</code>.</td>
</tr>
<tr>
<td>TraceFlags</td>
<td>Used to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
</tbody>
</table>

See Also

- `TCustomDASQLMonitor Class`
- `TCustomDASQLMonitor Class Members`
5.10.1.1.2.1 Active Property

Used to activate monitoring of SQL.

Class

TCustomDASQLMonitor

Syntax

property Active: boolean default True;

Remarks

Set the Active property to True to activate monitoring of SQL.

See Also

• OnSQL

5.10.1.1.2.2 DBMonitorOptions Property

Used to set options for dbMonitor.

Class

TCustomDASQLMonitor

Syntax

property DBMonitorOptions: TDBMonitorOptions;

Remarks

Use DBMonitorOptions to set options for dbMonitor.

5.10.1.1.2.3 Options Property

Used to include the desired properties for TCustomDASQLMonitor.

Class
**TCustomDASQLMonitor**

**Syntax**

```pascal
property Options: TMonitorOptions default [moDialog, moSQLMonitor, moDBMonitor, moCustom];
```

**Remarks**

Set Options to include the desired properties for TCustomDASQLMonitor.

**See Also**

- [OnSQL](#)

© 1997-2021 Devart. All Rights Reserved.

---

**5.10.1.2.4 TraceFlags Property**

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

**Class**

**TCustomDASQLMonitor**

**Syntax**

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

**Remarks**

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

**See Also**

- [OnSQL](#)

© 1997-2021 Devart. All Rights Reserved.

---
5.10.1.1.3 Events

Events of the `TCustomDASQLMonitor` class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSQL</td>
<td>Occurs when tracing of SQL activity on database components is needed.</td>
</tr>
</tbody>
</table>

See Also

- `TCustomDASQLMonitor Class`
- `TCustomDASQLMonitor Class Members`

© 1997-2021 Devart. All Rights Reserved.

5.10.1.1.3.1 OnSQL Event

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

Class

`TCustomDASQLMonitor`

Syntax

```
property OnSQL: TOnSQLEvent;
```

Remarks

Write the OnSQL event handler to let an application trace SQL activity on database components. The Text parameter holds the detected SQL statement. Use the Flag parameter to make selective processing of SQL in the handler body.

See Also

- `TraceFlags`

© 1997-2021 Devart. All Rights Reserved.
5.10.1.2 TDBMonitorOptions Class

This class holds options for dbMonitor.

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

Unit
DASQLMonitor

Syntax

TDBMonitorOptions = class(TPersistent);

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Used to set the host name or IP address of the computer where dbMonitor application runs.</td>
</tr>
<tr>
<td>Port</td>
<td>Used to set the port number for connecting to dbMonitor.</td>
</tr>
<tr>
<td>ReconnectTimeout</td>
<td>Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.</td>
</tr>
<tr>
<td>SendTimeout</td>
<td>Used to set timeout for sending events to dbMonitor.</td>
</tr>
</tbody>
</table>

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

Properties of the TDBMonitorOptions class.

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

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Used to set the host name or IP address of the computer where dbMonitor application runs.</td>
</tr>
<tr>
<td>Port</td>
<td>Used to set the port number for connecting to dbMonitor.</td>
</tr>
<tr>
<td>ReconnectTimeout</td>
<td>Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.</td>
</tr>
<tr>
<td>SendTimeout</td>
<td>Used to set timeout for sending events to dbMonitor.</td>
</tr>
</tbody>
</table>

See Also
- TDBMonitorOptions Class
- TDBMonitorOptions Class Members

© 1997-2021 Devart. All Rights Reserved.
Use the Host property to set the host name or IP address of the computer where dbMonitor application runs.

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

5.10.1.2.2.2 Port Property

Used to set the port number for connecting to dbMonitor.

Class

TDBMonitorOptions

Syntax

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

Remarks

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

5.10.1.2.2.3 ReconnectTimeout Property

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

Class

TDBMonitorOptions

Syntax

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

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

Class

**TDBMonitorOptions**

Syntax

```
property SendTimeout: integer default DefaultSendTimeout;
```

Remarks

Use the SendTimeout property to set timeout (in milliseconds) for sending events to dbMonitor. If dbMonitor does not respond in the specified timeout, event is ignored.

---

**5.10.2 Types**

Types in the **DASQLMonitor** unit.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TDA Trace Flags</strong></td>
<td>Represents the set of TDA TraceFlag.</td>
</tr>
<tr>
<td><strong>T Monitor Options</strong></td>
<td>Represents the set of T Monitor Option.</td>
</tr>
<tr>
<td><strong>T On SQL Event</strong></td>
<td>This type is used for the T Custom DASQL Monitor On SQL event.</td>
</tr>
</tbody>
</table>

---

© 1997-2021

Devart. All Rights Reserved.
Devart. All Rights Reserved.

5.10.2.1 TDATraceFlags Set

Represents the set of `TDATraceFlag`.

Unit

`DASQLMonitor`

Syntax

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

5.10.2.2 TMonitorOptions Set

Represents the set of `TMonitorOption`.

Unit

`DASQLMonitor`

Syntax

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

5.10.2.3 TOnSQLEvent Procedure Reference

This type is used for the `TCustomDASQLMonitor.OnSQL` event.

Unit

`DASQLMonitor`

Syntax

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

Parameters
Sender
An object that raised the event.

Text
Holds the detected SQL statement.

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

5.10.3 Enumerations

Enumerations in the DASQLMonitor unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlag</td>
<td>Use TraceFlags to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
<tr>
<td>TMonitorOption</td>
<td>Used to define where information from SQLMonitor will be displayed.</td>
</tr>
</tbody>
</table>

5.10.3.1 TDATraceFlag Enumeration

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

Unit
DASQLMonitor

Syntax

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tfBlob</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfConnect</td>
<td>Establishing a connection.</td>
</tr>
<tr>
<td>tfError</td>
<td>Errors of query execution.</td>
</tr>
<tr>
<td>tfMisc</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfObjDestroy</td>
<td>Destroying of components.</td>
</tr>
<tr>
<td>tfParams</td>
<td>Representing parameter values for tfQPrepare and tfQExecute.</td>
</tr>
<tr>
<td>tfPool</td>
<td>Connection pool operations.</td>
</tr>
<tr>
<td>tfQExecute</td>
<td>Execution of the queries.</td>
</tr>
<tr>
<td>tfQFetch</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfQPrepare</td>
<td>Queries preparation.</td>
</tr>
<tr>
<td>tfService</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfStmt</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfTransact</td>
<td>Processing transactions.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.10.3.2 TMonitorOption Enumeration

Used to define where information from SQLMonitor will be displayed.

Unit

DASQLMonitor

Syntax

TMonitorOption = (moDialog, moSQLMonitor, moDBMonitor, moCustom, moHandled);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>moCustom</td>
<td>Monitoring of SQL for individual components is allowed. Set Debug properties in SQL-related components to True to let TCustomDASQLMonitor instance to monitor their behavior. Has effect when moDialog is included.</td>
</tr>
<tr>
<td>moDBMonitor</td>
<td>Debug information is displayed in DBMonitor.</td>
</tr>
</tbody>
</table>
### 5.11 DBAccess

This unit contains base classes for most of the components.

#### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDAError</td>
<td>A base class for exceptions that are raised when an error occurs on the server side.</td>
</tr>
<tr>
<td>TCRDataSource</td>
<td>Provides an interface between a DAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TCustomConnectDialog</td>
<td>A base class for the connect dialog components.</td>
</tr>
<tr>
<td>TCustomDAConnection</td>
<td>A base class for components used to establish connections.</td>
</tr>
<tr>
<td>TCustomDADataset</td>
<td>Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.</td>
</tr>
<tr>
<td>TCustomDASQL</td>
<td>A base class for components executing SQL statements that do not return result sets.</td>
</tr>
<tr>
<td>TCustomDAUpdateSQL</td>
<td>A base class for components that provide DML statements for more flexible control over data modifications.</td>
</tr>
<tr>
<td>TDACCondition</td>
<td>Represents a condition from the TDACConditions list.</td>
</tr>
<tr>
<td>TDACConditions</td>
<td>Holds a collection of</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TDACCondition</td>
<td>TDACCondition objects.</td>
</tr>
<tr>
<td>TDAConnectionOptions</td>
<td>This class allows setting up the behaviour of the TDAConnection class.</td>
</tr>
<tr>
<td>TDAConnectionSSLOptions</td>
<td>This class is used to set up the SSL options.</td>
</tr>
<tr>
<td>TDADatasetOptions</td>
<td>This class allows setting up the behaviour of the TDADataset class.</td>
</tr>
<tr>
<td>TDAEncryption</td>
<td>Used to specify the options of the data encryption in a dataset.</td>
</tr>
<tr>
<td>TDAMapRule</td>
<td>Class that forms rules for Data Type Mapping.</td>
</tr>
<tr>
<td>TDAMapRules</td>
<td>Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.</td>
</tr>
<tr>
<td>TDAMetaData</td>
<td>A class for retrieving metainformation of the specified database objects in the form of dataset.</td>
</tr>
<tr>
<td>TDAParam</td>
<td>A class that forms objects to represent the values of the parameters set.</td>
</tr>
<tr>
<td>TDAParams</td>
<td>This class is used to manage a list of TDAParam objects for an object that uses field parameters.</td>
</tr>
<tr>
<td>TDATransaction</td>
<td>A base class that implements functionality for controlling transactions.</td>
</tr>
<tr>
<td>TMacro</td>
<td>Object that represents the value of a macro.</td>
</tr>
<tr>
<td>TMacros</td>
<td>Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataset components.</td>
</tr>
<tr>
<td>TPoolingOptions</td>
<td>This class allows setting up the behaviour of the connection pool.</td>
</tr>
<tr>
<td>TSmartFetchOptions</td>
<td>Smart fetch options are</td>
</tr>
</tbody>
</table>
used to set up the behavior of the SmartFetch mode.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAfterExecuteEvent</td>
<td>This type is used for the TCustomDADDataSet.AfterExecute and TCustomDASQL.AfterExecute events.</td>
</tr>
<tr>
<td>TAfterFetchEvent</td>
<td>This type is used for the TCustomDADDataSet.AfterFetch event.</td>
</tr>
<tr>
<td>TBeforeFetchEvent</td>
<td>This type is used for the TCustomDADDataSet.BeforeFetch event.</td>
</tr>
<tr>
<td>TConnectionLostEvent</td>
<td>This type is used for the TCustomDACConnection.OnConnectionLost event.</td>
</tr>
<tr>
<td>TDAConnectionErrorEvent</td>
<td>This type is used for the TCustomDACConnection.OnError event.</td>
</tr>
<tr>
<td>TDATransactionErrorEvent</td>
<td>This type is used for the TDATransactionOnError event.</td>
</tr>
<tr>
<td>TRefreshOptions</td>
<td>Represents the set of TRefreshOption.</td>
</tr>
<tr>
<td>TUpdateExecuteEvent</td>
<td>This type is used for the TCustomDADDataSet.AfterUpdateExecute and TCustomDADDataSet.BeforeUpdateExecute events.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLabelSet</td>
<td>Sets the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>TLockMode</td>
<td>Specifies the lock mode.</td>
</tr>
<tr>
<td>TRefreshOption</td>
<td>Indicates when the editing record will be refreshed.</td>
</tr>
</tbody>
</table>
TRetryMode

Specifies the application behavior when connection is lost.

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>When set to True allows data access components to change screen cursor for the execution time.</td>
</tr>
</tbody>
</table>

5.11.1 Classes

Classes in the DBAccess unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDAError</td>
<td>A base class for exceptions that are raised when an error occurs on the server side.</td>
</tr>
<tr>
<td>TCRDataSource</td>
<td>Provides an interface between a DAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TCustomConnectDialog</td>
<td>A base class for the connect dialog components.</td>
</tr>
<tr>
<td>TCustomDAConnection</td>
<td>A base class for components used to establish connections.</td>
</tr>
<tr>
<td>TCustomDADataset</td>
<td>Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.</td>
</tr>
<tr>
<td>TCustomDASQL</td>
<td>A base class for components executing SQL statements that do not return result sets.</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>TCustomDAUpdateSQL</td>
<td>A base class for components that provide DML statements for more flexible control over data modifications.</td>
</tr>
<tr>
<td>TDACCondition</td>
<td>Represents a condition from the TDACConditions list.</td>
</tr>
<tr>
<td>TDACConditions</td>
<td>Holds a collection of TDACCondition objects.</td>
</tr>
<tr>
<td>TDACConnectionOptions</td>
<td>This class allows setting up the behaviour of the TDACConnection class.</td>
</tr>
<tr>
<td>TDACConnectionSSLOptions</td>
<td>This class is used to set up the SSL options.</td>
</tr>
<tr>
<td>TDADatasetOptions</td>
<td>This class allows setting up the behaviour of the TDADataset class.</td>
</tr>
<tr>
<td>TDAEncryption</td>
<td>Used to specify the options of the data encryption in a dataset.</td>
</tr>
<tr>
<td>TDAMapRule</td>
<td>Class that forms rules for Data Type Mapping.</td>
</tr>
<tr>
<td>TDAMapRules</td>
<td>Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.</td>
</tr>
<tr>
<td>TDAMetaData</td>
<td>A class for retrieving metainformation of the specified database objects in the form of dataset.</td>
</tr>
<tr>
<td>TDAParam</td>
<td>A class that forms objects to represent the values of the parameters set.</td>
</tr>
<tr>
<td>TDAPrams</td>
<td>This class is used to manage a list of TDAParam objects for an object that uses field parameters.</td>
</tr>
<tr>
<td>TDATransaction</td>
<td>A base class that implements functionality for controlling transactions.</td>
</tr>
<tr>
<td>TMacro</td>
<td>Object that represents the value of a macro.</td>
</tr>
<tr>
<td>TMacros</td>
<td>Controls a list of TMacro</td>
</tr>
<tr>
<td><strong>TPoolingOptions</strong></td>
<td>This class allows setting up the behaviour of the connection pool.</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TSmartFetchOptions</strong></td>
<td>Smart fetch options are used to set up the behavior of the SmartFetch mode.</td>
</tr>
</tbody>
</table>

5.11.1.1 **EDAError Class**

A base class for exceptions that are raised when an error occurs on the server side.

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

**Unit**

```
DBAccess
```

**Syntax**

```
EDAError = class(EDatabaseError);
```

**Remarks**

EDAError is a base class for exceptions that are raised when an error occurs on the server side.

```
Component | Description |
----------|-------------|
Contains the component that
```
Properties of the EDAError class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Contains the component that caused the error.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>Determines the error code returned by the server.</td>
</tr>
</tbody>
</table>

See Also

- EDAError Class
- EDAError Class Members

Contains the component that caused the error.

Class

EDAError

Syntax

```
property Component: TObject;
```

Remarks

The Component property contains the component that caused the error.
5.11.1.2.2 ErrorCode Property

Determines the error code returned by the server.

Class
EDAError

Syntax

property ErrorCode: integer;

Remarks

Use the ErrorCode property to determine the error code returned by InterBase. This value is always positive.

See Also
- EIBCError.ErrorNumber

5.11.1.2 TCRDataSource Class

Provides an interface between a DAC dataset components and data-aware controls on a form.

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

Unit
DBAccess

Syntax

TCRDataSource = class(TDataSource);

Remarks

TCRDataSource provides an interface between a DAC dataset components and data-aware controls on a form.
TCRDataSource inherits its functionality directly from the TDataSource component.

At design time assign individual data-aware components' DataSource properties from their drop-down listboxes.

5.11.1.3  

**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**

```plaintext
TCustomConnectDialog = class(TComponent);
```

**Remarks**

TCustomConnectDialog is a base class for the connect dialog components. It provides functionality to show a dialog box where user can edit username, password and server name before connecting to a database. You can customize captions of buttons and labels by their properties.

5.11.1.3.1  Members

**TCustomConnectDialog** class overview.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelButton</td>
<td>Used to specify the label for the Cancel button.</td>
</tr>
<tr>
<td>Caption</td>
<td>Used to set the caption of dialog box.</td>
</tr>
<tr>
<td>ConnectButton</td>
<td>Used to specify the label for the Connect button.</td>
</tr>
<tr>
<td>DialogClass</td>
<td>Used to specify the class of the form that will be displayed to enter login information.</td>
</tr>
<tr>
<td>LabelSet</td>
<td>Used to set the language of buttons and labels captions.</td>
</tr>
<tr>
<td>PasswordLabel</td>
<td>Used to specify a prompt for password edit.</td>
</tr>
<tr>
<td>Retries</td>
<td>Used to indicate the number of retries of failed connections.</td>
</tr>
<tr>
<td>SavePassword</td>
<td>Used for the password to be displayed in ConnectDialog in asterisks.</td>
</tr>
<tr>
<td>ServerLabel</td>
<td>Used to specify a prompt for the server name edit.</td>
</tr>
<tr>
<td>StoreLogInfo</td>
<td>Used to specify whether the login information should be kept in system registry after a connection was established.</td>
</tr>
<tr>
<td>UsernameLabel</td>
<td>Used to specify a prompt for username edit.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.</td>
</tr>
<tr>
<td>GetServerList</td>
<td>Retrieves a list of available server names.</td>
</tr>
</tbody>
</table>
Properties of the **TCustomConnectDialog** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelButton</td>
<td>Used to specify the label for the Cancel button.</td>
</tr>
<tr>
<td>Caption</td>
<td>Used to set the caption of dialog box.</td>
</tr>
<tr>
<td>ConnectButton</td>
<td>Used to specify the label for the Connect button.</td>
</tr>
<tr>
<td>DialogClass</td>
<td>Used to specify the class of the form that will be displayed to enter login information.</td>
</tr>
<tr>
<td>LabelSet</td>
<td>Used to set the language of buttons and labels captions.</td>
</tr>
<tr>
<td>PasswordLabel</td>
<td>Used to specify a prompt for password edit.</td>
</tr>
<tr>
<td>Retries</td>
<td>Used to indicate the number of retries of failed connections.</td>
</tr>
<tr>
<td>SavePassword</td>
<td>Used for the password to be displayed in ConnectDialog in asterisks.</td>
</tr>
<tr>
<td>ServerLabel</td>
<td>Used to specify a prompt for the server name edit.</td>
</tr>
<tr>
<td>StoreLogInfo</td>
<td>Used to specify whether the login information should be kept in system registry after a connection was established.</td>
</tr>
<tr>
<td>UsernameLabel</td>
<td>Used to specify a prompt for username edit.</td>
</tr>
</tbody>
</table>

### See Also
- **TCustomConnectDialog Class**
- **TCustomConnectDialog Class Members**
5.11.1.3.2.1 CancelButton Property

Used to specify the label for the Cancel button.

Class

TCustomConnectDialog

Syntax

```property
CancelButton: string;
```

Remarks

Use the CancelButton property to specify the label for the Cancel button.

5.11.1.3.2.2 Caption Property

Used to set the caption of dialog box.

Class

TCustomConnectDialog

Syntax

```property
Caption: string;
```

Remarks

Use the Caption property to set the caption of dialog box.

5.11.1.3.2.3 ConnectButton Property

Used to specify the label for the Connect button.

Class
TCustomConnectDialog

Syntax

```pascal
property ConnectButton: string;
```

Remarks

Use the ConnectButton property to specify the label for the Connect button.

Class

TCustomConnectDialog

Syntax

```pascal
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 IBDAC installation directory.

See Also

- GetServerList
5.11.1.3.2.5 LabelSet Property

Used to set the language of buttons and labels captions.

Class

TCustomConnectDialog

Syntax

property LabelSet: TLabelSet default lsEnglish;

Remarks

Use the LabelSet property to set the language of labels and buttons captions.

The default value is lsEnglish.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.3.2.6 PasswordLabel Property

Used to specify a prompt for password edit.

Class

TCustomConnectDialog

Syntax

property PasswordLabel: string;

Remarks

Use the PasswordLabel property to specify a prompt for password edit.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.3.2.7 Retries Property

Used to indicate the number of retries of failed connections.

Class

TCustomConnectDialog
### Syntax

```property```
Retries: word default 3;
```property```

### Remarks

Use the Retries property to determine the number of retries of failed connections.

© 1997-2021 Devart. All Rights Reserved.

#### 5.11.1.3.2.8 SavePassword Property

Used for the password to be displayed in ConnectDialog in asterisks.

#### Class

`TCustomConnectDialog`

#### Syntax

```property```
SavePassword: boolean default False;
```property```

#### Remarks

If True, and the Password property of the connection instance is assigned, the password in ConnectDialog is displayed in asterisks.

© 1997-2021 Devart. All Rights Reserved.

#### 5.11.1.3.2.9 ServerLabel Property

Used to specify a prompt for the server name edit.

#### Class

`TCustomConnectDialog`

#### Syntax

```property```
ServerLabel: string;
```property```

#### Remarks

Use the ServerLabel property to specify a prompt for the server name edit.
5.11.1.3.2.10  StoreLogInfo Property

Used to specify whether the login information should be kept in system registry after a connection was established.

Class

TCustomConnectDialog

Syntax

```plaintext
property StoreLogInfo: boolean default True;
```

Remarks

Use the StoreLogInfo property to specify whether to keep login information in system registry after a connection was established using provided username, password and servername. Set this property to True to store login information.

The default value is True.

5.11.1.3.2.11  UsernameLabel Property

Used to specify a prompt for username edit.

Class

TCustomConnectDialog

Syntax

```plaintext
property UsernameLabel: string;
```

Remarks

Use the UsernameLabel property to specify a prompt for username edit.
5.11.1.3.3 Methods

Methods of the TCustomConnectDialog class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button. Returns True if connected. If user clicks Cancel, Execute returns False.</td>
</tr>
<tr>
<td>GetServerList</td>
<td>Retrieves a list of available server names.</td>
</tr>
</tbody>
</table>

See Also
- TCustomConnectDialog Class
- TCustomConnectDialog Class Members

© 1997-2021 Devart. All Rights Reserved.
In the case of failed connection Execute offers to connect repeat \texttt{Retries} times.

\textbf{5.11.1.3.2 GetServerList Method}

Retrieves a list of available server names.

\textbf{Class}

\texttt{TCustomConnectDialog}

\textbf{Syntax}

\begin{verbatim}
procedure GetServerList(List: TStrings); virtual;
\end{verbatim}

\textbf{Parameters}

\textit{List}

Holds a list of available server names.

\textbf{Remarks}

Call the GetServerList method to retrieve a list of available server names. It is particularly relevant for writing custom login form.

\textbf{See Also}

\texttt{DialogClass}

\textbf{5.11.1.4 TCustomDAConnection Class}

A base class for components used to establish connections.

For a list of all members of this type, see \texttt{TCustomDAConnection} members.

\textbf{Unit}

\texttt{DBAccess}

\textbf{Syntax}

\begin{verbatim}
TCustomDAConnection = class(TCustomConnection);
\end{verbatim}
Remarks

TCustomDACConnection is a base class for components that establish connection with database, provide customised login support, and perform transaction control.

Do not create instances of TCustomDACConnection. To add a component that represents a connection to a source of data, use descendants of the TCustomDACConnection class.

5.11.1.4.1 Members

**TCustomDACConnection** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectDialog</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td>ConnectString</td>
<td>Used to specify the connection information, such as: UserName, Password, Server, etc.</td>
</tr>
<tr>
<td>ConvertEOL</td>
<td>Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td>InTransaction</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td>LoginPrompt</td>
<td>Specifies whether a login dialog appears immediately before opening a new connection.</td>
</tr>
<tr>
<td>Options</td>
<td>Specifies the connection behavior.</td>
</tr>
<tr>
<td>Password</td>
<td>Serves to supply a password for login.</td>
</tr>
<tr>
<td>Pooling</td>
<td>Enables or disables using connection pool.</td>
</tr>
<tr>
<td>PoolingOptions</td>
<td>Specifies the behaviour of connection pool.</td>
</tr>
<tr>
<td>Server</td>
<td>Serves to supply the server name for login.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Username</td>
<td>Used to supply a user name for login.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td>Commit</td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td>Connect</td>
<td>Establishes a connection to the server.</td>
</tr>
<tr>
<td>CreateSQL</td>
<td>Creates a component for queries execution.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Performs disconnect.</td>
</tr>
<tr>
<td>ExecProc</td>
<td>Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td>ExecProcEx</td>
<td>Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td>ExecSQL</td>
<td>Executes a SQL statement with parameters.</td>
</tr>
<tr>
<td>ExecSQLEx</td>
<td>Executes any SQL statement outside the TQuery or TSQL components.</td>
</tr>
<tr>
<td>GetDatabaseNames</td>
<td>Returns a database list from the server.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetStoredProcNames</td>
<td>Returns a list of stored procedures from the server.</td>
</tr>
<tr>
<td>GetTableNames</td>
<td>Provides a list of available tables names.</td>
</tr>
<tr>
<td>MonitorMessage</td>
<td>Sends a specified message through the TCustomDASQLMonitor component.</td>
</tr>
<tr>
<td>Ping</td>
<td>Used to check state of connection to the server.</td>
</tr>
<tr>
<td>RemoveFromPool</td>
<td>Marks the connection that should not be returned to the pool after disconnect.</td>
</tr>
</tbody>
</table>
### Rollback
Discards all current data changes and ends transaction.

### StartTransaction
Begins a new user transaction.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnectionLost</td>
<td>This event occurs when connection was lost.</td>
</tr>
<tr>
<td>OnError</td>
<td>This event occurs when an error has arisen in the connection.</td>
</tr>
</tbody>
</table>

5.11.1.4.2 Properties

Properties of the **TCustomDAConnection** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectDialog</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td>ConnectString</td>
<td>Used to specify the connection information, such as: UserName, Password, Server, etc.</td>
</tr>
<tr>
<td>ConvertEOL</td>
<td>Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td>InTransaction</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td>LoginPrompt</td>
<td>Specifies whether a login dialog appears immediately before opening a new transaction.</td>
</tr>
<tr>
<td><strong>Component</strong></td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the connection behavior.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Serves to supply a password for login.</td>
</tr>
<tr>
<td><strong>Pooling</strong></td>
<td>Enables or disables using connection pool.</td>
</tr>
<tr>
<td><strong>PoolingOptions</strong></td>
<td>Specifies the behaviour of connection pool.</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Serves to supply the server name for login.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Used to supply a user name for login.</td>
</tr>
</tbody>
</table>

**See Also**
- **TCustomDAConnection Class**
- **TCustomDAConnection Class Members**
5.11.1.4.2.2 ConnectString Property

Used to specify the connection information, such as: UserName, Password, Server, etc.

Class

TCustomDAConnection

Syntax

property ConnectString: string stored False;

Remarks

IBDAC recognizes an ODBC-like syntax in provider string property values. Within the string, elements are delimited by using a semicolon. Each element consists of a keyword, an equal sign character, and the value passed on initialization. For example:

Server=London1;User ID=nancyd

Connection parameters

The following connection parameters can be used to customize connection:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoginPrompt</td>
<td>Specifies whether a login dialog appears immediately before opening a new connection.</td>
</tr>
<tr>
<td>Pooling</td>
<td>Enables or disables using connection pool.</td>
</tr>
<tr>
<td>ConnectionLifeTime</td>
<td>Used to specify the maximum time during which an opened connection can be used by connection pool.</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>Used to specify the maximum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td>MinPoolSize</td>
<td>Used to specify the minimum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td>Validate Connection</td>
<td>Used for a connection to be validated when it is returned from the pool.</td>
</tr>
<tr>
<td>Server</td>
<td>Serves to supply the server name for login.</td>
</tr>
<tr>
<td>Username</td>
<td>Used to supply a user name for login.</td>
</tr>
<tr>
<td>Password</td>
<td>Used to supply a user name for login.</td>
</tr>
<tr>
<td>ClientLibrary</td>
<td>Used to set or get the client library location.</td>
</tr>
<tr>
<td>Database</td>
<td>Used to set the name of the database to</td>
</tr>
<tr>
<td><strong>Charset</strong></td>
<td>Used to set the character set that IBDAC uses to read and write character data.</td>
</tr>
<tr>
<td><strong>Protocol</strong></td>
<td>Used to specify the Network protocol of connection with InterBase server.</td>
</tr>
<tr>
<td><strong>Role</strong></td>
<td>Used to specify the InterBase connection role.</td>
</tr>
<tr>
<td><strong>UseUnicode</strong></td>
<td>Used to enable or disable Unicode support.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Used to specify the port number for the connection.</td>
</tr>
</tbody>
</table>

**See Also**
- Password
- Username
- Server
- Connect

**5.11.1.4.2.3 ConvertEOL Property**

Allows customizing line breaks in string fields and parameters.

**Class**

TCustomDAConnection

**Syntax**

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

**Remarks**

Affects the line break behavior in string fields and parameters. When fetching strings (including the text BLOB fields) with ConvertEOL = True, dataset converts their line breaks from the LF to CRLF form. And when posting strings to server with ConvertEOL turned on, their line breaks are converted from CRLF to LF form. By default, strings are not converted.
InTransaction Property

Indicates whether the transaction is active.

Class

TCustomDAConnection

Syntax

```pascal
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`

LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Class

TCustomDAConnection

Syntax

```pascal
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 lbdacVcl unit appears to the uses clause.

5.11.4.2.6 Options Property

Specifies the connection behavior.

Class

TCustomDAConnection

Syntax

```plaintext
property Options: TDAConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of the connection.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowImplicitConnect</td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
<tr>
<td>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields.</td>
</tr>
<tr>
<td></td>
<td>It is used when a sort type is not specified explicitly after the field</td>
</tr>
<tr>
<td></td>
<td>name in the TMemDataSet.IndexFieldNames property of a dataset.</td>
</tr>
<tr>
<td>DisconnectedMode</td>
<td>Used to open a connection only when needed for performing a server call and</td>
</tr>
<tr>
<td></td>
<td>closes after performing the operation.</td>
</tr>
<tr>
<td>KeepDesignConnected</td>
<td>Used to prevent an application from establishing a connection at the time of</td>
</tr>
<tr>
<td></td>
<td>startup.</td>
</tr>
<tr>
<td>LocalFailover</td>
<td>If True, the OnConnectionLost event occurs and a failover operation can be</td>
</tr>
<tr>
<td></td>
<td>performed after connection breaks.</td>
</tr>
</tbody>
</table>

See Also

- Disconnected Mode
5.11.1.4.2.7 Password Property

Serves to supply a password for login.

Class

TCustomDAConnection

Syntax

```plaintext
property Password: string stored False;
```

Remarks

Use the Password property to supply a password to handle server's request for a login.

**Warning:** Storing hard-coded user name and password entries as property values or in code for the OnLogin event handler can compromise server security.

See Also

- Username
- Server

5.11.1.4.2.8 Pooling Property

Enables or disables using connection pool.

Class

TCustomDAConnection

Syntax

```plaintext
property Pooling: boolean default DefValPooling;
```

Remarks

Normally, when TCustomDAConnection establishes connection with the server it takes
server memory and time resources for allocating new server connection. For example, pooling can be very useful when using disconnect mode. If an application has wide user activity that forces many connect/disconnect operations, it may spend a lot of time on creating connection and sending requests to the server. TCustomDAConnection has software pool which stores open connections with identical parameters.

Connection pool uses separate thread that validates the pool every 30 seconds. Pool validation consists of checking each connection in the pool. If a connection is broken due to a network problem or another reason, it is deleted from the pool. The validation procedure removes also connections that are not used for a long time even if they are valid from the pool.

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, TIBCConnection.Database, TIBCConnectionOptions.Charset, TIBCConnectionOptions.UseUnicode, TIBCConnectionOptions.Role, TIBCConnection.SQLDialect, TIBCConnection.Params.

Note: Using Pooling := True can cause errors with working with temporary tables.

See Also
- Username
- Password
- PoolingOptions
- Connection Pooling

© 1997-2021
Devart. All Rights Reserved.
Set the properties of PoolingOptions to specify the behaviour of connection pool.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionLifetime</td>
<td>Used to specify the maximum time during which an opened connection can be used by connection pool.</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>Used to specify the maximum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td>MinPoolSize</td>
<td>Used to specify the minimum number of connections that can be opened in the connection pool.</td>
</tr>
<tr>
<td>Validate</td>
<td>Used for a connection to be validated when it is returned from the pool.</td>
</tr>
</tbody>
</table>

See Also
- *Pooling*

© 1997-2021 Devart. All Rights Reserved.

5.11.1.4.2.10  Server Property

Serves to supply the server name for login.

Class

`TCustomDAConnection`

Syntax

```
property server: string;
```

Remarks

Use the Server property to supply server name to handle server's request for a login.

See Also
- *Username*
- *Password*
5.11.1.4.2.11 Username Property

Used to supply a user name for login.

Class

TCustomDAConnection

Syntax

```property Username: string;```

Remarks

Use the Username property to supply a user name to handle server's request for login. If this property is not set, IBDAC 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**

Methods of the **TCustomDAConnection** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td><strong>Commit</strong></td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td><strong>Connect</strong></td>
<td>Establishes a connection to</td>
</tr>
</tbody>
</table>
CreateSQL | Creates a component for queries execution.
---|---
Disconnect | Performs disconnect.
ExecProc | Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx | Allows to execute a stored procedure or function.
ExecSQL | Executes a SQL statement with parameters.
ExecSQLEx | Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames | Returns a database list from the server.
GetKeyFieldNames | Provides a list of available key field names.
GetStoredProcNames | Returns a list of stored procedures from the server.
GetTableNames | Provides a list of available tables names.
MonitorMessage | Sends a specified message through the TCustomDASQLMonitor component.
Ping | Used to check state of connection to the server.
RemoveFromPool | Marks the connection that should not be returned to the pool after disconnect.
Rollback | Discards all current data changes and ends transaction.
StartTransaction | Begins a new user transaction.

See Also
- TCustomDAConnection Class
- TCustomDAConnection Class Members
5.11.1.4.3.1 ApplyUpdates Method

Applies changes in datasets.

Class

TCustomDAConnection

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyUpdates</td>
<td>Applies changes from all active datasets.</td>
</tr>
<tr>
<td>ApplyUpdates(const DataSets: array of TCustomDADataSet)</td>
<td>Applies changes from the specified datasets.</td>
</tr>
</tbody>
</table>

Applies changes from all active datasets.

Class

TCustomDAConnection

Syntax

```plaintext
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
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.

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
- TCustomIBCDataSet.FetchAll

5.11.1.4.3.3 Connect Method

Establishes a connection to the server.

Class
TCustomDAConnection

Syntax

```
procedure Connect; overload:
procedure Connect(const ConnectString: string); overload;
```

Remarks
Call the Connect method to establish a connection to the server. Connect sets the Connected property to True. If LoginPrompt is True, Connect prompts user for login information as required by the server, or otherwise tries to establish a connection using values provided in the Username, Password, and Server properties.

See Also
- Disconnect
- Username
- Password
- Server
- ConnectDialog
5.11.1.4.3.4 CreateSQL Method

Creates a component for queries execution.

Class

TCustomDAConnection

Syntax

function CreateSQL: TCustomDASQL; virtual;

Return Value

A new instance of the class.

Remarks

Call the CreateSQL to return a new instance of the TCustomDASQL class and associates it with this connection object. In the descendant classes this method should be overridden to create an appropriate descendant of the TCustomDASQL component.

5.11.1.4.3.5 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 will lead to applying the action specified in TIBCTransaction.DefaultCloseAction for the current user transaction.

**Note:** If a previously active connection is closed and then reopened, any associated datasets must be individually reopened; reopening the connection does not automatically reopen associated datasets.

### See Also
- [Connect](#)

5.11.1.4.3.6 ExecProc Method

Allows to execute stored procedure or function providing its name and parameters.

#### Class

**TCustomDACConnection**

#### Syntax

```delphi
function ExecProc(const Name: string; const Params: array of variant): variant; virtual;
```

#### Parameters

- **Name**
  - Holds the name of the stored procedure or function.

- **Params**
  - Holds the parameters of the stored procedure or function.

#### Return Value

- the result of the stored procedure.

#### Remarks

Allows to execute stored procedure or function providing its name and parameters.

Use the following Name value syntax for executing specific overloaded routine:
"StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.

Assign parameters' values to the Params array in exactly the same order and number as they appear in the stored procedure declaration. Out parameters of the procedure can be accessed with the ParamByName procedure.

If the value of an input parameter was not included to the Params array, parameter default value is taken. Only parameters at the end of the list can be unincluded to the Params array. If the parameter has no default value, the NULL value is sent.

**Note:** Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. The stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For further examples of parameter usage see [ExecSQL](#), [ExecSQLEx](#).

### Example

For example, having stored function declaration presented in Example 1), you may execute it and retrieve its result with commands presented in Example 2):

```
Example 1)
CREATE procedure MY_SUM (      A INTEGER,      B INTEGER)
RETURNS (      RESULT INTEGER)
as begin
  Result = a + b;
end;
Example 2)
Label2.Caption:= MyIBCConnection1.ParamByName('Result').AsString;
```

See Also
- [ExecProcEx](#)
- [ExecSQL](#)
- [ExecSQLEx](#)
- [TIBCConnection.ParamByName](#)
- [TIBCConnection.SQL](#)
5.11.4.3.7 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:

```pascal
Connection.ExecProcEx('Some_Stored_Procedure', ['Param_Name1', 'Param_Value1', 'Param_Name4', 'Param_Value4']);
```

See Also
- ExecSQL
- ExecSQLEx
- ExecProc

Executes a SQL statement with parameters.

Class

TCustomDAConnection

Syntax

```pascal
function ExecSQL(const Text: string): variant;
overload; function ExecSQL(const Text: string; const Params: array of variant): variant; overload; virtual;
```

Parameters

- **Text**
  - a SQL statement to be executed.
- **Params**
  - Array of parameter values arranged in the same order as they appear in SQL statement.

Return Value

- Out parameter with the name Result will hold the result of function having data type dtString. Otherwise returns Null.

Remarks

Use the ExecSQL method to execute any SQL statement outside the TCustomDADataset or
**TCustomDASQL** components. Supply the **Params** array with the values of parameters arranged in the same order as they appear in a SQL statement which itself is passed to the **Text** string parameter.

**Params** array must contain all IN and OUT parameters defined in SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to an ExecSQL method.

Out parameter with the name **Result** will hold the result of function having data type dtString. If none of the parameters in the Text statement is named **Result**, ExecSQL will return Null.

To get the values of OUT parameters use **ParamByName** function.

**See Also**
- ExecSQLEx
- ExecProc

© 1997-2021 Devart. All Rights Reserved.

5.11.1.4.3.9 ExecSQLEx Method

Executes any SQL statement outside the TQuery or TSQL components.

**Class**

TCustomDAConnection

**Syntax**

```delphi
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

```
IBCConnection.ExecSQLEx('begin :A:= :B + :C; end;',
    ['A', 0, 'B', 5, 'C', 3]);
A:= IBCConnection.ParamByName('A').AsInteger;
```

See Also

- ExecSQL

© 1997-2021 Devart. All Rights Reserved.

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

5.11.1.4.3.11 GetKeyFieldNames Method

Provides a list of available key field names.

Class

TCustomDAConnection

Syntax

```pascal
procedure GetKeyFieldNames(const TableName: string; List: TStrings); virtual;
```

Parameters

- **TableName**
  - Holds the table name
- **List**
  - The list of available key field names

Return Value

- Key field name

Remarks

Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

See Also

- GetTableNames
- GetStoredProcNames
5.11.4.3.12 GetStoredProcNames Method

5.11.4.3.12 GetStoredProcNames Method

Returns a list of stored procedures from the server.

Class

TCustomDAConnection

Syntax

```pascal
procedure GetStoredProcNames(List: TStrings; AllProcs: boolean = False); virtual;
```

Parameters

- **List**
  A TStrings descendant that will be filled with the names of stored procedures in the database.

- **AllProcs**
  True, if stored procedures from all schemas or including system procedures (depending on the server) are returned. False otherwise.

Remarks

Call the GetStoredProcNames method to get the names of available stored procedures and functions. GetStoredProcNames populates a string list with the names of stored procs in the database. If AllProcs = True, the procedure returns to the List parameter the names of the stored procedures that belong to all schemas; otherwise, List will contain the names of functions that belong to the current schema.

**Note:** Any contents already in the target string list object are eliminated and overwritten by data produced by GetStoredProcNames.

See Also

- GetDatabaseNames
- GetTableNames
5.11.1.4.3.13 GetTableNames Method

Provides a list of available tables names.

Class

TCustomDAConnection

Syntax

procedure GetTableNames(List: TStrings; AllTables: boolean = False; OnlyTables: boolean = False); virtual;

Parameters

List
   A TStrings descendant that will be filled with table names.

AllTables
   True, if procedure returns all table names including the names of system tables to the List parameter.

OnlyTables

Remarks

Call the GetTableNames method to get the names of available tables. Populates a string list with the names of tables in the database. If AllTables = True, procedure returns all table names including the names of system tables to the List parameter, otherwise List will not contain the names of system tables. If AllTables = True, the procedure returns to the List parameter the names of the tables that belong to all schemas; otherwise, List will contain the names of the tables that belong to the current schema.

Note: Any contents already in the target string list object are eliminated and overwritten by the data produced by GetTableNames.

See Also

- GetDatabaseNames
- GetStoredProcNames

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

Sends a specified message through the TCustomDASQLMonitor component.

Class
TCustomDAConnection

Syntax
procedure MonitorMessage(const Msg: string);

Parameters
Msg
   Message text that will be sent.

Remarks
Call the MonitorMessage method to output specified message via the TCustomDASQLMonitor component.

See Also
- TCustomDASQLMonitor

5.11.1.4.3.15  Ping Method

Used to check state of connection to the server.

Class
TCustomDAConnection

Syntax
procedure Ping;

Remarks
The method is used for checking server connection state.
5.11.1.4.3.16 RemoveFromPool Method

Marks the connection that should not be returned to the pool after disconnect.

Class

TCustomDAConnection

Syntax

```pascal
procedure RemoveFromPool;
```

Remarks

Call the RemoveFromPool method to mark the connection that should be deleted after disconnect instead of returning to the connection pool.

See Also

- Pooling
- PoolingOptions

5.11.1.4.3.17 Rollback Method

Discards all current data changes and ends transaction.

Class

TCustomDAConnection

Syntax

```pascal
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
5.11.1.4.3.18 StartTransaction Method

StartTransaction

Begins a new user transaction.

Class

TCustomDAConnection

Syntax

procedure StartTransaction; virtual;

Remarks

Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction property. If InTransaction is True, indicating that a transaction is already in progress, a subsequent call to StartTransaction without first calling Commit or Rollback to end the current transaction raises EDatabaseError. Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes, or Rollback to cancel them.

See Also

Commit
Rollback
InTransaction

5.11.1.4.4 Events

Events of the TCustomDAConnection class.

For a complete list of the TCustomDAConnection class members, see the TCustomDAConnection Members topic.
This event occurs when connection was lost.

Class

TCustomDAConnection

Syntax

property OnConnectionLost: TConnectionLostEvent;

Remarks

Write the OnConnectionLost event handler to process fatal errors and perform failover.

Note: To use the OnConnectionLost event handler, you should explicitly add the MemData unit to the 'uses' list and set the TCustomDAConnection.Options.LocalFailover property to True.
5.11.1.4.2 OnError Event

This event occurs when an error has arisen in the connection.

Class

TCustomDAConnection

Syntax

property OnError: TDAConnectionErrorEvent;

Remarks

Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.5 TCustomDADataset Class

Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.

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

Unit

DBAccess

Syntax

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.
**TMemDataSet**

**TCustomDADataset**

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.1 Members

**TCustomDADataset** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong></td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong></td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong></td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>FetchRows</strong></td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong></td>
<td>Used to change the WHERE</td>
</tr>
<tr>
<td><strong>FinalSQL</strong></td>
<td>clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td><strong>IsQuery</strong></td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>KeyFields</strong></td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>MacroCount</strong></td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong></td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong></td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong></td>
<td>Used to specify the data source component which binds current dataset to the</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of TCustomDADataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
</tbody>
</table>
### 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</strong></td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td><strong>ApplyRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Clears all pending cached updates from cache and...</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched</td>
<td>Used to find out whether TCustomDADataSet has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>Used to learn whether TCustomDADataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that restores dataset in its prior state.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>matches or is greater than the values specified in the KeyValues parameter.</td>
<td></td>
</tr>
<tr>
<td>FindParam</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>GetBlob (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>GetDataType</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td>Locate (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Excludes features that don't need to be included to the <strong>TMemDataSet.Locate</strong> method of <strong>TDataSet</strong>.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Sets or uses parameter information for a specific parameter based on its</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong></td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong></td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td>(inherited from TMemDataSet) Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td>(inherited from TMemDataSet) Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong></td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td>(inherited from TMemDataSet) Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td>(inherited from TMemDataSet) Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong></td>
<td>(inherited from TMemDataSet) Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong></td>
<td>(inherited from TMemDataSet) Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong></td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td><strong>UnLock</strong></td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td>(inherited from TMemDataSet) Frees the resources allocated for a previously prepared query on the</td>
</tr>
</tbody>
</table>
**UpdateResult** (inherited from **TMemDataSet**)

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**UpdateStatus** (inherited from **TMemDataSet**)

Indicates the current update status for the dataset when cached updates are enabled.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong></td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td><strong>AfterFetch</strong></td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong></td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong></td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong></td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td><strong>OnUpdateError</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td><strong>OnUpdateRecord</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>
Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates</td>
<td>(inherited from TMemDataSet) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions</td>
<td>Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>DataTypeMap</td>
<td>Used to set data type mapping rules.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DetailFields</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>FetchRows</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from <strong>TMemDataSet</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of TCustomDADataset object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the</td>
</tr>
</tbody>
</table>
See Also
- TCustomDADataSet Class
- TCustomDADataSet Class Members

5.11.1.5.2.1 BaseSQL Property

Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.

Class
TCustomDADataSet

Syntax

```pascal
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.
5.11.1.5.2.2 Conditions Property

Used to add WHERE conditions to a query

Class

TCustomDADataSet

Syntax

```
property Conditions: TDAConditions stored False;
```

See Also

- TDAConditions

5.11.1.5.2.3 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

TCustomDADataSet

Syntax

```
property Connection: 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.

5.11.1.5.2.4  DataTypeMap Property

Used to set data type mapping rules

Class

TCustomDACDataSet

Syntax

```property```  
**Data**TypeMap: TDA**Map**Rules stored  
IsMapRulesStored;

See Also

- TDA**Map**Rules

5.11.1.5.2.5  Debug Property

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

Class

TCustomDACDataSet

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

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

**See Also**
- TCustomDASQL.Debug

Class
- TCustomDADataSet

**Syntax**

```pascal
property DetailFields: string;
```

**Remarks**

Use the DetailFields property to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship. DetailFields is a string containing one or more field names in the detail table. Separate field names with semicolons.

Use Field Link Designer to set the value in design time.

**See Also**
- MasterFields
- MasterSource
5.11.1.5.2.7 Disconnected Property

Used to keep dataset opened after connection is closed.

Class

TCustomDADataSet

Syntax

property Disconnected: boolean;

Remarks

Set the Disconnected property to True to keep dataset opened after connection is closed.

5.11.1.5.2.8 FetchRows Property

Used to define the number of rows to be transferred across the network at the same time.

Class

TCustomDADataSet

Syntax

property FetchRows: integer default 25;

Remarks

The number of rows that will be transferred across the network at the same time. This property can have a great impact on performance. So it is preferable to choose the optimal value of the FetchRows property for each SQL statement and software/hardware configuration experimentally.

The default value is 25.
5.11.1.5.2.9 FilterSQL Property

Used to change the WHERE clause of SELECT statement and reopen a query.

Class

TCustomDADataSet

Syntax

```
property FilterSQL: string;
```

Remarks

The FilterSQL property is similar to the Filter property, but it changes the WHERE clause of SELECT statement and reopens query. Syntax is the same to the WHERE clause.

Note: the FilterSQL property adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

Example

```
Query1.FilterSQL := 'Dept >= 20 and DName LIKE ''M%'';
```

See Also

- AddWhere

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

5.11.1.5.2.10 FinalSQL Property

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Class

TCustomDADataSet

Syntax

```
property FinalSQL: string;
```

Remarks

Use FinalSQL to return SQL text with all changes performed by AddWhere, SetOrderBy, and
FilterSQL, and with expanded macros. This is the exact statement that will be passed on to the database server.

See Also
- FinalSQL
- AddWhere
- SaveSQL
- SQLSaved
- RestoreSQL
- BaseSQL

5.11.1.5.2.11 IsQuery Property

Used to check whether SQL statement returns rows.

Class
TCustomDADataSet

Syntax

```property
IsQuery: boolean;
```

Remarks

After the TCustomDADataSet component is prepared, the IsQuery property returns True if SQL statement is a SELECT query.

Use the IsQuery property to check whether the SQL statement returns rows or not.

IsQuery is a read-only property. Reading IsQuery on unprepared dataset raises an exception.

5.11.1.5.2.12 KeyFields Property

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
Class

TCustomDADataset

Syntax

```
property KeyFields: string;
```

Remarks

TCustomDADataset uses the KeyFields property to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database. For this feature KeyFields may hold a list of semicolon-delimited field names. If KeyFields is not defined before opening a dataset, TCustomDADataset requests information about primary keys from server sending an additional query.

KeyFields property may hold the name of a field which will be later assigned with an InterBase sequenced values. Beforehand InterBase generator must be created and its name passed to the TCustomIBCDataSet.KeyGenerator property. Sequences are generated when either Insert or Post method is called. Which of these two methods is used to modify the database is determined by the TCustomIBCDataSet.GeneratorMode property.

**Note:** 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
- TCustomIBCDataSet.KeyGenerator
- TCustomIBCDataSet.GeneratorMode

© 1997-2021
Devart. All Rights Reserved.

5.11.1.5.2.13 MacroCount Property

Used to get the number of macros associated with the Macros property.
**TCustomDADataSet**

**Syntax**

```delphi
property MacroCount: word;
```

**Remarks**

Use the MacroCount property to get the number of macros associated with the Macros property.

**See Also**

- [Macros](#)

---

5.11.1.5.2.14 Macros Property

Makes it possible to change SQL queries easily.

**Class**

**TCustomDADataSet**

**Syntax**

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

**Remarks**

With the help of macros you can easily change SQL query text at design- or runtime. Marcos extend abilities of parameters and allow to change conditions in a WHERE clause or sort order in an ORDER BY clause. You just insert &MacroName in the SQL query text and change value of macro in the Macro property editor at design time or call the MacroByName function at run time. At the time of opening the query macro is replaced by its value.

**Example**

```delphi
IBCQuery.SQL.Text := 'SELECT * FROM Dept ORDER BY &Order';
IBCQuery_MACROByName('Order').Value := 'DeptNo';
IBCQuery.Open;
```

**See Also**

- [TMacro](#)
5.11.1.5.2.15 MasterFields Property

Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

Class

TCustomDADataSet

Syntax

```plaintext
property MasterFields: string;
```

Remarks

Use the MasterFields property after setting the `MasterSource` property to specify the names of one or more fields that are used as foreign keys for this dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in these fields are used to select corresponding records in this table for display.

Use Field Link Designer to set the values at design time after setting the MasterSource property.

See Also

- `DetailFields`
- `MasterSource`
- `Master/Detail Relationships`
5.11.1.5.2.16  MasterSource Property

Used to specify the data source component which binds current dataset to the master one.

Class

TCustomDADataSet

Syntax

```object
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`

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

Set the properties of Options to specify the behaviour of a TCustomDADataSet object.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPrepare</td>
<td>Used to execute automatic Prepare on the query execution.</td>
</tr>
<tr>
<td>CacheCalcFields</td>
<td>Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td>CompressBlobMode</td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used to request default values/expressions from the server and assign them to the DefaultExpression property.</td>
</tr>
<tr>
<td>DetailDelay</td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td>FieldsOrigin</td>
<td>Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td>FlatBuffers</td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td>InsertAllSetFields</td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td>LocalMasterDetail</td>
<td>Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.</td>
</tr>
<tr>
<td>LongStrings</td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td>MasterFieldsNullable</td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td>NumberRange</td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.</td>
</tr>
<tr>
<td>QueryRecCount</td>
<td>Used for TCustomDADataSet to perform</td>
</tr>
</tbody>
</table>
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

© 1997-2021 Devart. All Rights Reserved.

Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

### Class
**TCustomDADataSet**

**Syntax**

```delphi
property ParamCheck: boolean default True;
```

**Remarks**

Use the ParamCheck property to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Set ParamCheck to True to let dataset automatically generate the Params property for the dataset based on a SQL statement.

Setting ParamCheck to False can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of stored procedures which themselves will accept parameterized values. The default value is True.

**See Also**

- [Params](#)

---

5.11.1.5.2.19  ParamCount Property

Used to indicate how many parameters are there in the Params property.

**Class**

**TCustomDADataSet**

**Syntax**

```delphi
property ParamCount: word;
```

**Remarks**

Use the ParamCount property to determine how many parameters are there in the Params property.

**See Also**

- [Params](#)
5.11.1.5.2.20 Params Property

Used to view and set parameter names, values, and data types dynamically.

**Class**

TCustomDADataSet

**Syntax**

```delphi
property Params: TDAParams stored False;
```

**Remarks**

Contains the parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

**See Also**

- ParamByName
- Macros

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.2.21 ReadOnly Property

Used to prevent users from updating, inserting, or deleting data in the dataset.

**Class**

TCustomDADataSet

**Syntax**

```delphi
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.

Class

TCustomDADataset

Syntax

property RefreshOptions: TRefreshOptions default [];
5.11.1.5.2.23 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class
TCustomDADataSet

Syntax

```pascal
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.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.2.24 SQL Property

Used to provide a SQL statement that a query component executes when its Open method is called.

Class
TCustomDADataSet

Syntax

```pascal
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
5.11.1.5.2.25  SQLDelete Property

Used to specify a SQL statement that will be used when applying a deletion to a record.

Class

TCustomDADataset

Syntax

property SQLDelete: TStrings;

Remarks

Use the SQLDelete property to specify the SQL statement that will be used when applying a deletion to a record. Statements can be parameterized queries.

To create a SQLDelete statement at design-time, use the query statements editor.

Example

DELETE FROM Orders
WHERE
       OrderID = :Old_OrderID

See Also

- SQL
- SQLInsert
- SQLUpdate
- SQLRefresh
5.11.1.5.2.26 SQLInsert Property

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Class

TCustomDADataset

Syntax

```property SQLInsert: TStrings;```

Remarks

Use the SQLInsert property to specify the SQL statement that will be used when applying an insertion to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. Parameters prefixed with OLD_ allow using current values of fields prior to the actual operation.

Use ReturnParam to return OUT parameters back to dataset.

To create a SQLInsert statement at design-time, use the query statements editor.

See Also

- SQL
- SQLUpdate
- SQLDelete
- SQLRefresh

5.11.1.5.2.27 SQLLock Property

Used to specify a SQL statement that will be used to perform a record lock.

Class

TCustomDADataset

Syntax

```property SQLLock: TStrings;```

Remarks
Use the SQLLock property to specify a SQL statement that will be used to perform a record lock. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

To create a SQLLock statement at design-time, the use query statement editor.

See Also
- SQL
- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh

5.11.1.5.2.28  SQLRecCount Property

Used to specify the SQL statement that is used to get the record count when opening a dataset.

Class
TCustomDADataSet

Syntax

property SQLRecCount: TStrings;

Remarks

Use the SQLRecCount property to specify the SQL statement that is used to get the record count when opening a dataset. The SQL statement is used if the TDADatasetOptions.QueryRecCount property is True, and the TCustomDADataSet.FetchAll property is False. Is not used if the FetchAll property is True.

To create a SQLRecCount statement at design-time, use the query statements editor.

See Also
- SQLInsert
- SQLUpdate
5.11.1.5.2.29 SQLRefresh Property

Used to specify a SQL statement that will be used to refresh current record by calling the RefreshRecord procedure.

Class

TCustomDADataSet

Syntax

property SQLRefresh: TStrings;

Remarks

Use the SQLRefresh property to specify a SQL statement that will be used to refresh current record by calling the RefreshRecord procedure.

Different behavior is observed when the SQLRefresh property is assigned with a single WHERE clause that holds frequently altered search condition. In this case the WHERE clause from SQLRefresh is combined with the same clause of the SELECT statement in a SQL property and this final query is then sent to the database server.

To create a SQLRefresh statement at design-time, use the query statements editor.

Example

```
SELECT Shipname FROM Orders
WHERE OrderID = :OrderID
```

See Also

- RefreshRecord
- SQL
- SQLInsert
- SQLUpdate
5.11.5.2.30  SQLUpdate Property

Used to specify a SQL statement that will be used when applying an update to a dataset.

Class

TCustomDADataSet

Syntax

```pascal
property SQLUpdate: TStrings;
```

Remarks

Use the SQLUpdate property to specify a SQL statement that will be used when applying an update to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

Use ReturnParam to return OUT parameters back to the dataset.

To create a SQLUpdate statement at design-time, use the query statement editor.

Example

```sql
UPDATE Orders
  set  
    ShipName = :ShipName
WHERE 
    OrderID = :Old_OrderID
```

See Also

- SQL
- SQLInsert
- SQLDelete
- SQLRefresh

© 1997-2021 Devart. All Rights Reserved.
5.11.1.5.2.31 UniDirectional Property

Used if an application does not need bidirectional access to records in the result set.

Class

TCustomDADataset

Syntax

property UniDirectional: boolean default False;

Remarks

Traditionally SQL cursors are unidirectional. They can travel only forward through a dataset. TCustomDADataset, however, permits bidirectional travelling by caching records. If an application does not need bidirectional access to the records in the result set, set UniDirectional to True. When UniDirectional is True, an application requires less memory and performance is improved. However, UniDirectional datasets cannot be modified. In FetchAll=False mode data is fetched on demand. When UniDirectional is set to True, data is fetched on demand as well, but obtained rows are not cached except for the current row. In case if the UniDirectional property is True, the FetchAll property will be automatically set to False. And if the FetchAll property is True, the UniDirectional property will be automatically set to False. The default value of UniDirectional is False, enabling forward and backward navigation.

Note: Pay attention to the specificity of using the FetchAll property=False

See Also

• TIBCQuery.FetchAll
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>DeferredPost (inherited from TMemDataSet)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>EditRangeEnd (inherited from TMemDataSet)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart (inherited from TMemDataSet)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Fetched</td>
<td>Used to find out whether TCustomDADataSet has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>Used to learn whether TCustomDADataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td>FindParam</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>GetDataType</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>Sets the current record in this dataset similar to the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Locate</td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Excludes features that don't need to be included to the <strong>TMemDataSet.Locate</strong> method of <strong>TDataSet</strong>.</td>
</tr>
<tr>
<td>Lock</td>
<td></td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Sets or uses parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>(inherited from <strong>TMemDataSet</strong>) Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>(inherited from <strong>TMemDataSet</strong>) Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveSQL</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetOrderBy</td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td>SetRange</td>
<td>(inherited from <strong>TMemDataSet</strong>) Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SQLSaved</td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td>UnLock</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

**See Also**
- TCustomDADataSet Class
- TCustomDADataSet Class Members

---

5.11.1.5.3.1 AddWhere Method

Adds condition to the WHERE clause of SELECT statement in the SQL property.

**Class**

TCustomDADataSet
Syntax

```objectivec
procedure AddWhere(const Condition: string);
```

Parameters

*Condition*

Holds the condition that will be added to the WHERE clause.

Remarks

Call the AddWhere method to add a condition to the WHERE clause of SELECT statement in the SQL property.

If SELECT has no WHERE clause, AddWhere creates it.

**Note:** the AddWhere method is implicitly called by `RefreshRecord`. The AddWhere method works for the SELECT statements only.

**Note:** the AddWhere method adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

See Also

- `DeleteWhere`

5.11.1.5.3.2 BreakExec Method

Breaks execution of the SQL statement on the server.

Class

`TCustomDADataSet`

Syntax

```objectivec
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
5.11.1.5.3.3  CreateBlobStream Method

Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.

Class

TCustomDADataSet

Syntax

function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override;

Parameters

Field
  Holds the BLOB field for reading data from or writing data to from a stream.

Mode
  Holds the stream mode, for which the stream will be used.

Return Value

The BLOB Stream.

Remarks

Call the CreateBlobStream method to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter. It must be a TBlobField component. You can specify whether the stream will be used for reading, writing, or updating the contents of the field with the Mode parameter.

5.11.1.5.3.4  DeleteWhere Method

Removes WHERE clause from the SQL property and assigns the BaseSQL property.

Class
**TCustomDADataSet**

**Syntax**

```plaintext
procedure DeleteWhere;
```

**Remarks**

Call the DeleteWhere method to remove WHERE clause from the SQL property and assign BaseSQL.

**See Also**
- `AddWhere`
- `BaseSQL`

© 1997-2021 Devart. All Rights Reserved.

## 5.11.1.5.3.5 Execute Method

Executes a SQL statement on the server.

**Class**

**TCustomDADataSet**

**Overload List**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Execute(Iters: integer; Offset: integer)</td>
<td>Used to perform Batch operations.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
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
- TIBCStoredProc.PrepareSQL
- TIBCStoredProc.Prepare

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.
5.11.1.5.3.6 Executing Method

Indicates whether SQL statement is still being executed.

Class

TCustomDADataSet

Syntax

```pascal
function Executing: boolean;
```

Return Value

True, if SQL statement is still being executed.

Remarks

Check Executing to learn whether TCustomDADataSet is still executing SQL statement.

5.11.1.5.3.7 Fetched Method

Used to find out whether TCustomDADataSet has fetched all rows.

Class

TCustomDADataSet

Syntax

```pascal
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.
5.11.1.5.3.8 Fetching Method

Used to learn whether TCustomDADataSet is still fetching rows.

Class

TCustomDADataSet

Syntax

function Fetching: boolean;

Return Value

True, if TCustomDADataSet is still fetching rows.

Remarks

Check Fetching to learn whether TCustomDADataSet is still fetching rows. Use the Fetching method if NonBlocking is True.

See Also

• Fetching

5.11.1.5.3.9 FetchingAll Method

Used to learn whether TCustomDADataSet is fetching all rows to the end.

Class

TCustomDADataSet

Syntax

function FetchingAll: boolean;

Return Value
True, if TCustomDADataset is fetching all rows to the end.

Remarks
Check FetchingAll to learn whether TCustomDADataset is fetching all rows to the end.

See Also
- Executing

5.11.1.5.3.10 FindKey Method

Searches for a record which contains specified field values.

Class
TCustomDADataset

Syntax

```pascal
function FindKey(const KeyValues: array of System.TVarRec): Boolean;
```

Parameters

**KeyValues**
Holds a key.

Remarks

Call the FindKey method to search for a specific record in a dataset. KeyValues holds a comma-delimited array of field values, that is called a key.

This function is provided for BDE compatibility only. It is recommended to use functions TMemDataSet.Locate and TMemDataSet.LocateEx for the record search.

5.11.1.5.3.11 FindMacro Method

Finds a macro with the specified name.

Class
TCustomDADataSet

Syntax

```pascal
function FindMacro(const Value: string): TMacro;
```

Parameters

- **Value**
  Holds the 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**

Class

- **TCustomDADataSet**

Syntax

```pascal
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

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.
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`

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`
5.11.1.5.3.14 GetDataType Method

Returns internal field types defined in the MemData and accompanying modules.

Class

TCustomDADataSet

Syntax

```
function GetDataType(const FieldName: string): integer; virtual;
```

Parameters

FieldName

Holds the name of the field.

Return Value

internal field types defined in MemData and accompanying modules.

Remarks

Call the GetDataType method to return internal field types defined in the MemData and accompanying modules. Internal field data types extend the TFieldType type of VCL by specific database server data types. For example, ftString, ftFile, ftObject.

5.11.1.5.3.15 GetFieldObject Method

Returns a multireference shared object from field.

Class

TCustomDADataSet

Syntax

```
function GetFieldObject(Field: TField): TSharedObject;
overload;
function GetFieldObject(Field: TField; RecBuf: TRecordBuffer): TSharedObject; overload;
```

© 1997-2021
Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
GetFieldObject(FieldDesc: TFieldDesc): TSharedObject;
 overload; function GetFieldObject(FieldDesc: TFieldDesc; RecBuf: TRecordBuffer): TSharedObject; overload; function GetFieldObject(const FieldName: string): TSharedObject; overload;

Parameters

FieldName
   Holds the field name.

Return Value
   multireference shared object.

Remarks

Call the GetFieldObject method to return a multireference shared object from field. If field
does not hold one of the TSharedObject descendants, GetFieldObject raises an exception.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.5.3.16 GetFieldPrecision Method

Retrieves the precision of a number field.

Class

TCustomDADataSet

Syntax

function GetFieldPrecision(const FieldName: string): integer;

Parameters

FieldName
   Holds the existing field name.

Return Value
   precision of number field.

Remarks

Call the GetFieldPrecision method to retrieve the precision of a number field. FieldName is the
name of an existing field.

See Also

* GetFieldScale
5.11.1.5.3.17 GetFieldScale Method

Retrieves the scale of a number field.

Class

**TCustomDADataSet**

Syntax

```pascal
function GetFieldScale(const FieldName: string): integer;
```

Parameters

- **FieldName**
  
  Holds the existing field name.

Return Value

The scale of the number field.

Remarks

Call the GetFieldScale method to retrieve the scale of a number field. FieldName is the name of an existing field.

See Also

- *GetFieldPrecision*

5.11.1.5.3.18 GetKeyFieldNames Method

Provides a list of available key field names.

Class

**TCustomDADataSet**

Syntax

```pascal
procedure GetKeyFieldNames(List: TStrings);
```

Parameters
List
   The list of available key field names

Return Value
   Key field name

Remarks
   Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

See Also
   - TCustomDACConnection.GetTableNames
   - TCustomDACConnection.GetStoredProcNames

© 1997-2021
Devart. All Rights Reserved.

5.11.1.5.3.19 GetOrderBy Method

Retrieves an ORDER BY clause from a SQL statement.

Class
   TCustomDADataset

Syntax

```
function GetOrderBy: string;
```

Return Value
   an ORDER BY clause from the SQL statement.

Remarks
   Call the GetOrderBy method to retrieve an ORDER BY clause from a SQL statement.

Note: GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

See Also
   - SetOrderBy

© 1997-2021
Devart. All Rights Reserved.
5.11.1.5.3.20 GotoCurrent Method

Sets the current record in this dataset similar to the current record in another dataset.

Class

TCustomDADataSet

Syntax

procedure GotoCurrent(DataSet: TCustomDADataSet);

Parameters

DataSet

Holds the TCustomDADataSet descendant to synchronize the record position with.

Remarks

Call the GotoCurrent method to set the current record in this dataset similar to the current record in another dataset. The key fields in both these DataSets must be coincident.

See Also

- TMemDataSet.Locate
- TMemDataSet.LocateEx

5.11.1.5.3.21 Lock Method

Locks the current record.

Class

TCustomDADataSet

Syntax

procedure Lock; virtual;

Remarks

Call the Lock method to lock the current record by executing the statement that is defined in the SQLLock property.

The Lock method sets the savepoint with the name LOCK_ + <component_name>. 

© 2021 Devart
5.11.5.3.22 MacroByName Method

Finds a macro with the specified name.

Class

TCustomDADataSet

Syntax

function MacroByName(const Value: string): TMacro;

Parameters

Value
  Holds the name of a macro to search for.

Return Value
  TMacro object if a match is found.

Remarks

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

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

To set a value to a macro, use the TMacro.Value property.

Example

IBCQuery.SQL := 'SELECT * FROM Scott.Dept ORDER BY &Order';
IBCQuery_MACROByName('Order').Value := 'DeptNo';
IBCQuery.Open;

See Also

• TMacro
5.11.1.5.3.23  ParamByName Method

Sets or uses parameter information for a specific parameter based on its name.

Class

TCustomDADataSet

Syntax

function ParamByName(const Value: string): TDAParam;

Parameters

Value

Holds the name of the parameter for which to retrieve information.

Return Value

a TDAParam object.

Remarks

Call the ParamByName method to set or use parameter information for a specific parameter based on its name. Name is the name of the parameter for which to retrieve information. ParamByName is used to set a parameter's value at runtime and returns a TDAParam object.

Example

The following statement retrieves the current value of a parameter called "Contact" into an edit box:

```
Edit1.Text := Query1.ParamsByName('Contact').AsString;
```

See Also

- Params
- FindParam
5.11.1.5.3.24  Prepare Method

Allocates, opens, and parses cursor for a query.

Class

TCustomDADataSet

Syntax

```delphi
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.

TCustomDADataSet automatically prepares a query if it is executed without being prepared first. After execution, TCustomDADataSet unprepares the query. When a query is executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.

The UnPrepare method unprepares a query.

**Note**: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- TMemDataSet.Prepared
- TMemDataSet.UnPrepare
- Options

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

5.11.1.5.3.25  RefreshRecord Method

Actualizes field values for the current record.

Class

TCustomDADataSet

Syntax
procedure RefreshRecord;

Remarks
Call the RefreshRecord method to actualize field values for the current record. RefreshRecord performs query to database and refetches new field values from the returned cursor.

See Also
- RefreshOptions
- SQLRefresh

© 1997-2021 Devart. All Rights Reserved.

Class
TCustomDADataSet

Syntax
procedure RestoreSQL;

Remarks
Call the RestoreSQL method to restore the SQL property modified by AddWhere and SetOrderBy.

See Also
- AddWhere
- SetOrderBy
- SaveSQL
- SQLSaved

© 1997-2021 Devart. All Rights Reserved.
5.11.1.5.3.27 SaveSQL Method

Saves the SQL property value to BaseSQL.

Class
TCustomDADataSet

Syntax

```pascal
procedure SaveSQL;
```

Remarks
Call the SaveSQL method to save the SQL property value to the BaseSQL property.

See Also
- SQLSaved
- RestoreSQL
- BaseSQL

5.11.1.5.3.28 SetOrderBy Method

Builds an ORDER BY clause of a SELECT statement.

Class
TCustomDADataSet

Syntax

```pascal
procedure SetOrderBy(const Fields: string);
```

Parameters

`Fields`

Holds the names of the fields which will be added to the ORDER BY clause.

Remarks
Call the SetOrderBy method to build an ORDER BY clause of a SELECT statement. The fields are identified by the comma-delimited field names.
Note: The GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

Example

```plaintext
Query1.SetOrderBy('DeptNo;DName');
```

See Also

- GetOrderBy

SQLSaved Method

Determines if the SQL property value was saved to the BaseSQL property.

Class

`TCustomDADataSet`

Syntax

```plaintext
function SQLSaved: boolean;
```

Return Value

True, if the SQL property value was saved to the BaseSQL property.

Remarks

Call the SQLSaved method to know whether the SQL property value was saved to the BaseSQL property.

UnLock Method

Releases a record lock.

Class

`TCustomDADataSet`

Syntax
procedure UnLock;

Remarks
Call the Unlock method to release the record lock made by the Lock method before.

Unlock is performed by rolling back to the savepoint set by the Lock method.

See Also
- Lock

5.11.1.5.4 Events

Events of the TCustomDADataSet class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td>OnUpdateError (inherited from TMemDataSet)</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord (inherited from TMemDataSet)</td>
<td>Occurs when a single update component can not</td>
</tr>
</tbody>
</table>
handle the updates.

See Also

- TCustomDADataSet Class
- TCustomDADataSet Class Members

5.11.1.5.4.1 AfterExecute Event

Occurs after a component has executed a query to database.

Class

TCustomDADataSet

Syntax

property AfterExecute: TAfterExecuteEvent;

Remarks

Occurs after a component has executed a query to database.

See Also

- TCustomDADataSet.Execute

5.11.1.5.4.2 AfterFetch Event

Occurs after dataset finishes fetching data from server.

Class

TCustomDADataSet

Syntax

property AfterFetch: TAfterFetchEvent;

Remarks
The AfterFetch event occurs after dataset finishes fetching data from server.

See Also
- BeforeFetch

5.11.1.5.4.3 AfterUpdateExecute Event

Occurs after executing insert, delete, update, lock and refresh operations.

Class
- TCustomDADataSet

Syntax

```property AfterUpdateExecute: TUpdateExecuteEvent;```

Remarks

Occurs after executing insert, delete, update, lock, and refresh operations. You can use AfterUpdateExecute to set the parameters of corresponding statements.

5.11.1.5.4.4 BeforeFetch Event

Occurs before dataset is going to fetch block of records from the server.

Class
- TCustomDADataSet

Syntax

```property BeforeFetch: TBeforeFetchEvent;```

Remarks

The BeforeFetch event occurs every time before dataset is going to fetch a block of records from the server. Set Cancel to True to abort current fetch operation. To get an example on how to interrupt fetch operation refer to the Loader demo.
5.11.1.5.4.5 BeforeUpdateExecute Event

Occurs before executing insert, delete, update, lock, and refresh operations.

Class

TCustomDADataSet

Syntax

property BeforeUpdateExecute: TUpdateExecuteEvent;

Remarks

Occurs before executing insert, delete, update, lock, and refresh operations. You can use BeforeUpdateExecute to set the parameters of corresponding statements.

See Also

• AfterUpdateExecute

5.11.1.6 TCustomDASQL Class

A base class for components executing SQL statements that do not return result sets.

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

Unit

DBAccess

Syntax

TCustomDASQL = class(TComponent);

Remarks
TCustomDASQL is a base class that defines functionality for descendant classes which access database using SQL statements. Applications never use TCustomDASQL objects directly. Instead they use descendants of TCustomDASQL.

Use TCustomDASQL when client application must execute SQL statement or call stored procedure on the database server. The SQL statement should not retrieve rows from the database.

5.11.1.6.1 Members

**TCustomDASQL class overview.**

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>Enables or disables changing screen cursor when executing commands in the NonBlocking mode.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return a SQL statement with expanded macros.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Indicates the number of parameters in the Params.</td>
</tr>
</tbody>
</table>
### InterBase Data Access Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Params</strong></td>
<td>Used to contain parameters for a SQL statement.</td>
</tr>
<tr>
<td><strong>ParamValues</strong></td>
<td>Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BreakExec</strong></td>
<td>Breaks execution of an SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>Checks whether TCustomDASQL still executes a SQL statement.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Finds a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Finds a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td>Frees the resources allocated for a previously</td>
</tr>
</tbody>
</table>
**Properties**

Properties of the `TCustomDASQL` class.

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

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>Enables or disables changing screen cursor when executing commands in the NonBlocking mode.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return a SQL statement with expanded macros.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>SQL queries easily.</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Indicates the number of parameters in the Params property.</td>
</tr>
<tr>
<td><strong>ParamValues</strong></td>
<td>Used to contain parameters for a SQL statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.</td>
</tr>
</tbody>
</table>

**See Also**
- **TCustomDASQL Class**
- **TCustomDASQL Class Members**

© 1997-2021 Devart. All Rights Reserved.

5.11.1.6.2.1 ChangeCursor Property

Enables or disables changing screen cursor when executing commands in the NonBlocking mode.

**Class**

*TCustomDASQL*
### Syntax

```
property ChangeCursor: boolean;
```

### Remarks

Set the ChangeCursor property to False to prevent the screen cursor from changing to crSQLArrow when executing commands in the NonBlocking mode. The default value is True.

### 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.

### Class

**TCustomDASQL**
TCustomDASQL

Syntax

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

Remarks

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

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

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

See Also
- TCustomDADataSet.Debug

5.11.1.6.2.4 FinalSQL Property

Used to return a SQL statement with expanded macros.

Class

TCustomDASQL

Syntax

```pascal
property FinalSQL: string;
```

Remarks

Read the FinalSQL property to return a SQL statement with expanded macros. This is the exact statement that will be passed on to the database server.

5.11.1.6.2.5 MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

**TCustomDASQL**

Syntax

```pascal
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also

- [Macros](#)

© 1997-2021 Devart. All Rights Reserved.

5.11.1.6.2.6 Macros Property

Makes it possible to change SQL queries easily.

Class

**TCustomDASQL**

Syntax

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

Remarks

With the help of macros you can easily change SQL query text at design- or runtime. Marcos extend abilities of parameters and allow to change conditions in a WHERE clause or sort order in an ORDER BY clause. You just insert &MacroName in the SQL query text and change value of macro in the Macro property editor at design time or call the MacroByName function at run time. At the time of opening the query macro is replaced by its value.

See Also

- [TMacro](#)
5.11.1.6.2.7 `ParamCheck` Property

Used to specify whether parameters for the `Params` property are implicitly generated when the SQL property is being changed.

**Class**

`TCustomDASQL`

**Syntax**

```property ParamCheck: boolean default True;```

**Remarks**

Use the `ParamCheck` property to specify whether parameters for the `Params` property are implicitly generated when the SQL property is being changed.

Set `ParamCheck` to `True` to let `TCustomDASQL` generate the `Params` property for the dataset based on a SQL statement automatically.

Setting `ParamCheck` to `False` can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of the stored procedures that will accept parameterized values themselves. The default value is `True`.

**See Also**

- `Params`

5.11.1.6.2.8 `ParamCount` Property

Indicates the number of parameters in the `Params` property.

**Class**

`TCustomDASQL`
Syntax

```
property ParamCount: word;
```

Remarks

Use the ParamCount property to determine how many parameters are there in the Params property.

Example

Setting parameters at runtime:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
begin
with IBCSQL do
begin
  SQL.Clear;
  SQL.Add('INSERT INTO Temp_Table(Id, Name)');
  SQL.Add('VALUES (:id, :Name)');
  ParamByName('Id').AsInteger := 55;
  Params[1].AsString := ' Green';
  Execute;
end;
```
5.11.1.6.2.10  ParamValues Property(Indexer)

Used to get or set the values of individual field parameters that are identified by name.

Class

TCustomDASQL

Syntax

```
property ParamValues[const ParamName: string]: Variant; default;
```

Parameters

**ParamName**

Holds parameter names separated by semicolon.

Remarks

Use the ParamValues property to get or set the values of individual field parameters that are identified by name.

Setting ParamValues sets the Value property for each parameter listed in the ParamName string. Specify the values as Variants.

Getting ParamValues retrieves an array of variants, each of which represents the value of one of the named parameters.

**Note:** The Params array is generated implicitly if ParamCheck property is set to True. If ParamName includes a name that does not match any of the parameters in Items, an exception is raised.
5.11.1.6.2.11 Prepared Property

Used to indicate whether a query is prepared for execution.

Class

TCustomDASQL

Syntax

```property
Prepared: boolean;
```

Remarks

Check the Prepared property to determine if a query is already prepared for execution. True means that the query has already been prepared. As a rule prepared queries are executed faster, but the preparation itself also takes some time. One of the proper cases for using preparation is parametrized queries that are executed several times.

See Also

- Prepare

5.11.1.6.2.12 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

TCustomDASQL

Syntax

```property
RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

Example
For correct initializing this property you should explicitly prepare SQL as it is shown in the example below.

```delphi
IBCSQL.SQL.Text := 'Update Employee set salary = :sal where emp_no = :no';
IBCSQL.Prepare;
IBCSQL.Execute;
AffRows := IBSQL.RowsAffected;
IBCSQL.Unprepare;
```

### 5.11.1.6.13 SQL Property

Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

**Class**

**TCustomDASQL**

**Syntax**

```delphi
property SQL: TStrings;
```

**Remarks**

Use the SQL property to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called. At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

**See Also**

- `FinalSQL`
- `TCustomDASQL.Execute`

### 5.11.1.6.3 Methods

Methods of the TCustomDASQL class.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BreakExec</td>
<td>Breaks execution of an SQL statement on the server.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Checks whether TCustomDASQL still executes a SQL statement.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindParam</td>
<td>Finds a parameter with the specified name.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Finds a parameter with the specified name.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>WaitExecuting</td>
<td>Waits until TCustomDASQL executes a SQL statement.</td>
</tr>
</tbody>
</table>

See Also

- TCustomDASQL Class
- TCustomDASQL Class Members

© 1997-2021 Devart. All Rights Reserved.  

5.11.6.3.1 BreakExec Method

Breaks execution of an SQL statement on the server.

Class

TCustomDASQL
Syntax

```plaintext
procedure BreakExec;
```

Remarks

Call the BreakExec method to break execution of an SQL statement on the server. It makes sense to call BreakExec only from another thread. Useful when NonBlocking is True.

See Also

- TCustomDASQL.Execute
- TCustomDADataSet.BreakExec

Executes a SQL statement on the server.

Class

TCustomDASQL

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Execute(Iter: integer; Offset: integer)</td>
<td>Used to perform Batch operations .</td>
</tr>
</tbody>
</table>

Executes a SQL statement on the server.

Class

TCustomDASQL

Syntax

```plaintext
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.

© 1997-2021 Devart. All Rights Reserved.

Used to perform Batch operations.

Class

`TCustomDASQL`

Syntax

```pascal
procedure Execute(Iters: integer; Offset: integer = 0); overload;
virtual;
```

Parameters

- **Iters**
  - Specifies the number of inserted rows.
- **Offset**
  - Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The Execute method executes the specified batch SQL query. See the Batch operations article for samples.

See Also

- Batch operations

© 1997-2021 Devart. All Rights Reserved.

5.11.1.6.3.3 Executing Method

Checks whether TCustomDASQL still executes a SQL statement.

Class
TCustomDASQL

Syntax

```delphi
function Executing: boolean;
```

Return Value

True, if a SQL statement is still being executed by TCustomDASQL.

Remarks

Check Executing to find out whether TCustomDASQL still executes a SQL statement.

FindMacro Method

Finds a macro with the specified name.

Class

TCustomDASQL

Syntax

```delphi
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
5.11.1.6.3.5  FindParam Method

Finds a parameter with the specified name.

Class

TCustomDASQL

Syntax

```pascal
function FindParam(const Value: string): TDAParam;
```

Parameters

Value

Holds the parameter name to search for.

Return Value

a TDAParam object, if a parameter with the specified name has been found. If it has not, returns nil.

Remarks

Call the FindParam method to find a parameter with the specified name in a dataset.

See Also

• ParamByName

5.11.1.6.3.6  MacroByName Method

Finds a macro with the specified name.

Class

TCustomDASQL

Syntax

```pascal
function MacroByName(const Value: string): TMacro;
```

Parameters
**ParamByName Method**

Finds a parameter with the specified name.

**Class**

`TCustomDASQL`

**Syntax**

```pascal
function ParamByName(const Value: string): TDAParam;
```

**Parameters**

- `Value`
  
  Holds the name of the parameter to search for.

**Return Value**

- a `TDAParam` object, if a match was found. Otherwise, an exception is raised.

**Remarks**

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 `TMacs.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`
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

```plaintext
IBCSQL.Execute;
Edit1.Text := IBCSQL.ParamsByName('Contact').AsString;
```

See Also

- **FindParam**

© 1997-2021 Devart. All Rights Reserved.

5.11.1.6.3.8 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

**TCustomDASQL**

Syntax

```plaintext
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.

TCustomDADADataSet automatically prepares a query if it is executed without being prepared first. After execution, TCustomDADADataSet unprepares the query. When a query is executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.

The UnPrepare method unprepares a query.

**Note:** When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- **Prepared**
- **UnPrepare**
5.11.1.6.3.9  UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

**Class**

`TCustomDASQL`

**Syntax**

```plaintext
procedure UnPrepare; virtual;
```

**Remarks**

Call the UnPrepare method to free resources allocated for a previously prepared query on the server and client sides.

**See Also**

- `Prepare`

---

5.11.1.6.3.10  WaitExecuting Method

Waits until `TCustomDASQL` executes a SQL statement.

**Class**

`TCustomDASQL`

**Syntax**

```plaintext
function WaitExecuting(TimeOut: integer = 0): boolean;
```

**Parameters**

`TimeOut`

Holds the time in seconds to wait while `TCustomDASQL` executes the SQL statement. Zero means infinite time.

**Return Value**

True, if the execution of a SQL statement was completed in the preset time.
Remarks

Call the WaitExecuting method to wait until TCustomDASQL executes a SQL statement.

See Also

- Executing

5.11.1.6.4 Events

Events of the TCustomDASQL class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a SQL statement has been executed.</td>
</tr>
</tbody>
</table>

See Also

- TCustomDASQL Class
- TCustomDASQL Class Members

5.11.1.6.4.1 AfterExecute Event

Occurs after a SQL statement has been executed.

Class

TCustomDASQL

Syntax

```property AfterExecute: TAfterExecuteEvent;```
Remarks

Occurs after a SQL statement has been executed. This event may be used for descendant components which use multithreaded environment.

See Also
- TCustomDASQL.Execute

5.11.1.7 TCustomDAUpdateSQL Class

A base class for components that provide DML statements for more flexible control over data modifications.

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

Unit

DBAccess

Syntax

TCustomDAUpdateSQL = class(TComponent);

Remarks

TCustomDAUpdateSQL is a base class for components that provide DML statements for more flexible control over data modifications. Besides providing BDE compatibility, this component allows to associate a separate component for each update command.

See Also
- TCustomIBCDataSet.UpdateObject
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataSet</strong></td>
<td>Used to hold a reference to the TCustomDADataset object that is being updated.</td>
</tr>
<tr>
<td><strong>DeleteObject</strong></td>
<td>Provides ability to perform advanced adjustment of the delete operations.</td>
</tr>
<tr>
<td><strong>DeleteSQL</strong></td>
<td>Used when deleting a record.</td>
</tr>
<tr>
<td><strong>InsertObject</strong></td>
<td>Provides ability to perform advanced adjustment of insert operations.</td>
</tr>
<tr>
<td><strong>InsertSQL</strong></td>
<td>Used when inserting a record.</td>
</tr>
<tr>
<td><strong>LockObject</strong></td>
<td>Provides ability to perform advanced adjustment of lock operations.</td>
</tr>
<tr>
<td><strong>LockSQL</strong></td>
<td>Used to lock the current record.</td>
</tr>
<tr>
<td><strong>ModifyObject</strong></td>
<td>Provides ability to perform advanced adjustment of modify operations.</td>
</tr>
<tr>
<td><strong>ModifySQL</strong></td>
<td>Used when updating a record.</td>
</tr>
<tr>
<td><strong>RefreshObject</strong></td>
<td>Provides ability to perform advanced adjustment of refresh operations.</td>
</tr>
<tr>
<td><strong>RefreshSQL</strong></td>
<td>Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apply</strong></td>
<td>Sets parameters for a SQL statement and executes it to update a record.</td>
</tr>
</tbody>
</table>
5.11.1.7.2 Properties

Properties of the **TCustomDAUpdateSQL** class.

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

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataSet</strong></td>
<td>Used to hold a reference to the TCustomDADataset object that is being updated.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.</td>
</tr>
</tbody>
</table>

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DeleteObject</strong></td>
<td>Provides ability to perform advanced adjustment of the delete operations.</td>
</tr>
<tr>
<td><strong>DeleteSQL</strong></td>
<td>Used when deleting a record.</td>
</tr>
<tr>
<td><strong>InsertObject</strong></td>
<td>Provides ability to perform advanced adjustment of insert operations.</td>
</tr>
<tr>
<td><strong>InsertSQL</strong></td>
<td>Used when inserting a record.</td>
</tr>
<tr>
<td><strong>LockObject</strong></td>
<td>Provides ability to perform advanced adjustment of lock operations.</td>
</tr>
<tr>
<td><strong>LockSQL</strong></td>
<td>Used to lock the current record.</td>
</tr>
<tr>
<td><strong>ModifyObject</strong></td>
<td>Provides ability to perform advanced adjustment of</td>
</tr>
</tbody>
</table>
### See Also
- [TCustomDAUpdateSQL Class](#)
- [TCustomDAUpdateSQL Class Members](#)

### 5.11.1.7.2.1  DataSet Property

Used to hold a reference to the TCustomDADataSet object that is being updated.

### Class
- **TCustomDAUpdateSQL**

### Syntax

```plaintext
property DataSet: TCustomDADataSet;
```

### Remarks

The DataSet property holds a reference to the TCustomDADataSet object that is being updated. Generally it is not used directly.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModifySQL</td>
<td>Used when updating a record.</td>
</tr>
<tr>
<td>RefreshObject</td>
<td>Provides ability to perform advanced adjustment of refresh operations.</td>
</tr>
<tr>
<td>RefreshSQL</td>
<td>Used to specify an SQL statement that will be used for refreshing the current record by <code>TCustomDADataSet.RefreshRecord</code> procedure.</td>
</tr>
</tbody>
</table>
5.11.1.7.2.2 DeleteObject Property

Provides ability to perform advanced adjustment of the delete operations.

Class

TCustomDAUpdateSQL

Syntax

property DeleteObject: TComponent;

Remarks

Assign SQL component or a TCustomIBCDataSet descendant to this property to perform advanced adjustment of the delete operations. In some cases this can give some additional performance. Use the same principle to set the SQL property of an object as for setting the DeleteSQL property.

See Also

• DeleteSQL

© 1997-2021 Devart. All Rights Reserved.

5.11.1.7.2.3 DeleteSQL Property

Used when deleting a record.

Class

TCustomDAUpdateSQL

Syntax

property DeleteSQL: TStrings;

Remarks

Set the DeleteSQL property to a DELETE statement to use when deleting a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.
5.11.1.7.2.4 InsertObject Property

Provides ability to perform advanced adjustment of insert operations.

Class

TCustomDAUpdateSQL

Syntax

property InsertObject: TComponent;

Remarks

Assign SQL component or TCustomIBCDataSet descendant to this property to perform advanced adjustment of insert operations. In some cases this can give some additional performance. Set the SQL property of the object in the same way as used for the InsertSQL property.

See Also

• InsertSQL

© 1997-2021 Devart. All Rights Reserved.

5.11.1.7.2.5 InsertSQL Property

Used when inserting a record.

Class

TCustomDAUpdateSQL

Syntax

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.

© 1997-2021 Devart. All Rights Reserved.
5.11.1.7.2.6  LockObject Property

Provides ability to perform advanced adjustment of lock operations.

Class

TCustomDAUpdateSQL

Syntax

```pascal
property LockObject: TComponent;
```

Remarks

Assign a SQL component or TCustomIBCDataSet descendant to this property to perform advanced adjustment of lock operations. In some cases that can give some additional performance. Set the SQL property of an object in the same way as used for the LockSQL property.

See Also

• LockSQL

© 1997-2021
Devart. All Rights Reserved.
5.11.1.7.2.8 ModifyObject Property

Provides ability to perform advanced adjustment of modify operations.

Class

TCustomDAUpdateSQL

Syntax

property ModifyObject: TComponent;

Remarks

Assign a SQL component or TCustomIBCDataSet descendant to this property to perform advanced adjustment of modify operations. In some cases this can give some additional performance. Set the SQL property of the object in the same way as used for the ModifySQL property.

See Also

• ModifySQL

5.11.1.7.2.9 ModifySQL Property

Used when updating a record.

Class

TCustomDAUpdateSQL

Syntax

property ModifySQL: TStrings;

Remarks

Set ModifySQL to an UPDATE statement to use when updating a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.
5.11.1.7.2.10 RefreshObject Property

Provides ability to perform advanced adjustment of refresh operations.

Class

TCustomDAUpdateSQL

Syntax

property RefreshObject: TComponent;

Remarks

Assign a SQL component or TCustomIBCDataSet descendant to this property to perform advanced adjustment of refresh operations. In some cases that can give some additional performance. Set the SQL property of the object in the same way as used for the RefreshSQL property.

See Also

• RefreshSQL

5.11.1.7.2.11 RefreshSQL Property

Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataSet.RefreshRecord procedure.

Class

TCustomDAUpdateSQL

Syntax

property RefreshSQL: TStrings;

Remarks

Use the RefreshSQL property to specify a SQL statement that will be used for refreshing the current record by the TCustomDADataSet.RefreshRecord procedure.

You can assign to SQLRefresh a WHERE clause only. In such a case it is added to SELECT defined by the SQL property by TCustomDADataSet.AddWhere.
To create a RefreshSQL statement at design time, use the query statements editor.

See Also
- TCustomDADataset.RefreshRecord

Class
TCustomDAUpdateSQL

Syntax

```pascal
property SQL[UpdateKind: TUpdateKind]: TStrings;
```

Parameters

* `UpdateKind` Specifies which of update SQL statements to return.

Remarks

Returns a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties, depending on the value of the UpdateKind index.

### Methods

Methods of the TCustomDAUpdateSQL class.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply</td>
<td>Sets parameters for a SQL</td>
</tr>
</tbody>
</table>
Sets parameters for a SQL statement and executes it to update a record.

Class

**TCustomDAUpdateSQL**

Syntax

```pascal
procedure Apply(UpdateKind: TUpdateKind); virtual;
```

Parameters

*UpdateKind*

Specifies which of update SQL statements to execute.

Remarks

Call the Apply method to set parameters for a SQL statement and execute it to update a record. UpdateKind indicates which SQL statement to bind and execute.

Apply is primarily intended for manually executing update statements from an OnUpdateRecord event handler.

**Note:** If a SQL statement does not contain parameters, it is more efficient to call **ExecSQL** instead of Apply.

See Also

- **ExecSQL**
5.11.1.7.2 ExecSQL Method

Executes a SQL statement.

Class

TCustomDAUpdateSQL

Syntax

procedure ExecSQL(UpdateKind: TUpdateKind);

Parameters

UpdateKind

Specifies the kind of update statement to be executed.

Remarks

Call the ExecSQL method to execute a SQL statement, necessary for updating the records belonging to a read-only result set when cached updates is enabled. UpdateKind specifies the statement to execute.

ExecSQL is primarily intended for manually executing update statements from the OnUpdateRecord event handler.

Note: To both bind parameters and execute a statement, call Apply.

See Also

• Apply

© 1997-2021
Devart. All Rights Reserved.

5.11.1.8 TDACCondition Class

Represents a condition from the TDAConditions list.

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

Unit

DBAccess

Syntax

TDACCondition = class(TCollectionItem);
Remarks
Manipulate conditions using **TDAConditions**.

See Also
- **TDAConditions**

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Indicates whether the condition is enabled or not</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the condition</td>
</tr>
<tr>
<td>Value</td>
<td>The value of the condition</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Disables the condition</td>
</tr>
<tr>
<td>Enable</td>
<td>Enables the condition</td>
</tr>
</tbody>
</table>

5.11.1.8.2 Properties

Properties of the **TDACondition** class.

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

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Enabled</td>
<td>Indicates whether the condition is enabled or not</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the condition</td>
</tr>
<tr>
<td>Value</td>
<td>The value of the condition</td>
</tr>
</tbody>
</table>

See Also
- TDACondition Class
- TDACondition Class Members

5.11.1.8.2.1 Enabled Property

Indicates whether the condition is enabled or not

Class

TDACondition

Syntax

```csharp
property Enabled: Boolean default True;
```

5.11.1.8.2.2 Name Property

The name of the condition

Class

TDACondition

Syntax

```csharp
property Name: string;
```
5.11.1.8.3 Value Property

The value of the condition

Class

TDACCondition

Syntax

```
property value: string;
```

Methods

Methods of the TDACCondition class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Disables the condition</td>
</tr>
<tr>
<td>Enable</td>
<td>Enables the condition</td>
</tr>
</tbody>
</table>

See Also

- TDACCondition Class
- TDACCondition Class Members

© 1997-2021 Devart. All Rights Reserved.

5.11.1.8.3.1 Disable Method

Disables the condition

Class

TDACCondition
5.11.1.8.3.2  Enable Method

Enables the condition

Class

**TDACondition**

5.11.1.9  TDAConditions Class

Holds a collection of **TDACondition** objects.

For a list of all members of this type, see **TDAConditions** members.

Unit

**DBAccess**

Syntax

```delphi
TDAConditions = class(TCollection);
```

Remarks

The given example code

```delphi
UniTable1.Conditions.Add('1','JOB="MANAGER"');
UniTable1.Conditions.Add('2','SAL>2500');
UniTable1.Conditions.Enable;
UniTable1.Open;
```

will return the following SQL:

```sql
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
```

© 2021 Devart
### TDAConditions class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>Used to iterate through all the conditions.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Indicates whether the condition is enabled</td>
</tr>
<tr>
<td>Items</td>
<td>Used to iterate through all conditions.</td>
</tr>
<tr>
<td>Text</td>
<td>The property returns condition names and values as CONDITION_NAME=CONDITION</td>
</tr>
<tr>
<td>WhereSQL</td>
<td>Returns the SQL WHERE condition added in the Conditions property.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Overloaded. Adds a condition to the WHERE clause of the query.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the condition</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables the condition</td>
</tr>
<tr>
<td>Enable</td>
<td>Enables the condition</td>
</tr>
<tr>
<td>Find</td>
<td>Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise nil.</td>
</tr>
<tr>
<td>Get</td>
<td>Retrieving a TDACondition object by its name. If found,</td>
</tr>
</tbody>
</table>
the TDACondition object is returned, otherwise - an exception is raised.

**IndexOf**

Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.

**Remove**

Removes the condition

### 5.11.1.9.2 Properties

Properties of the **TDAConditions** class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>Used to iterate through all the conditions.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Indicates whether the condition is enabled</td>
</tr>
<tr>
<td>Items</td>
<td>Used to iterate through all conditions.</td>
</tr>
<tr>
<td>Text</td>
<td>The property returns condition names and values as <strong>CONDITION_NAME=CONDITON</strong></td>
</tr>
<tr>
<td>WhereSQL</td>
<td>Returns the SQL WHERE condition added in the Conditions property.</td>
</tr>
</tbody>
</table>

See Also
- **TDAConditions Class**
- **TDAConditions Class Members**
5.11.1.9.2.1 Condition Property(Indexer)

Used to iterate through all the conditions.

Class

TDAConditions

Syntax

\textbf{property} \texttt{Condition[Index: Integer]: TDACondition;}

Parameters

\textit{Index}

5.11.1.9.2.2 Enabled Property

Indicates whether the condition is enabled

Class

TDAConditions

Syntax

\textbf{property} \texttt{Enabled: Boolean;}

5.11.1.9.2.3 Items Property(Indexer)

Used to iterate through all conditions.

Class

TDAConditions

Syntax

\textbf{property} \texttt{Items[Index: Integer]: TDACondition; default;}

© 2021 Devart
Parameters

Index
Holds an index in the range 0..Count - 1.

Remarks
Use the Items property to iterate through all conditions. Index identifies the index in the range 0..Count - 1. Items can reference a particular condition by its index, but the Condition property is preferred in order to avoid depending on the order of the conditions.

5.11.1.9.2.4  Text Property

The property returns condition names and values as CONDITION_NAME=CONDITION

Class
TDAConditions

Syntax

```property Text: string;```

5.11.1.9.2.5  WhereSQL Property

Returns the SQL WHERE condition added in the Conditions property.

Class
TDAConditions

Syntax

```property WhereSQL: string;```
Methods of the **TDACOnditions** class.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Overloaded. Adds a condition to the WHERE clause of the query.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the condition</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables the condition</td>
</tr>
<tr>
<td>Enable</td>
<td>Enables the condition</td>
</tr>
<tr>
<td>Find</td>
<td>Search for TDACOndition (the condition) by its name. If found, the TDACOndition object is returned, otherwise - nil.</td>
</tr>
<tr>
<td>Get</td>
<td>Retrieving a TDACOndition object by its name. If found, the TDACOndition object is returned, otherwise - an exception is raised.</td>
</tr>
<tr>
<td>IndexOf</td>
<td>Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the condition</td>
</tr>
</tbody>
</table>

See Also
- [TDACOnditions Class](#)
- [TDACOnditions Class Members](#)
5.11.1.9.3.1 Add Method

Adds a condition to the WHERE clause of the query.

Class
TDAConditions

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add(const Value: string; Enabled: Boolean)</td>
<td>Adds a condition to the WHERE clause of the query.</td>
</tr>
<tr>
<td>Add(const Name: string; const Value: string; Enabled: Boolean)</td>
<td>Adds a condition to the WHERE clause of the query.</td>
</tr>
</tbody>
</table>

Syntax

```pascal
function Add(const Value: string; Enabled: Boolean = True): TDACondition; overload;
```

Parameters

- **Value**
  - The value of the condition

- **Enabled**
  - Indicates that the condition is enabled

Remarks

If you want then to access the condition, you should use Add and its name in the Name parameter.

The given example code will return the following SQL:

```sql
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
```
Adds a condition to the WHERE clause of the query.

Class

**TDAConditions**

Syntax

```haskell
function Add(const Name: string; const Value: string; Enabled: Boolean = True): TDACondition; overload;
```

**Parameters**

- **Name**
  - Sets the name of the condition
- **Value**
  - The value of the condition
- **Enabled**
  - Indicates that the condition is enabled

**Remarks**

The given example code will return the following SQL:

```sql
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and (SAL<2500)
```

5.11.1.9.3.2  Delete Method

Deletes the condition

Class

**TDAConditions**

Syntax

```haskell
procedure Delete(Index: integer);
```
**Parameters**

*Index*

Index of the condition

5.11.1.9.3.3 Disable Method

Disables the condition

**Class**

*TDAConditions*

**Syntax**

```pascal
procedure Disable;
```

5.11.1.9.3.4 Enable Method

Enables the condition

**Class**

*TDAConditions*

**Syntax**

```pascal
procedure Enable;
```

5.11.1.9.3.5 Find Method

Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.

**Class**

*TDAConditions*
5.11.1.9.3.6  Get Method

Retrieving a TDACCondition object by its name. If found, the TDACCondition object is returned, otherwise - an exception is raised.

Class
TDACConditions

Syntax
function Get(const Name: string): TDACCondition;

Parameters
Name

5.11.1.9.3.7  IndexOf Method

Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.

Class
TDACConditions

Syntax
function IndexOf(const Name: string): Integer;

Parameters
Name

© 1997-2021 Devart. All Rights Reserved.
5.11.9.3.8 Remove Method

Removes the condition

Class

TDAConditions

Syntax

```pascal
procedure Remove(const Name: string);
```

Parameters

Name

Specifies the name of the removed condition

5.11.10 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

```pascal
TDACConnectionOptions = class(TPersistent);
```

5.11.10.1 Members

TDACConnectionOptions class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Properties of the TDAConnectionOptions class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowImplicitConnect</td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
<tr>
<td>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.</td>
</tr>
<tr>
<td>DisconnectedMode</td>
<td>Used to open a connection only when needed for performing a server call and closes after performing the operation.</td>
</tr>
<tr>
<td>KeepDesignConnected</td>
<td>Used to prevent an application from establishing a connection at the time of startup.</td>
</tr>
<tr>
<td>LocalFailover</td>
<td>If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
</tbody>
</table>
**DisconnectedMode**

Used to open a connection only when needed for performing a server call and closes after performing the operation.

**KeepDesignConnected**

Used to prevent an application from establishing a connection at the time of startup.

**LocalFailover**

If True, the `TCustomDACConnection.OnConnectionLost` event occurs and a failover operation can be performed after connection breaks.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowImplicitConnect</td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
</tbody>
</table>

### See Also

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

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

© 1997-2021 Devart. All Rights Reserved.

5.11.1.10.2.2 **DefaultSortType Property**

Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the `TMemDataSet.IndexFieldNames` property of a dataset.

**Class**

`TDACollectionOptions`

**Syntax**

**property** DefaultSortType: TSortType *default* stCaseSensitive;

**Remarks**

Use the DefaultSortType property to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the `TMemDataSet.IndexFieldNames` property of a dataset.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.10.2.3 **DisconnectedMode Property**

Used to open a connection only when needed for performing a server call and closes after performing the operation.
Class

TDACConnectionOptions

Syntax

```property
DisconnectedMode: boolean default False;
```

Remarks

If True, connection opens only when needed for performing a server call and closes after performing the operation. Datasets remain opened when connection closes. May be useful to save server resources and operate in unstable or expensive network. Drawback of using disconnect mode is that each connection establishing requires some time for authorization. If connection is often closed and opened it can slow down the application work. See the Disconnected Mode topic for more information.

Class

TDACConnectionOptions

Syntax

```property
KeepDesignConnected: boolean default True;
```

Remarks

At the time of startup prevents application from establishing a connection even if the Connected property was set to True at design-time. Set KeepDesignConnected to False to initialize the connected property to False, even if it was True at design-time.
5.11.1.10.2.5 LocalFailover Property

If True, the *TCustomDAConnection.OnConnectionLost* event occurs and a failover operation can be performed after connection breaks.

Class

**TDACConnectionOptions**

Syntax

```
property LocalFailover: boolean default False;
```

Remarks

If True, the *TCustomDAConnection.OnConnectionLost* event occurs and a failover operation can be performed after connection breaks. Read the *Working in an Unstable Network* topic for more information about using failover.

5.11.1.11 TDACConnectionSSLOptions Class

This class is used to set up the SSL options.

For a list of all members of this type, see **TDACConnectionSSLOptions** members.

Unit

**DBAccess**

Syntax

```
TDACConnectionSSLOptions = class(TPersistent);
```

5.11.1.11.1 Members

**TDACConnectionSSLOptions** class overview.

Properties
## 5.11.1.11.2 Properties

Properties of the **TDACConnectionSSLOptions** class.

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

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACert</td>
<td>Holds the path to the certificate authority file.</td>
</tr>
<tr>
<td>Cert</td>
<td>Holds the path to the client certificate.</td>
</tr>
<tr>
<td>CipherList</td>
<td>Holds the list of allowed SSL ciphers.</td>
</tr>
<tr>
<td>Key</td>
<td>Holds the path to the private client key.</td>
</tr>
</tbody>
</table>

### See Also
- **TDACConnectionSSLOptions Class**
- **TDACConnectionSSLOptions Class Members**
5.11.1.11.2.1  **CACert Property**

Holds the path to the certificate authority file.

**Class**

*TDACConnectionSSLOptions*

**Syntax**

```
property CACert: string;
```

**Remarks**

Use the CACert property to specify the path to the certificate authority file.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.11.2.2  **Cert Property**

Holds the path to the client certificate.

**Class**

*TDACConnectionSSLOptions*

**Syntax**

```
property Cert: string;
```

**Remarks**

Use the Cert property to specify the path to the client certificate.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.11.2.3  **CipherList Property**

Holds the list of allowed SSL ciphers.

**Class**

*TDACConnectionSSLOptions*
Syntax

```pascal
property CipherList: string;
```

Remarks

Use the CipherList property to specify the list of allowed SSL ciphers.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.11.2.4 Key Property

Holds the path to the private client key.

Class

`TDAConnectionSSLOptions`

Syntax

```pascal
property Key: string;
```

Remarks

Use the Key property to specify the path to the private client key.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.12 TDADataSetOptions Class

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

For a list of all members of this type, see `TDADataSetOptions` members.

Unit

`DBAccess`

Syntax

```pascal
TDADataSetOptions = class(TPersistent);
```

© 1997-2021 Devart. All Rights Reserved.
**TDADatasetOptions** class overview.

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoPrepare</strong></td>
<td>Used to execute automatic TCustomDADataset.Prepare on the query execution.</td>
</tr>
<tr>
<td><strong>CacheCalcFields</strong></td>
<td>Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td><strong>CompressBlobMode</strong></td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td><strong>DefaultValues</strong></td>
<td>Used to request default values/expressions from the server and assign them to the DefaultExpression property.</td>
</tr>
<tr>
<td><strong>DetailDelay</strong></td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td><strong>FieldsOrigin</strong></td>
<td>Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td><strong>FlatBuffers</strong></td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td><strong>InsertAllSetFields</strong></td>
<td>Used to include all set dataset fields in the generated INSERT statement</td>
</tr>
<tr>
<td><strong>LocalMasterDetail</strong></td>
<td>Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LongStrings</strong></td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td><strong>MasterFieldsNullable</strong></td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td><strong>NumberRange</strong></td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.</td>
</tr>
<tr>
<td><strong>QueryRecCount</strong></td>
<td>Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td><strong>QuoteNames</strong></td>
<td>Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td><strong>RemoveOnRefresh</strong></td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td><strong>RequiredFields</strong></td>
<td>Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td><strong>ReturnParams</strong></td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td><strong>SetFieldsReadOnly</strong></td>
<td>Used for a dataset to set the ReadOnly property to True for all fields that do not refer to the server.</td>
</tr>
</tbody>
</table>
### Properties of the `TDADataSetOptions` class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPrepare</td>
<td>Used to execute automatic <code>TCustomDADataSet.Prepare</code> on the query execution.</td>
</tr>
<tr>
<td>CacheCalcFields</td>
<td>Used to enable caching of the <code>TField.Calculated</code> and <code>TField.Lookup</code> fields.</td>
</tr>
<tr>
<td>CompressBlobMode</td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used to request default values/expressions from the server and assign them to the DefaultExpression property.</td>
</tr>
<tr>
<td>DetailDelay</td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td>FieldsOrigin</td>
<td>Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td>FlatBuffers</td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td>InsertAllSetFields</td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td>LocalMasterDetail</td>
<td>Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.</td>
</tr>
<tr>
<td>LongStrings</td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td>MasterFieldsNullable</td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td>NumberRange</td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>QueryRecCount</strong></td>
<td>Used for TCustomDADataset to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td><strong>QuoteNames</strong></td>
<td>Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td><strong>RemoveOnRefresh</strong></td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td><strong>RequiredFields</strong></td>
<td>Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td><strong>ReturnParams</strong></td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td><strong>SetFieldsReadOnly</strong></td>
<td>Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpatingTable or can not be updated.</td>
</tr>
<tr>
<td><strong>StrictUpdate</strong></td>
<td>Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.</td>
</tr>
<tr>
<td><strong>TrimFixedChar</strong></td>
<td>Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
<tr>
<td><strong>UpdateAllFields</strong></td>
<td>Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
</tbody>
</table>
### UpdateBatchSize

**Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.**

**See Also**
- **TDADatasetOptions Class**
- **TDADatasetOptions Class Members**

### AutoPrepare Property

**Used to execute automatic TCustomDADataset.Prepare on the query execution.**

**Class**

**TDADatasetOptions**

**Syntax**

```plaintext
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.

© 1997-2021

Devart. All Rights Reserved.

Request Support     DAC Forum     Provide Feedback

### CacheCalcFields Property

**Used to enable caching of the TField-Calculated and TField-Lookup fields.**

**Class**

**TDADatasetOptions**

**Syntax**

© 2021 Devart
**property** CacheCalcFields: boolean **default** False;

**Remarks**
Use the CacheCalcFields property to enable caching of the TField.Calculated and TField.Lookup fields. It can be useful for reducing CPU usage for calculated fields. Using caching of calculated and lookup fields increases memory usage on the client side.

5.11.1.12.2.3 CompressBlobMode Property

Used to store values of the BLOB fields in compressed form.

**Class**
TDADatasetOptions

**Syntax**

```plaintext
property CompressBlobMode: TCompressBlobMode **default** cbNone;
```

**Remarks**
Use the CompressBlobMode property to store values of the BLOB fields in compressed form. Add the MemData unit to uses list to use this option. Compression rate greatly depends on stored data, for example, usually graphic data compresses badly unlike text.

5.11.1.12.2.4 DefaultValues Property

Used to request default values/expressions from the server and assign them to the DefaultExpression property.

**Class**
TDADatasetOptions

**Syntax**

```plaintext
property DefaultValues: boolean **default** False;
```
Remarks

If True, the default values/expressions are requested from the server and assigned to the DefaultExpression property of TField objects replacing already existent values.

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.

Class

TDADatasetOptions

Syntax

```property``` FieldsOrigin: boolean;

Remarks

Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.
If True, TCustomDADataSet fills the Origin property of the TField objects by appropriate value when opening a dataset.

5.11.12.2.7 FlatBuffers Property

Used to control how a dataset treats data of the ftString and ftVarBytes fields.

Class

TDADatasetOptions

Syntax

```pascal
property FlatBuffers: boolean default False;
```

Remarks

Use the FlatBuffers property to control how a dataset treats data of the ftString and ftVarBytes fields. When set to True, all data fetched from the server is stored in record pdata without unused tails.

5.11.12.2.8 InsertAllSetFields Property

Used to include all set dataset fields in the generated INSERT statement

Class

TDADatasetOptions

Syntax

```pascal
property InsertAllSetFields: boolean default False;
```

Remarks

If True, all set dataset fields, including those set to NULL explicitly, will be included in the generated INSERT statements. Otherwise, only set fields containing not NULL values will be included to the generated INSERT statement.
5.11.1.12.2.9 LocalMasterDetail Property

Used for TCustomDADataSet to use local filtering to establish master/detail relationship for
detail dataset and does not refer to the server.

Class

TDADataSetOptions

Syntax

property LocalMasterDetail: boolean default False;

Remarks

If True, for detail dataset in master-detail relationship TCustomDADataSet uses local filtering
for establishing master/detail relationship and does not refer to the server. Otherwise detail
dataset performs query each time a record is selected in master dataset. This option is useful
for reducing server calls number, server resources economy. It can be useful for slow
connection. The TMemDataSet.CachedUpdates mode can be used for detail dataset only
when this option is set to true. Setting the LocalMasterDetail option to True is not
recommended when detail table contains too many rows, because when it is set to False,
only records that correspond to the current record in master dataset are fetched.

5.11.1.12.2.10 LongStrings Property

Used to represent string fields with the length that is greater than 255 as TStringField.

Class

TDADataSetOptions

Syntax

property LongStrings: boolean default True;

Remarks

Use the LongStrings property to represent string fields with the length that is greater than 255
as TStringField, not as TMemoField.

5.11.1.12.2.11 MasterFieldsNullable Property

Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).

Class

**TDADatasetOptions**

Syntax

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

Remarks

Use the NumberRange property to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

5.11.1.12.2.12 NumberRange Property

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Class

**TDADatasetOptions**

Syntax

```plaintext
property NumberRange: boolean default False;
```
5.11.1.12.2.13  QueryRecCount Property

Used for TCustomDADataset to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.

Class

TDADatasetOptions

Syntax

```pascal
property QueryRecCount: boolean default False;
```

Remarks

If True, and the FetchAll property is False, TCustomDADataset performs additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records. Does not have any effect if the FetchAll property is True.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.12.2.14  QuoteNames Property

Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.

Class

TDADatasetOptions

Syntax

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

Remarks

If True, TCustomDADataset quotes all database object names in autogenerated SQL statements such as update SQL.

© 1997-2021
Devart. All Rights Reserved.
5.11.1.2.2.15 RemoveOnRefresh Property

Used for a dataset to locally remove a record that can not be found on the server.

Class

TDADatasetOptions

Syntax

```
property RemoveOnRefresh: boolean default True;
```

Remarks

When the RefreshRecord procedure can't find necessary record on the server and RemoveOnRefresh is set to True, dataset removes the record locally. Usually RefreshRecord can't find necessary record when someone else dropped the record or changed the key value of it.

This option makes sense only if the StrictUpdate option is set to False. If the StrictUpdate option is True, error will be generated regardless of the RemoveOnRefresh option value.

5.11.1.2.2.16 RequiredFields Property

Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.

Class

TDADatasetOptions

Syntax

```
property RequiredFields: boolean default True;
```

Remarks

If True, TCustomDADataSet sets the Required property of the TField objects for the NOT NULL fields. It is useful when table has a trigger which updates the NOT NULL fields.
5.11.1.12.2.17 ReturnParams Property

Used to return the new value of fields to dataset after insert or update.

**Class**

TDADatasetOptions

**Syntax**

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

**Remarks**

Use the ReturnParams property to return the new value of fields to dataset after insert or update. The actual value of field after insert or update may be different from the value stored in the local memory if the table has a trigger. When ReturnParams is True, OUT parameters of the SQLInsert and SQLUpdate statements is assigned to the corresponding fields.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.12.2.18 SetFieldsReadOnly Property

Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.

**Class**

TDADatasetOptions

**Syntax**

```plaintext
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.

© 1997-2021 Devart. All Rights Reserved.
5.11.1.12.2.19 StrictUpdate Property

Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.

Class

TDADatasetOptions

Syntax

property StrictUpdate: boolean default True;

Remarks

If True, TCustomDADataset raises an exception when the number of updated or deleted records is not equal 1. Setting this option also causes the exception if the RefreshRecord procedure returns more than one record. The exception does not occur when you execute SQL query, that doesn't return resultset.

Note: There can be problems if this option is set to True and triggers for UPDATE, DELETE, REFRESH commands that are defined for the table. So it is recommended to disable (set to False) this option with triggers.

TrimFixedChar specifies whether to discard all trailing spaces in the string fields of a dataset.

5.11.1.12.2.20 TrimFixedChar Property

Specifies whether to discard all trailing spaces in the string fields of a dataset.

Class

TDADatasetOptions

Syntax

property TrimFixedChar: boolean default True;

Remarks

Specifies whether to discard all trailing spaces in the string fields of a dataset.
5.11.1.12.2.21 UpdateAllFields Property

Used to include all dataset fields in the generated UPDATE and INSERT statements.

Class

TDADatasetOptions

Syntax

property UpdateAllFields: boolean default False;

Remarks

If True, all dataset fields will be included in the generated UPDATE and INSERT statements. Unspecified fields will have NULL value in the INSERT statements. Otherwise, only updated fields will be included to the generated update statements.

5.11.1.12.2.22 UpdateBatchSize Property

Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Class

TDADatasetOptions

Syntax

property UpdateBatchSize: Integer default 1;

Remarks

Use the UpdateBatchSize property to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch. Takes effect only when updating dataset in the TMemDataSet.CachedUpdates mode. The default value is 1.
5.11.1.13 TDAEncryption Class

Used to specify the options of the data encryption in a dataset.

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

Unit
DBAccess

Syntax

TDAEncryption = class(TPersistent);

Remarks

Set the properties of Encryption to specify the options of the data encryption in a dataset.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.13.1 Members

TDAEncryption class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryptor</td>
<td>Used to specify the encryptor class that will perform the data encryption.</td>
</tr>
<tr>
<td>Fields</td>
<td>Used to set field names for which encryption will be performed.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.11.1.13.2 Properties

Properties of the TDAEncryption class.

For a complete list of the TDAEncryption class members, see the TDAEncryption Members topic.
5.11.1.13.2.1 Encryptor Property

Used to specify the encryptor class that will perform the data encryption.

**Class**

**TDAEncryption**

**Syntax**

```pascal
property Encryptor: TCREncryptor;
```

**Remarks**

Use the Encryptor property to specify the encryptor class that will perform the data encryption.
5.11.1.13.2.2 Fields Property

Used to set field names for which encryption will be performed.

Class

TDAEncryption

Syntax

property Fields: string;

Remarks

Used to set field names for which encryption will be performed. Field names must be separated by semicolons.

5.11.1.14 TDAMapRule Class

Class that forms rules for Data Type Mapping.

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

Unit

DBAccess

Syntax

TDAMapRule = class(TMapRule);

Remarks

Using properties of this class, it is possible to change parameter values of the specified rules from the TDAMapRules set.
5.11.1.14.1 Members

**TDAMapRule** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLengthMax</td>
<td>Maximum DB field length, until which the rule is applied.</td>
</tr>
<tr>
<td>DBLengthMin</td>
<td>Minimum DB field length, starting from which the rule is applied.</td>
</tr>
<tr>
<td>DBScaleMax</td>
<td>Maximum DB field scale, until which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td>DBScaleMin</td>
<td>Minimum DB field Scale, starting from which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td>DbType</td>
<td>DB field type, that the rule is applied to.</td>
</tr>
<tr>
<td>FieldLength</td>
<td>The resultant field length in Delphi.</td>
</tr>
<tr>
<td>FieldName</td>
<td>DataSet field name, for which the rule is applied.</td>
</tr>
<tr>
<td>FieldScale</td>
<td>The resultant field Scale in Delphi.</td>
</tr>
<tr>
<td>FieldType</td>
<td>Delphi field type, that the specified DB type or DataSet field will be mapped to.</td>
</tr>
<tr>
<td>IgnoreErrors</td>
<td>Ignoring errors when converting data from DB to Delphi type.</td>
</tr>
</tbody>
</table>

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

5.11.1.14.2 Properties

Properties of the **TDAMapRule** class.

For a complete list of the **TDAMapRule** class members, see the **TDAMapRule Members**
Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLengthMax</td>
<td>Maximum DB field length, until which the rule is applied.</td>
</tr>
<tr>
<td>DBLengthMin</td>
<td>Minimum DB field length, starting from which the rule is applied.</td>
</tr>
<tr>
<td>DBScaleMax</td>
<td>Maximum DB field scale, until which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td>DBScaleMin</td>
<td>Minimum DB field Scale, starting from which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td>DBType</td>
<td>DB field type, that the rule is applied to.</td>
</tr>
<tr>
<td>FieldLength</td>
<td>The resultant field length in Delphi.</td>
</tr>
<tr>
<td>FieldName</td>
<td>DataSet field name, for which the rule is applied.</td>
</tr>
<tr>
<td>FieldScale</td>
<td>The resultant field Scale in Delphi.</td>
</tr>
<tr>
<td>FieldType</td>
<td>Delphi field type, that the specified DB type or DataSet field will be mapped to.</td>
</tr>
<tr>
<td>IgnoreErrors</td>
<td>Ignoring errors when converting data from DB to Delphi type.</td>
</tr>
</tbody>
</table>

See Also
- [TDAMapRule Class](#)
- [TDAMapRule Class Members](#)

© 1997-2021 Devart. All Rights Reserved.
5.11.1.14.2.1 DBLengthMax Property

Maximum DB field length, until which the rule is applied.

Class
TDAMapRule

Syntax

\[ \text{property DBLengthMax: Integer default rlAny;} \]

Remarks
Setting maximum DB field length, until which the rule is applied to the specified DB field.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.14.2.2 DBLengthMin Property

Minimum DB field length, starting from which the rule is applied.

Class
TDAMapRule

Syntax

\[ \text{property DBLengthMin: Integer default rlAny;} \]

Remarks
Setting minimum DB field length, starting from which the rule is applied to the specified DB field.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.14.2.3 DBScaleMax Property

Maximum DB field scale, until which the rule is applied to the specified DB field.

Class
TDAMapRule
5.11.1.14.2.4  DBScaleMin Property

Minimum DB field Scale, starting from which the rule is applied to the specified DB field.

Class
TDAMapRule

Syntax

```
property DBScaleMin: Integer default rlAny;
```

Remarks

Setting minimum DB field Scale, starting from which the rule is applied to the specified DB field.

5.11.1.14.2.5  DBType Property

DB field type, that the rule is applied to.

Class
TDAMapRule

Syntax

```
property DBType: Word default dtUnknown;
```

Remarks

Setting DB field type, that the rule is applied to. If the current rule is set for Connection, the
The resultant field length in Delphi.

Class

TDAMapRule

Syntax

```property FieldLength: Integer default rlAny;```

Remarks

Setting the Delphi field length after conversion.

DataSet field name, for which the rule is applied.

Class

TDAMapRule

Syntax

```property FieldName: string;```

Remarks

Specifies the DataSet field name, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields with such name in DataSets related to this Connection.
5.11.1.14.2.8 FieldScale Property

The resultant field Scale in Delphi.

Class
TDAMapRule

Syntax

```
property FieldScale: Integer default rlAny;
```

Remarks
Setting the Delphi field Scale after conversion.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.14.2.9 FieldType Property

Delphi field type, that the specified DB type or DataSet field will be mapped to.

Class
TDAMapRule

Syntax

```
property FieldType: TFieldType stored IsFieldTypeStored default ftUnknown;
```

Remarks
Setting Delphi field type, that the specified DB type or DataSet field will be mapped to.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.14.2.10 IgnoreErrors Property

Ignoring errors when converting data from DB to Delphi type.

Class
TDAMapRule
Syntax

```delphi
property IgnoreErrors: Boolean default False;
```

Remarks

Allows to ignore errors while data conversion in case if data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

5.11.15 TDAMapRules Class

Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.

For a list of all members of this type, see `TDAMapRules` members.

Unit

`DBAccess`

Syntax

```delphi
TDAMapRules = class(TMapRules);
```

Inheritance Hierarchy

`TMapRules`

```
TDAMapRules
```

5.11.15.1.1 Members

**TDAMapRules** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnoreInvalidRules</td>
<td>Used to avoid raising exception on mapping rules that can't be applied.</td>
</tr>
</tbody>
</table>
Properties of the **TDAMapRules** class.

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

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnoreInvalidRules</td>
<td>Used to avoid raising exception on mapping rules that can't be applied.</td>
</tr>
</tbody>
</table>

**See Also**

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

**IgnoreInvalidRules Property**

Used to avoid raising exception on mapping rules that can't be applied.

**Class**

**TDAMapRules**

**Syntax**

```delphi
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
5.11.1.16 TDAMetaData Class

A class for retrieving metainformation of the specified database objects in the form of dataset.

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

Unit

DBAccess

Syntax

TDAMetaData = class(TMemDataSet);

Remarks

TDAMetaData is a TDataSet descendant standing for retrieving metainformation of the specified database objects in the form of dataset. First of all you need to specify which kind of metainformation you want to see. For this you need to assign the TDAMetaData.MetaDataKind property. Provide one or more conditions in the TDAMetaData.Restrictions property to diminish the size of the resultset and get only information you are interested in.

Use the TDAMetaData.GetMetaDataKinds method to get the full list of supported kinds of meta data. With the TDAMetaData.GetRestrictions method you can find out what restrictions are applicable to the specified MetaDataKind.

Example

The code below demonstrates how to get information about columns of the 'emp' table:

```
MetaData.Connection := Connection;
MetaData.MetaDataKind := 'Columns';
MetaData.Restrictions.Values['TABLE_NAME'] := 'Emp';
MetaData.Open;
```

Inheritance Hierarchy
**TDAMetaData** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MetaDataKind</strong></td>
<td>Used to specify which kind of metainformation to show.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Determines whether a query is prepared for execution or not.</td>
</tr>
</tbody>
</table>
Ranged (inherited from TMemDataSet) Indicates whether a range is applied to a dataset.

Restrictions Used to provide one or more conditions restricting the list of objects to be described.

UpdateRecordTypes (inherited from TMemDataSet) Used to indicate the update status for the current record when cached updates are enabled.

UpdatesPending (inherited from TMemDataSet) Used to check the status of the cached updates buffer.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong> (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from TMemDataSet)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetMetaDataKinds</td>
<td>Used to get values acceptable in the MetaDataKind property.</td>
</tr>
<tr>
<td>GetRestrictions</td>
<td>Used to find out which restrictions are applicable to a certain MetaDataKind.</td>
</tr>
<tr>
<td>Locate (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx (inherited from TMemDataSet)</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Prepare (inherited from TMemDataSet)</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>RestoreUpdates (inherited from TMemDataSet)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord (inherited from TMemDataSet)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML (inherited from TMemDataSet)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange (inherited from TMemDataSet)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
</tbody>
</table>
| UnPrepare (inherited from TMemDataSet) | Frees the resources allocated for a previously
5.11.1.16.2 Properties

Properties of the **TDAMetaData** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to get or set the list of</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MetaDataKind</strong></td>
<td>Used to specify which kind of metainformation to show.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>Used to provide one or more conditions restricting the list of objects to be described.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

**See Also**
- **TDAMetaData Class**
- **TDAMetaData Class Members**

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

```property``` Connection: `TCustomDAConnection`;

**Remarks**

Use the `Connection` property to specify a connection object to use to connect to a data store.

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

At runtime, set the `Connection` property to reference an instanciated `TCustomDAConnection` object.

**5.11.16.2.2 MetaDataKind Property**

Used to specify which kind of metainformation to show.

**Class**

`TDAMetaData`

**Syntax**

```property``` MetaDataKind: `string`;

**Remarks**

This string property specifies which kind of metainformation to show. The value of this property should be assigned before activating the component. If `MetaDataKind` equals to an empty string (the default value), the full value list that this property accepts will be shown.

They are described in the table below:

<table>
<thead>
<tr>
<th>MetaDataKind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>show metainformation about columns of existing tables</td>
</tr>
<tr>
<td>Constraints</td>
<td>show metainformation about the constraints defined in the database</td>
</tr>
<tr>
<td>IndexColumns</td>
<td>show metainformation about indexed columns</td>
</tr>
<tr>
<td>Indexes</td>
<td>show metainformation about indexes in a database</td>
</tr>
<tr>
<td>MetaDataKinds</td>
<td>show the acceptable values of this property. You will get the same result if the <code>MetaDataKind</code> property is an empty string</td>
</tr>
<tr>
<td>ProcedurePara</td>
<td>show metainformation about parameters of existing procedures</td>
</tr>
</tbody>
</table>
meters

<table>
<thead>
<tr>
<th>Procedures</th>
<th>show metainformation about existing procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrictions</td>
<td>generates a dataset that describes which restrictions are applicable to each MetaDataKind</td>
</tr>
<tr>
<td>Tables</td>
<td>show metainformation about existing tables</td>
</tr>
<tr>
<td>Databases</td>
<td>show metainformation about existing databases</td>
</tr>
</tbody>
</table>

If you provide a value that equals neither of the values described in the table, an error will be raised.

See Also

- Restrictions

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
Methods of the TDMetaData class.

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

## Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong> (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from TMemDataSet)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>GetMetaDataKinds</strong></td>
<td>Used to get values acceptable in the MetaDataKind property.</td>
</tr>
<tr>
<td><strong>GetRestrictions</strong></td>
<td>Used to find out which restrictions are applicable to a certain MetaDataKind.</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
</tbody>
</table>
**UpdateStatus** (inherited from **TMemDataSet**)

Indicates the current update status for the dataset when cached updates are enabled.

See Also
- **TDAMetaData Class**
- **TDAMetaData Class Members**

## 5.11.1.16.3.1 GetMetaDataKinds Method

Used to get values acceptable in the MetaDataKind property.

### Class

**TDAMetaData**

### Syntax

```plaintext
procedure GetMetaDataKinds(List: TStrings);
```

### Parameters

- **List**
  
  Holds the object that will be filled with metadata kinds (restrictions).

### Remarks

Call the GetMetaDataKinds method to get values acceptable in the MetaDataKind property. The List parameter will be cleared and then filled with values.

See Also
- **MetaDataKind**
5.11.1.16.3.2 GetRestrictions Method

Used to find out which restrictions are applicable to a certain MetaDataKind.

Class
TDAMetaData

Syntax
procedure GetRestrictions(List: TStrings; const MetaDataKind: string);

Parameters
List
Holds the object that will be filled with metadata kinds (restrictions).

MetaDataKind
Holds the metadata kind for which restrictions are returned.

Remarks
Call the GetRestrictions method to find out which restrictions are applicable to a certain MetaDataKind. The List parameter will be cleared and then filled with values.

See Also
- Restrictions
- GetMetaDataKinds

5.11.1.17 TDAParam Class

A class that forms objects to represent the values of the parameters set.

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

Unit
DBAccess

Syntax
TDAParam = class(TParam);
Remarks
Use the properties of TDAParam to set the value of a parameter. Objects that use parameters create TDAParam objects to represent these parameters. For example, TDAParam objects are used by TCustomDASQL, TCustomDADataSet.

TDAParam shares many properties with TField, as both describe the value of a field in a dataset. However, a TField object has several properties to describe the field binding and the way the field is displayed, edited, or calculated, that are not needed in a TDAParam object. Conversely, TDAParam includes properties that indicate how the field value is passed as a parameter.

See Also
- TCustomDADataSet
- TCustomDASQL
- TDAParams

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

5.11.1.17.1 Members

TDAParam class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsBlob</td>
<td>Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td>AsBlobRef</td>
<td>Used to set and read the value of the BLOB parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to assign the value for a float field to a parameter.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to assign the value for an integer field to the parameter.</td>
</tr>
<tr>
<td>AsLargeInt</td>
<td>Used to assign the value for a LargeInteger field to the parameter.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AsMemo</td>
<td>Used to assign the value for a memo field to the parameter.</td>
</tr>
<tr>
<td>AsMemoRef</td>
<td>Used to set and read the value of the memo parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsSQLTimeStamp</td>
<td>Used to specify the value of the parameter when it represents a SQL timestamp field.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to assign the string value to the parameter.</td>
</tr>
<tr>
<td>AsWideString</td>
<td>Used to assign the Unicode string value to the parameter.</td>
</tr>
<tr>
<td>DataType</td>
<td>Indicates the data type of the parameter.</td>
</tr>
<tr>
<td>IsNull</td>
<td>Used to indicate whether the value assigned to a parameter is NULL.</td>
</tr>
<tr>
<td>ParamType</td>
<td>Used to indicate the type of use for a parameter.</td>
</tr>
<tr>
<td>Size</td>
<td>Specifies the size of a string type parameter.</td>
</tr>
<tr>
<td>Value</td>
<td>Used to represent the value of the parameter as Variant.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignField</td>
<td>Assigns field name and field value to a param.</td>
</tr>
<tr>
<td>AssignFieldValue</td>
<td>Assigns the specified field properties and value to a parameter.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Places the content of a specified file into a TDAParam object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Places the content from a stream into a TDAParam object.</td>
</tr>
<tr>
<td>SetBlobData</td>
<td>Overloaded. Writes the data from a specified buffer to</td>
</tr>
</tbody>
</table>
Properties of the **TDAParam** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsBlob</td>
<td>Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td>AsBlobRef</td>
<td>Used to set and read the value of the BLOB parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to assign the value for a float field to a parameter.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to assign the value for an integer field to the parameter.</td>
</tr>
<tr>
<td>AsLargeInt</td>
<td>Used to assign the value for a LargeInteger field to the parameter.</td>
</tr>
<tr>
<td>AsMemo</td>
<td>Used to assign the value for a memo field to the parameter.</td>
</tr>
<tr>
<td>AsMemoRef</td>
<td>Used to set and read the value of the memo parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsSQLTimeStamp</td>
<td>Used to specify the value of the parameter when it represents a SQL timestamp field.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to assign the string value to the parameter.</td>
</tr>
<tr>
<td>AsWideString</td>
<td>Used to assign the Unicode string value to the parameter.</td>
</tr>
</tbody>
</table>
IsNull

Used to indicate whether the value assigned to a parameter is NULL.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataType</strong></td>
<td>Indicates the data type of the parameter.</td>
</tr>
<tr>
<td><strong>ParamType</strong></td>
<td>Used to indicate the type of use for a parameter.</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>Specifies the size of a string type parameter.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>Used to represent the value of the parameter as Variant.</td>
</tr>
</tbody>
</table>

### See Also
- TDAParam Class
- TDAParam Class Members

5.11.1.17.2.1 AsBlob Property

Used to set and read the value of the BLOB parameter as string.

### Class

TDAParam

### Syntax

```pascal
    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. AsBlob is the value of the parameter when it represents the value of LONG RAW type.
5.11.1.17.2.2 AsBlobRef Property

Used to set and read the value of the BLOB parameter as a TBlob object.

Class
TDAParam

Syntax

property AsBlobRef: TBlob;

Remarks
Use the AsBlobRef property to set and read the value of the BLOB parameter as a TBlob object. Setting AsBlobRef will set the DataType property to ftBlob. Specifies the value of the parameter when it represents the value of LONG RAW type.

5.11.1.17.2.3 AsFloat Property

Used to assign the value for a float field to a parameter.

Class
TDAParam

Syntax

property AsFloat: double;

Remarks
Use the AsFloat property to assign the value for a float field to the parameter. Setting AsFloat will set the DataType property to dtFloat.

Read the AsFloat property to determine the value that was assigned to an output parameter, represented as Double. The value of the parameter will be converted to the Double value if possible.
Reserved.

5.11.1.17.2.4  AsInteger Property

Used to assign the value for an integer field to the parameter.

Class

TDAParam

Syntax

```
property AsInteger: LongInt;
```

Remarks

Use the AsInteger property to assign the value for an integer field to the parameter. Setting AsInteger will set the DataType property to dtInteger.

Read the AsInteger property to determine the value that was assigned to an output parameter, represented as a 32-bit integer. The value of the parameter will be converted to the Integer value if possible.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.17.2.5  AsLargeInt Property

Used to assign the value for a LargeInteger field to the parameter.

Class

TDAParam

Syntax

```
property AsLargeInt: Int64;
```

Remarks

Set the AsLargeInt property to assign the value for an Int64 field to the parameter. Setting AsLargeInt will set the DataType property to dtLargeint.

Read the AsLargeInt property to determine the value that was assigned to an output parameter, represented as a 64-bit integer. The value of the parameter will be converted to the Int64 value if possible.
5.11.1.17.2.6  AsMemo Property

Used to assign the value for a memo field to the parameter.

Class

TDAParam

Syntax

```plaintext
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. AsMemo is the value of the parameter when it represents the value of LONG type.

5.11.1.17.2.7  AsMemoRef Property

Used to set and read the value of the memo parameter as a TBlob object.

Class

TDAParam

Syntax

```plaintext
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. Specifies the value of the parameter when it represents the value of LONG type.
5.11.1.17.2.8 AsSQLTimeStamp Property

Used to specify the value of the parameter when it represents a SQL timestamp field.

Class

TDAParam

Syntax

```property AsSQLTimeStamp: TSQLTimeStamp;```

Remarks

Set the AsSQLTimeStamp property to assign the value for a SQL timestamp field to the parameter. Setting AsSQLTimeStamp sets the DataType property to ftTimeStamp.

5.11.1.17.2.9 AsString Property

Used to assign the string value to the parameter.

Class

TDAParam

Syntax

```property AsString: string;```

Remarks

Use the AsString property to assign the string value to the parameter. Setting AsString will set the DataType property to ftString.

Read the AsString property to determine the value that was assigned to an output parameter represented as a string. The value of the parameter will be converted to a string.
5.11.1.17.2.10  AsWideString Property

Used to assign the Unicode string value to the parameter.

Class
TDAParam

Syntax

```pascal
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.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.17.2.11  DataType Property

Indicates the data type of the parameter.

Class
TDAParam

Syntax

```pascal
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.

© 1997-2021 Devart. All Rights Reserved.
5.11.1.17.2.12 IsNull Property

Used to indicate whether the value assigned to a parameter is NULL.

Class

TDAParam

Syntax

```plaintext
property IsNull: boolean;
```

Remarks

Use the IsNull property to indicate whether the value assigned to a parameter is NULL.

5.11.1.17.2.13 ParamType Property

Used to indicate the type of use for a parameter.

Class

TDAParam

Syntax

```plaintext
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.

Note: The value of ParamType is important for LONG, LONG RAW, BLOB and CLOB parameters. To write data to database, set ptInput to ParamType, to read data from database, set ptOutput to ParamType.
5.11.1.17.2.14 Size Property

Specifies the size of a string type parameter.

Class
TDAParam

Syntax

```bulgarian
property size: integer default 0;
```

Remarks

Use the Size property to indicate the maximum number of characters the parameter may contain. Use the Size property only for Output parameters of the ftString, ftFixedChar, ftBytes, ftVarBytes, or ftWideString type.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.17.2.15 Value Property

Used to represent the value of the parameter as Variant.

Class
TDAParam

Syntax

```bulgarian
property value: variant stored IsValueStored;
```

Remarks

The Value property represents the value of the parameter as Variant.

Use Value in generic code that manipulates the values of parameters without the need to know the field type the parameter represent.

© 1997-2021 Devart. All Rights Reserved.
Methods of the **TDAParam** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AssignField</strong></td>
<td>Assigns field name and field value to a param.</td>
</tr>
<tr>
<td><strong>AssignFieldValue</strong></td>
<td>Assigns the specified field properties and value to a parameter.</td>
</tr>
<tr>
<td><strong>LoadFromFile</strong></td>
<td>Places the content of a specified file into a TDAParam object.</td>
</tr>
<tr>
<td><strong>LoadFromStream</strong></td>
<td>Places the content from a stream into a TDAParam object.</td>
</tr>
<tr>
<td><strong>SetBlobData</strong></td>
<td>Overloaded. Writes the data from a specified buffer to BLOB.</td>
</tr>
</tbody>
</table>

### See Also
- [TDAParam Class](#)
- [TDAParam Class Members](#)

© 1997-2021
Devart. All Rights Reserved.

5.11.1.17.3.1 **AssignField Method**

Assigns field name and field value to a param.

### Class

**TDAParam**

### Syntax

```pascal
procedure AssignField(Field: TField);
```

### Parameters
5.11.1.17.3.2 AssignFieldValue Method

Assigns the specified field properties and value to a parameter.

Class
TDAParam

Syntax

```
procedure AssignFieldValue(Field: TField; const Value: Variant);
virtual;
```

Parameters

Field
Holds the field the properties of which will be assigned to the parameter.

Value
Holds the value for the parameter.

Remarks

Call the AssignFieldValue method to assign the specified field properties and value to a parameter.

5.11.1.17.3.3 LoadFromFile Method

Places the content of a specified file into a TDAParam object.

Class
TDAParam
Syntax

**procedure** LoadFromFile(const FileName: **string**; BlobType: TBlobType);

**Parameters**

*FileName*
Holds the name of the file.

*BlobType*
Holds a value that modifies the DataType property so that this TDAParam object now holds the BLOB value.

**Remarks**

Use the LoadFromFile method to place the content of a file specified by FileName into a TDAParam object. The BlobType value modifies the DataType property so that this TDAParam object now holds the BLOB value.

**See Also**
- [LoadFromStream](#)

© 1997-2021 Devart. All Rights Reserved.

5.11.17.3.4  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

5.11.1.17.3.5  SetBlobData Method

Waits the data from a specified buffer to BLOB.

Class

TDAParam

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetBlobData(Buffer: TValueBuffer)</td>
<td>Writes the data from a specified buffer to BLOB.</td>
</tr>
<tr>
<td>SetBlobData(Buffer: IntPtr; Size: Integer)</td>
<td>Writes the data from a specified buffer to BLOB.</td>
</tr>
</tbody>
</table>

Syntax

```pascal
procedure SetBlobData(Buffer: TValueBuffer); overload;
```

Parameters

- Buffer
Holds the pointer to the data.

Writes the data from a specified buffer to BLOB.

Class

**TDAParam**

Syntax

```plaintext
procedure SetBlobData(Buffer: IntPtr; Size: Integer); overload;
```

Parameters

- **Buffer**
  - Holds the pointer to data.
- **Size**
  - Holds the number of bytes to read from the buffer.

Remarks

Call the `SetBlobData` method to write data from a specified buffer to BLOB.

5.11.1.18 **TDAParams Class**

This class is used to manage a list of TDAParam objects for an object that uses field parameters.

For a list of all members of this type, see **TDAParams** members.

Unit

**DBAccess**

Syntax

```plaintext
TDAParams = class(TParams);
```

Remarks

Use TDAParams to manage a list of TDAParam objects for an object that uses field
parameters. For example, TCustomDADataSet objects and TCustomDASQL objects use TDAParams objects to create and access their parameters.

See Also
- TCustomDADataSet.Params
- TCustomDASQL.Params
- TDAParam

Properties of the **TDAParams** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all parameters.</td>
</tr>
</tbody>
</table>
Items Property(Indexer)

Used to iterate through all parameters.

See Also
- **TDAParams Class**
- **TDAParams Class Members**

Class
**TDAParams**

Syntax

```pascal
property Items[Index: integer]: TDAParam; default;
```

Parameters

- **Index**
  - Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all parameters. Index identifies the index in the range 0..Count - 1. Items can reference a particular parameter by its index, but the ParamByName method is preferred in order to avoid depending on the order of the parameters.

5.11.1.18.3 Methods

Methods of the **TDAParams** class.

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

Public
## FindParam Method

Searches for a parameter with the specified name.

### Class

**TDAParams**

### Syntax

```delphi
function FindParam(const Value: string): TDAParam;
```

### Parameters

- **Value**
  - Holds the parameter name.

### Return Value

- a parameter, if a match was found. Nil otherwise.

### Remarks

Use the FindParam method to find a parameter with the name passed in Value. If a match is found, FindParam returns the parameter. Otherwise, it returns nil. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate more than one parameter at a time by name, use the GetParamList method instead. To get only the value of a named parameter, use the ParamValues property.
5.11.1.18.3.2 ParamByName Method

Searches for a parameter with the specified name.

Class

TDAParams

Syntax

function ParamByName(const Value: string): TDAParam;

Parameters

Value

Holds the parameter name.

Return Value

a parameter, if the match was found. otherwise an exception is raised.

Remarks

Use the ParamByName method to find a parameter with the name passed in Value. If a match was found, ParamByName returns the parameter. Otherwise, an exception is raised. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

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

© 1997-2021 Devart. All Rights Reserved.

5.11.1.19 TDATransaction Class

A base class that implements functionality for controlling transactions.

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

Unit

DBAccess

Syntax

TDATransaction = class(TComponent);

Remarks
TDATransaction is a base class for components implementing functionality for managing transactions.

Do not create instances of TDATransaction. Use descendants of the TDATransaction class instead.

5.11.1.19.1 Members

**TDATransaction** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if the transaction is active.</td>
</tr>
<tr>
<td>DefaultCloseAction</td>
<td>Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>Commits the current transaction.</td>
</tr>
<tr>
<td>Rollback</td>
<td>Discards all modifications of data associated with the current transaction and ends the transaction.</td>
</tr>
<tr>
<td>StartTransaction</td>
<td>Begins a new transaction.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnCommit</td>
<td>Occurs after the transaction has been successfully committed.</td>
</tr>
</tbody>
</table>
### Properties of the `TDATransaction` class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong></td>
<td>Used to determine if the transaction is active.</td>
</tr>
<tr>
<td><strong>DefaultCloseAction</strong></td>
<td>Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.</td>
</tr>
</tbody>
</table>

**See Also**

- [TDATransaction Class](#)
- [TDATransaction Class Members](#)
5.11.1.19.2.1 Active Property

Used to determine if the transaction is active.

Class
TDATransaction

Syntax
property Active: boolean;

Remarks
Indicates whether the transaction is active. This property is read-only.

5.11.1.19.2.2 DefaultCloseAction Property

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

Class
TDATransaction

Syntax
property DefaultCloseAction: TCRTransactionAction default taRollback;

Remarks
Use DefaultCloseAction to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

5.11.1.19.3 Methods

Methods of the TDATransaction class.

For a complete list of the TDATransaction class members, see the TDATransaction
Commit Method

Commits the current transaction.

Class
TDATransaction

Syntax

```plaintext
procedure Commit; virtual;
```

Remarks
Call the Commit method to commit the current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database, and then finishes the transaction.

See Also
- Rollback
- StartTransaction
5.11.1.19.3.2 Rollback Method

Discards all modifications of data associated with the current transaction and ends the transaction.

Class

TDATransaction

Syntax

```pascal
procedure Rollback; virtual;
```

Remarks

Call Rollback to cancel all data modifications made within the current transaction to the database server, and finish the transaction.

See Also

- Commit
- StartTransaction

5.11.1.19.3.3 StartTransaction Method

Begins a new transaction.

Class

TDATransaction

Syntax

```pascal
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
- TIBCTransaction.Active
- TIBCTransaction.Commit
- TIBCTransaction.Rollback

Events of the TDATransaction class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnCommit</td>
<td>Occurs after the transaction has been successfully committed.</td>
</tr>
<tr>
<td>OnCommitRetaining</td>
<td>Occurs after CommitRetaining has been executed.</td>
</tr>
<tr>
<td>OnError</td>
<td>Used to process errors that occur during executing a transaction.</td>
</tr>
<tr>
<td>OnRollback</td>
<td>Occurs after the transaction has been successfully rolled back.</td>
</tr>
<tr>
<td>OnRollbackRetaining</td>
<td>Occurs after RollbackRetaining has been executed.</td>
</tr>
</tbody>
</table>
5.11.1.19.4.1 OnCommit Event

Occurs after the transaction has been successfully committed.

Class
TDATransaction

Syntax

```property
OnCommit: TNotifyEvent;
```

Remarks

The OnCommit event fires when the M:Devart.Dac.TDATransaction.Commit method is executed, just after the transaction is successfully committed. In order to respond to the TIBCTransaction.CommitRetaining method execution, the OnCommitRetaining event is used. When an error occurs during commit, the OnError event fires.

See Also

- Commit
- TIBCTransaction.CommitRetaining
- OnCommitRetaining
- OnError

© 1997-2021
Devart. All Rights Reserved.

5.11.1.19.4.2 OnCommitRetaining Event

Occurs after CommitRetaining has been executed.

Class
TDATransaction
Syntax

```property`` OnCommitRetaining: TNotifyEvent;
```

Remarks

The OnCommitRetaining event fires when the CommitRetaining method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.Dac.TDATransaction.Commit method execution, the OnCommit event is used.

When an error occurs during commit, the OnError event fired.

See Also

- TIBCTransaction.CommitRetaining
- Commit
- OnCommit
- OnError

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.11.1.19.4.3  OnError Event

Used to process errors that occur during executing a transaction.

Class

TDATransaction

Syntax

```property`` OnError: TDATransactionErrorEvent;
```

Remarks

Add a handler to the OnError event to process errors that occur during executing a transaction control statements such as Commit, Rollback. Check the E parameter to get the error code.

See Also

- Commit
- Rollback
- StartTransaction
5.11.19.4.4 OnRollback Event

Occurs after the transaction has been successfully rolled back.

**Class**

**TDATransaction**

**Syntax**

```pascal
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 `TIBCTransaction.RollbackRetaining` method execution, the `OnRollbackRetaining` event is used.

When an error occurs during rollback, the `OnError` event fired.

**See Also**

- Rollback
- `TIBCTransaction.RollbackRetaining`
- `OnRollbackRetaining`
- `OnError`

© 1997-2021 Devart. All Rights Reserved.

5.11.19.4.5 OnRollbackRetaining Event

Occurs after RollbackRetaining has been executed.

**Class**

**TDATransaction**

**Syntax**

```pascal
property OnRollbackRetaining: TNotifyEvent;
```
Remarks

The OnRollbackRetaining event fires when the RollbackRetaining method is executed, just after the transaction is successfully rolled back. In order to respond to the Rollback method execution, the OnRollback event is used. When an error occurs during rollback, the OnError event fired.

See Also
- Rollback
- TIBCTransaction.RollbackRetaining
- OnRollback
- OnError

5.11.1.20 TMacro Class

Object that represents the value of a macro.

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

Unit

DBAccess

Syntax

TMacro = class(TCollectionItem);

Remarks

TMacro object represents the value of a macro. Macro is a variable that holds string value. You just insert & MacroName in a SQL query text and change the value of macro by the Macro property editor at design time or the Value property at run time. At the time of opening query macro is replaced by its value.

If by any reason it is not convenient for you to use the ' & ' symbol as a character of macro replacement, change the value of the MacroChar variable.

See Also
- TMacro
5.11.1.20.1 Members

**TMacro** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if the macro should be expanded.</td>
</tr>
<tr>
<td>AsDateTime</td>
<td>Used to set the TDateTime value to a macro.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to set the float value to a macro.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to set the integer value to a macro.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to assign the string value to a macro.</td>
</tr>
<tr>
<td>Name</td>
<td>Used to identify a particular macro.</td>
</tr>
<tr>
<td>Value</td>
<td>Used to set the value to a macro.</td>
</tr>
</tbody>
</table>

5.11.1.20.2 Properties

Properties of the **TMacro** class.

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

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsDateTime</td>
<td>Used to set the TDateTime value to a macro.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to set the float value to a macro.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to set the integer value to a macro.</td>
</tr>
</tbody>
</table>
AsString

Used to assign the string value to a macro.

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if the macro should be expanded.</td>
</tr>
<tr>
<td>Name</td>
<td>Used to identify a particular macro.</td>
</tr>
<tr>
<td>Value</td>
<td>Used to set the value to a macro.</td>
</tr>
</tbody>
</table>

See Also

- TMacro Class
- TMacro Class Members

5.11.1.20.2.1 Active Property

Used to determine if the macro should be expanded.

Class

TMacro

Syntax

```pascal
property Active: boolean default True;
```

Remarks

When set to True, the macro will be expanded, otherwise macro definition is replaced by null string. You can use the Active property to modify the SQL property.

The default value is True.

Example

```pascal
IBCQuery.SQL.Text := 'SELECT * FROM Dept WHERE DeptNo > 20 &Cond1';
IBCQuery.Macros[0].Value := 'and DName is NULL';
IBCQuery.Macros[0].Active := False;
```
5.11.1.20.2.2 AsDateTime Property

Used to set the TDateTime value to a macro.

Class

TMacro

Syntax

property AsDateTime: TDateTime;

Remarks

Use the AsDateTime property to set the TDateTime value to a macro.

5.11.1.20.2.3 AsFloat Property

Used to set the float value to a macro.

Class

TMacro

Syntax

property AsFloat: double;

Remarks

Use the AsFloat property to set the float value to a macro.

5.11.1.20.2.4 AsInteger Property

Used to set the integer value to a macro.

Class
**TMacro**

**Syntax**

```property AsInteger: integer;```

**Remarks**

Use the AsInteger property to set the integer value to a macro.

---

**5.11.1.20.2.5 AsString Property**

Used to assign the string value to a macro.

**Class**

**TMacro**

**Syntax**

```property AsString: string;```

**Remarks**

Use the AsString property to assign the string value to a macro. Read the AsString property to determine the value of macro represented as a string.

---

**5.11.1.20.2.6 Name Property**

Used to identify a particular macro.

**Class**

**TMacro**

**Syntax**

```property Name: string;```

**Remarks**
Use the Name property to identify a particular macro.

Class

`TMacro`

Syntax

```pascal
property value: string;
```

Remarks

Use the Value property to set the value to a macro.

5.11.1.21 TMacros Class

Controls a list of TMacro objects for the `TCustomDASQL.Macros` or `TCustomDADataset` components.

For a list of all members of this type, see `TMacros` members.

Unit

`DBAccess`

Syntax

```pascal
TMacros = class(TCollection);
```

Remarks

Use TMacros to manage a list of TMacro objects for the `TCustomDASQL` or `TCustomDADataset` components.

See Also
### TMacros class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all the macros parameters.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignValues</td>
<td>Copies the macros values and properties from the specified source.</td>
</tr>
<tr>
<td>Expand</td>
<td>Changes the macros in the passed SQL statement to their values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>IsEqual</td>
<td>Compares itself with another TMacro object.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Used to search for a macro with the specified name.</td>
</tr>
<tr>
<td>Scan</td>
<td>Creates a macros from the passed SQL statement.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Items</td>
<td>Used to iterate through all the macros parameters.</td>
</tr>
</tbody>
</table>

See Also
- **TMacros Class**
- **TMacros Class Members**

5.11.1.21.2.1  Items Property(Indexer)

Used to iterate through all the macros parameters.

**Class**

**TMacros**

**Syntax**

```delphi
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.

© 1997-2021
Devart. All Rights Reserved.
### AssignValues Method

Copies the macros values and properties from the specified source.

#### Class

**TMacros**

#### Syntax

```delphi
procedure AssignValues(Value: TMacros);
```

#### Parameters

- **Value**
  
  Holds the source to copy the macros values and properties from.

#### Remarks

The Assign method copies the macros values and properties from the specified source. Macros are not recreated. Only the values of macros with matching names are assigned.
5.11.1.21.3.2  Expand Method

Changes the macros in the passed SQL statement to their values.

Class

TMacros

Syntax

```
procedure Expand(var SQL: string);
```

Parameters

SQL
Holds the passed SQL statement.

Remarks

Call the Expand method to change the macros in the passed SQL statement to their values.

5.11.1.21.3.3  FindMacro Method

Finds a macro with the specified name.

Class

TMacros

Syntax

```
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.

Class

`TMacros`

Syntax

```plaintext
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.

Class

`TMacros`

Syntax

```plaintext
function MacroByName(const Value: string): TMacro;
```

Parameters
**Value**
Holds a name of the macro to search for.

**Return Value**
TMacro object, if a macro with specified name was found.

**Remarks**
Call the MacroByName method to find a Macro with the name passed in Value. If a match is found, MacroByName returns the Macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the Items property to avoid depending on the order of the items.

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

To set a value to a macro, use the TMacro.Value property.

---

**5.11.1.21.3.6 Scan Method**

Creates a macros from the passed SQL statement.

**Class**

**TMacros**

**Syntax**

```plaintext
procedure Scan(const SQL: string);
```

**Parameters**

**SQL**
Holds the passed SQL statement.

**Remarks**

Call the Scan method to create a macros from the passed SQL statement. On that all existing TMacro objects are cleared.
5.11.1.22 TPoolingOptions Class

This class allows setting up the behaviour of the connection pool.

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

Unit

DBAccess

Syntax

TPoolingOptions = class(TPersistent);

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionLifetime</td>
<td>Used to specify the maximum time during which an opened connection can be used by connection pool.</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>Used to specify the maximum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td>MinPoolSize</td>
<td>Used to specify the minimum number of connections that can be opened in the connection pool.</td>
</tr>
<tr>
<td>Validate</td>
<td>Used for a connection to be validated when it is returned from the pool.</td>
</tr>
</tbody>
</table>
5.11.1.22.2 Properties

Properties of the **TPoolingOptions** class.

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

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ConnectionLifetime</strong></td>
<td>Used to specify the maximum time during which an opened connection can be used by connection pool.</td>
</tr>
<tr>
<td><strong>MaxPoolSize</strong></td>
<td>Used to specify the maximum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td><strong>MinPoolSize</strong></td>
<td>Used to specify the minimum number of connections that can be opened in the connection pool.</td>
</tr>
<tr>
<td><strong>Validate</strong></td>
<td>Used for a connection to be validated when it is returned from the pool.</td>
</tr>
</tbody>
</table>

**See Also**
- [TPoolingOptions Class](#)
- [TPoolingOptions Class Members](#)

© 1997-2021 Devart. All Rights Reserved.

5.11.1.22.2.1 **ConnectionLifetime Property**

Used to specify the maximum time during which an opened connection can be used by connection pool.

**Class**

[TPoolingOptions](#)

**Syntax**
property ConnectionLifetime: integer default DefValConnectionLifetime;

Remarks
Use the ConnectionLifeTime property to specify the maximum time during which an opened connection can be used by connection pool. Measured in milliseconds. Pool deletes connections with exceeded connection lifetime when TCustomDACConnection is about to close. If the ConnectionLifetime property is set to 0 (by default), then the lifetime of connection is infinity. ConnectionLifetime concerns only inactive connections in the pool.

5.11.1.22.2.2 MaxPoolSize Property

Used to specify the maximum number of connections that can be opened in connection pool.

Class
TPoolingOptions

Syntax

property MaxPoolSize: integer default DefValMaxPoolSize;

Remarks
Specifies the maximum number of connections that can be opened in connection pool. Once this value is reached, no more connections are opened. The valid values are 1 and higher.

5.11.1.22.2.3 MinPoolSize Property

Used to specify the minimum number of connections that can be opened in the connection pool.

Class
TPoolingOptions

Syntax
property MinPoolSize: integer default DefValMinPoolSize;

Remarks
Use the MinPoolSize property to specify the minimum number of connections that can be opened in the connection pool.

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 TCustomDAConnection component.

5.11.1.23 TSmartFetchOptions Class

Smart fetch options are used to set up the behavior of the SmartFetch mode.

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

Unit
DBAccess

Syntax
TSmartFetchOptions = class(TPersistent);
### Properties of the **TSmartFetchOptions** class.

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

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enabled</strong></td>
<td>Sets SmartFetch mode enabled or not.</td>
</tr>
<tr>
<td><strong>LiveBlock</strong></td>
<td>Used to minimize memory consumption.</td>
</tr>
<tr>
<td><strong>PrefetchedFields</strong></td>
<td>List of fields additional to key fields that will be read from the database on dataset open.</td>
</tr>
<tr>
<td><strong>SQLGetKeyValues</strong></td>
<td>SQL query for the read key and prefetched fields from the database.</td>
</tr>
</tbody>
</table>
and prefetched fields from the database.

See Also
- TSmartFetchOptions Class
- TSmartFetchOptions Class Members

5.11.1.23.2.1 Enabled Property

Sets SmartFetch mode enabled or not.

Class
TSmartFetchOptions

Syntax

```property``
```Enabled``
```Boolean``
```default``
```False``

5.11.1.23.2.2 LiveBlock Property

Used to minimize memory consumption.

Class
TSmartFetchOptions

Syntax

```property``
```LiveBlock``
```Boolean``
```default``
```True``

Remarks

If LiveBlock is True, then on navigating through a dataset forward or backward, memory will be allocated for records count defined in the FetchRows property, and no additional memory will be allocated. But if you return records that were read from the database before, they will be read from the database again, because when you left block with these records, memory was free. So the LiveBlock mode minimizes memory consumption, but can
decrease performance, because it can lead to repeated data reading from the database.

The default value of LiveBlock is False.

5.11.1.23.2.3 PrefetchedFields Property

List of fields additional to key fields that will be read from the database on dataset open.

Class

TSmartFetchOptions

Syntax

property PrefetchedFields: string;

Remarks

If you are going to use locate, filter or sort by some fields, then these fields should be added to the prefetched fields list to avoid excessive reading from the database.

5.11.1.23.4 SQLGetKeyValues Property

SQL query for the read key and prefetched fields from the database.

Class

TSmartFetchOptions

Syntax

property SQLGetKeyValues: TStrings;

Remarks

SQLGetKeyValues is used when the basic SQL query is complex and the query for reading the key and prefetched fields can't be generated automatically.
### 5.11.2 Types

Types in the **DBAccess** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TAfterExecuteEvent</strong></td>
<td>This type is used for the <strong>TCustomDADataSet.AfterExecute</strong> and <strong>TCustomDASQL.AfterExecute</strong> events.</td>
</tr>
<tr>
<td><strong>TAfterFetchEvent</strong></td>
<td>This type is used for the <strong>TCustomDADataSet.AfterFetch</strong> event.</td>
</tr>
<tr>
<td><strong>TBeforeFetchEvent</strong></td>
<td>This type is used for the <strong>TCustomDADataSet.BeforeFetch</strong> event.</td>
</tr>
<tr>
<td><strong>TConnectionLostEvent</strong></td>
<td>This type is used for the <strong>TCustomDAConnection.OnConnectionLost</strong> event.</td>
</tr>
<tr>
<td><strong>TDACreationErrorEvent</strong></td>
<td>This type is used for the <strong>TCustomDAConnection.OnCreationError</strong> event.</td>
</tr>
<tr>
<td><strong>TDATransactionErrorEvent</strong></td>
<td>This type is used for the <strong>TDATransaction.onError</strong> event.</td>
</tr>
<tr>
<td><strong>TRefreshOptions</strong></td>
<td>Represents the set of <strong>TRefreshOption</strong>.</td>
</tr>
<tr>
<td><strong>TUpdateExecuteEvent</strong></td>
<td>This type is used for the <strong>TCustomDADataSet.AfterUpdateExecute</strong> and <strong>TCustomDADataSet.BeforeUpdateExecute</strong> events.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback

### 5.11.2.1 TAfterExecuteEvent Procedure Reference

This type is used for the **TCustomDADataSet.AfterExecute** and **TCustomDASQL.AfterExecute** events.
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.

**Parameters**

```
TAfterFetchEvent = procedure (DataSet: TCustomDADataSet) of object;
```

- **DataSet**
  - Holds the TCustomDADataSet descendant to synchronize the record position with.

**Parameters**

```
TBeforeFetchEvent = procedure (DataSet: TCustomDADataSet) of object;
```

- **DataSet**
  - Holds the TCustomDADataSet descendant to synchronize the record position with.
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.

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.

TDACOnnectionErrorEvent = procedure (Sender: TObject; var ErrorCode: Integer) of object;

Parameters

- **Sender**

- **ErrorCode**
  - The error code.
**DBAccess**

**Syntax**

```plaintext
TDACConnectionErrorEvent = procedure (Sender: TObject; E: EDAError; var Fail: boolean) of object;
```

**Parameters**

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

**E**
The error information.

**Fail**
False, if an error dialog should be prevented from being displayed and EAbort exception should be raised to cancel current operation.

---

**5.11.2.6 TDATransactionErrorEvent Procedure Reference**

This type is used for the `TDATransaction.OnError` event.

**Unit**

`DBAccess`

**Syntax**

```plaintext
TDATransactionErrorEvent = procedure (Sender: TObject; E: EDAError; var Fail: boolean) of object;
```

**Parameters**

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

**E**
The error code.

**Fail**
False, if an error dialog should be prevented from being displayed and EAbort exception to cancel the current operation should be raised.
5.11.2.7  **TRefreshOptions Set**

Represents the set of **TRefreshOption**.

**Unit**

**DBAccess**

**Syntax**

```
TRefreshOptions = set of TRefreshOption;
```

5.11.2.8  **TUpdateExecuteEvent Procedure Reference**

This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.

**Unit**

**DBAccess**

**Syntax**

```
TUpdateExecuteEvent = procedure (Sender: TDataSet; StatementTypes: TStatementTypes; Params: TDAParams) of object;
```

**Parameters**

- **Sender**
  - An object that raised the event.
- **StatementTypes**
  - Holds the type of the SQL statement being executed.
- **Params**
  - Holds the parameters with which the SQL statement will be executed.

© 1997-2021

Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.11.3  **Enumerations**

Enumerations in the **DBAccess** unit.

**Enumerations**

© 2021 Devart
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLabelSet</td>
<td>Sets the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>TLockMode</td>
<td>Specifies the lock mode.</td>
</tr>
<tr>
<td>TRefreshOption</td>
<td>Indicates when the editing record will be refreshed.</td>
</tr>
<tr>
<td>TRetryMode</td>
<td>Specifies the application behavior when connection is lost.</td>
</tr>
</tbody>
</table>

### 5.11.3.1 TLabelSet Enumeration

Sets the language of labels in the connect dialog.

**Unit**

`DBAccess`

**Syntax**

```plaintext
TLabelSet = (lsCustom, lsEnglish, lsFrench, lsGerman, lsItalian, lsPolish, lsPortuguese, lsRussian, lsSpanish);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsCustom</td>
<td>Set the language of labels in the connect dialog manually.</td>
</tr>
<tr>
<td>lsEnglish</td>
<td>Set English as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsFrench</td>
<td>Set French as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsGerman</td>
<td>Set German as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsItalian</td>
<td>Set Italian as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsPolish</td>
<td>Set Polish as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsPortuguese</td>
<td>Set Portuguese as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsRussian</td>
<td>Set Russian as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsSpanish</td>
<td>Set Spanish as the language of labels in the connect dialog.</td>
</tr>
</tbody>
</table>
5.11.3.2 TLockMode Enumeration

Specifies the lock mode.

Unit

DBAccess

Syntax

TLockMode = (lmNone);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lmLockDelayed</td>
<td>Locking occurs when the user posts an edited record, then the lock is released. Locking is done by the RefreshRecord method. Corresponds to optimistic locking.</td>
</tr>
<tr>
<td>lmLockImmediate</td>
<td>Locking occurs when the user starts editing a record. The lock is released after the user has posted or canceled the changes. Corresponds to pessimistic locking.</td>
</tr>
<tr>
<td>lmNone</td>
<td>No locking occurs. This mode should only be used in single user applications. The default value.</td>
</tr>
</tbody>
</table>

5.11.3.3 TRefreshOption Enumeration

Indicates when the editing record will be refreshed.

Unit

DBAccess

Syntax

TRefreshOption = (roAfterInsert, roAfterUpdate, roBeforeEdit);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>roAfterInsert</td>
<td>Refresh is performed after inserting.</td>
</tr>
<tr>
<td>roAfterUpdate</td>
<td>Refresh is performed after updating.</td>
</tr>
<tr>
<td>roBeforeEdit</td>
<td>Refresh is performed by Edit method.</td>
</tr>
</tbody>
</table>
5.11.3.4 TRetryMode Enumeration

Specifies the application behavior when connection is lost.

Unit

DBAccess

Syntax

TRetryMode = (rmRaise, rmReconnect, rmReconnectExecute);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmRaise</td>
<td>An exception is raised.</td>
</tr>
<tr>
<td>rmReconnect</td>
<td>Reconnect is performed and then exception is raised.</td>
</tr>
<tr>
<td>rmReconnectExec</td>
<td>Reconnect is performed and abortive operation is reexecuted. Exception is not raised.</td>
</tr>
</tbody>
</table>

5.11.4 Variables

Variables in the DBAccess unit.

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>When set to True allows data access components to change screen cursor for the execution time.</td>
</tr>
</tbody>
</table>
5.11.4.1 ChangeCursor Variable

When set to True allows data access components to change screen cursor for the execution time.

Unit

DBAccess

Syntax

```
changeCursor: boolean = True;
```

5.12 IBC

This unit contains main components of IBDAC.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomIBCDataSet</td>
<td>A base class that defines InterBase functionality for a dataset.</td>
</tr>
<tr>
<td>TCustomIBCQuery</td>
<td>A base class for defining functionality for descendant classes which access database using SQL statements.</td>
</tr>
<tr>
<td>TCustomIBCTable</td>
<td>A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td>TIBCArray</td>
<td>A class representing the value of the InterBase array data type.</td>
</tr>
<tr>
<td>TIBCArrayField</td>
<td>A class encapsulating the fundamental behavior common to the InterBase array fields.</td>
</tr>
<tr>
<td>TIBConnection</td>
<td>A component for setting and controlling connections to an</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TIBCConnectionOptions</strong></td>
<td>This class allows setting up the behaviour of the TIBCConnection class.</td>
</tr>
<tr>
<td><strong>TIBCDatasetOptions</strong></td>
<td>This class allows setting up the behaviour of the TIBCDataset class.</td>
</tr>
<tr>
<td><strong>TIBCDatasource</strong></td>
<td>TIBCDatasource provides an interface between an IBDAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td><strong>TIBCdbKeyField</strong></td>
<td>A class representing the InterBase RDB$DB_KEY field.</td>
</tr>
<tr>
<td><strong>TIBCEncryptor</strong></td>
<td>The class that performs encrypting and decrypting of data.</td>
</tr>
<tr>
<td><strong>TIBCMetaData</strong></td>
<td>A component for obtaining metainformation about database objects from the server.</td>
</tr>
<tr>
<td><strong>TIBCPParam</strong></td>
<td>A class that is used to set the values of individual parameters passed with queries or stored procedures.</td>
</tr>
<tr>
<td><strong>TIBCPParams</strong></td>
<td>Used to control TIBCPParam objects.</td>
</tr>
<tr>
<td><strong>TIBCQuery</strong></td>
<td>A component for executing queries and operating record sets. It also provides flexible way to update data.</td>
</tr>
<tr>
<td><strong>TIBCSQL</strong></td>
<td>A component for executing SQL statements and calling stored procedures on the database server.</td>
</tr>
<tr>
<td><strong>TIBCSSLLConnectionOptions</strong></td>
<td>A class for setting up the SSL options.</td>
</tr>
<tr>
<td><strong>TIBCStoredProc</strong></td>
<td>A component for accessing and executing stored procedures and functions.</td>
</tr>
<tr>
<td><strong>TIBCTable</strong></td>
<td>A component for retrieving and updating data in a</td>
</tr>
</tbody>
</table>
single table without writing SQL statements.

**TIBCTransaction**
A component for managing transactions in an application.

**TIBCUpdateSQL**
A component for tuning update operations for the DataSet component.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCTransactionErrorEvent</td>
<td>This type is used for the TIBCTransaction.OnError event.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGeneratorMode</td>
<td>Specifies the method used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>TIBCProtocol</td>
<td>Specifies the network protocol of connection with InterBase server.</td>
</tr>
<tr>
<td>TIBCTransactionAction</td>
<td>Specifies the transaction behaviour when connection closes.</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>Holds pointers to all TIBCConnection objects of an application.</td>
</tr>
<tr>
<td>DefConnection</td>
<td>Read this variable to get pointer to default connection object. Same as DefaultConnection function.</td>
</tr>
<tr>
<td>UseDefConnection</td>
<td>When set to true enables TCustomIBCDataSet and TIBCSQL components to</td>
</tr>
</tbody>
</table>
use default connection if they are not attached to any connection.

### Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBDACVersion</td>
<td>Read this constant to get current version number for IBDAC.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

#### 5.12.1 Classes

Classes in the IBC unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomIBCDataSet</td>
<td>A base class that defines InterBase functionality for a dataset.</td>
</tr>
<tr>
<td>TCustomIBCQuery</td>
<td>A base class for defining functionality for descendant classes which access database using SQL statements.</td>
</tr>
<tr>
<td>TCustomIBCTable</td>
<td>A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td>TIBCArray</td>
<td>A class representing the value of the InterBase array data type.</td>
</tr>
<tr>
<td>TIBCArrayField</td>
<td>A class encapsulating the fundamental behavior common to the InterBase array fields.</td>
</tr>
<tr>
<td>TIBCConnection</td>
<td>A component for setting and controlling connections to an</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TIBCConnectionOptions</td>
<td>This class allows setting up the behaviour of the TIBCConnection class.</td>
</tr>
<tr>
<td>TIBCDatasetOptions</td>
<td>This class allows setting up the behaviour of the TIBCDataset class.</td>
</tr>
<tr>
<td>TIBCDataSource</td>
<td>TIBCDataSource provides an interface between an IBDAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TIBCDbKeyField</td>
<td>A class representing the InterBase RDB$DB_KEY field.</td>
</tr>
<tr>
<td>TIBCEncryptor</td>
<td>The class that performs encrypting and decrypting of data.</td>
</tr>
<tr>
<td>TIBCMetaData</td>
<td>A component for obtaining metainformation about database objects from the server.</td>
</tr>
<tr>
<td>TIBCParam</td>
<td>A class that is used to set the values of individual parameters passed with queries or stored procedures.</td>
</tr>
<tr>
<td>TIBCParams</td>
<td>Used to control TIBCParam objects.</td>
</tr>
<tr>
<td>TIBCQuery</td>
<td>A component for executing queries and operating record sets. It also provides flexible way to update data.</td>
</tr>
<tr>
<td>TIBCSQL</td>
<td>A component for executing SQL statements and calling stored procedures on the database server.</td>
</tr>
<tr>
<td>TIBCSSLConnectionOptions</td>
<td>A class for setting up the SSL options.</td>
</tr>
<tr>
<td>TIBCStoredProc</td>
<td>A component for accessing and executing stored procedures and functions.</td>
</tr>
<tr>
<td>TIBCTable</td>
<td>A component for retrieving and updating data in a</td>
</tr>
</tbody>
</table>
5.12.1.1 TCustomIBCDataSet Class

A base class that defines InterBase functionality for a dataset.

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

Unit

IBC

Syntax

TCustomIBCDataSet = class(TCustomDADataSet);

Remarks

TCustomIBCDataSet is a component that defines InterBase functionality for a dataset. TCustomIBCDataSet can execute queries, fetch rows and controls InterBase specific data types. Applications never use TCustomIBCDataSet objects directly. Instead they use the descendants of TCustomIBCDataSet, such as TIBCQuery, TIBCStoredProc and TIBCTable, which inherit its database-related properties and methods.

TIBCQuery provides insert, delete and update operations on records by dynamically generated SQL statements. It uses TCustomDADataSet.KeyFields property to build SQL statements for the SQLDelete, SQLInsert and SQLUpdate properties if they were empty before updating the database.

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
  TCustomIBCDataSet
See Also
- TCustomDADataSet.KeyFields
- TIBCQuery
- TIBCStoredProc
- TIBCTable

© 1997-2021
Devart. All Rights Reserved.

5.12.1.1.1 Members

**TCustomIBCDataSet** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoCommit</strong></td>
<td>Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td><strong>BaseSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td><strong>Cursor</strong></td>
<td>Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to set data type mapping rules</td>
</tr>
</tbody>
</table>
| **Debug** (inherited from **TCustomDADataSet**) | Used to display the statement that is being executed and the values and
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DetailFields</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong></td>
<td>Used to refresh record by the RETURNING clause when insert is performed.</td>
</tr>
<tr>
<td><strong>Encryption</strong></td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>FetchAll</strong></td>
<td>Used to retrieve all records in a dataset.</td>
</tr>
<tr>
<td><strong>FetchRows</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td><strong>GeneratorMode</strong></td>
<td>Used to specify which method is used internally to generate a sequenced field.</td>
</tr>
<tr>
<td><strong>GeneratorStep</strong></td>
<td>Used to set the increment for increasing or decreasing current generator value when using the automatic key field value generation feature.</td>
</tr>
<tr>
<td><strong>Handle</strong></td>
<td>Used to specify the handle for the SQL statement of TCustomIBCDataSet.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
</tbody>
</table>
| **IsQuery**       | Used to check if the SQL
statement returns rows.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeyGenerator</td>
<td>Used 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.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>LockMode</td>
<td>Used to indicate when to perform a locking of editing record.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of the TCustomIBCDataSet object.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>(inherited from TCustomDADataSet) Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>(inherited from TCustomDADataSet) Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>(inherited from TCustomDADataSet) Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Plan</strong></td>
<td>Used to get or set the PLAN clause of the SELECT statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from TMemDataSet) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from TMemDataSet) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>(inherited from TCustomDADataSet) Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>(inherited from TCustomDADataSet) Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>(inherited from TCustomDADataSet) Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsDeleted</strong></td>
<td>Used to indicate the number of rows that were deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsFetched</strong></td>
<td>Used to get the number of the currently fetched rows.</td>
</tr>
<tr>
<td><strong>RowsInserted</strong></td>
<td>Used to indicate the number of rows that were inserted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsUpdated</strong></td>
<td>Used to indicate the number of rows that were updated during the last query operation.</td>
</tr>
<tr>
<td><strong>SmartFetch</strong></td>
<td>The SmartFetch mode is used for fast navigation.</td>
</tr>
</tbody>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th><strong>Property</strong></th>
<th><strong>Description</strong></th>
<th><strong>Inheritance</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL (inherited from TCustomDADataSet)</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SQLDelete (inherited from TCustomDADataSet)</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SQLInsert (inherited from TCustomDADataSet)</td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SQLLock (inherited from TCustomDADataSet)</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SQLRecCount (inherited from TCustomDADataSet)</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SQLRefresh (inherited from TCustomDADataSet)</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SQLType</td>
<td>Used to get the typecode of the SQL statement being processed by the InterBase database server.</td>
<td></td>
</tr>
<tr>
<td>SQLUpdate (inherited from TCustomDADataSet)</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Transaction</td>
<td>Used to determine the transaction under which the query of this dataset executes.</td>
<td></td>
</tr>
<tr>
<td>UniDirectional (inherited from TCustomDADataSet)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
<td>TCustomDADataSet</td>
</tr>
</tbody>
</table>

through a huge amount of records and to minimize memory consumption.
**UpdateObject**

Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.

**UpdateRecordTypes** (inherited from `TMemDataSet`)

Used to indicate the update status for the current record when cached updates are enabled.

**UpdatesPending** (inherited from `TMemDataSet`)

Used to check the status of the cached updates buffer.

**UpdateTransaction**

Used to get or set the transaction for modifying a dataset.

---

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td><strong>ApplyRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CreateProcCall</td>
<td>Assigns PL/SQL block that calls stored procedure to the SQL property.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from TMemDataSet)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong> (inherited from TCustomDADataset)</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>Execute</strong> (inherited from TCustomDADataset)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong> (inherited from TCustomDADataset)</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong> (inherited from TCustomDADataset)</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong> (inherited from TCustomDADataset)</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong> (inherited from TCustomDADataset)</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td><strong>FindKey</strong> (inherited from TCustomDADataset)</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong> (inherited from TCustomDADataset)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</strong> (inherited from TCustomDADataset)</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>GetArray</strong></td>
<td>Retrieves a TIBCArray object for a field when only its name is known.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>Retrieves a TIBCBlob object for a field when only its name is known.</td>
</tr>
<tr>
<td><strong>GetDataType</strong> (inherited from TCustomDADataSet)</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldObject</strong> (inherited from TCustomDADataSet)</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong> (inherited from TCustomDADataSet)</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><strong>GetFieldScale</strong> (inherited from TCustomDADataSet)</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong> (inherited from TCustomDADataSet)</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td><strong>GetOrderBy</strong> (inherited from TCustomDADataSet)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td><strong>GotoCurrent</strong> (inherited from TCustomDADataSet)</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td><strong>Lock</strong> (inherited from TCustomDADataSet)</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td><strong>MacroByName</strong> (inherited from TCustomDADataSet)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Called to set or use parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from TCustomDADataSet)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SaveSQL</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SetOrderBy</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SetRange</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SQLSaved</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>UnLock</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>TMemDataSet</td>
</tr>
</tbody>
</table>
### Properties

**Properties of the TCustomIBCDataSet class.**

For a complete list of the TCustomIBCDataSet class members, see the TCustomIBCDataSet Members topic.
## Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCommit</td>
<td>Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td>BaseSQL (inherited from TCustomDADataSet)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions (inherited from TCustomDADataSet)</td>
<td>Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td>Cursor</td>
<td>Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.</td>
</tr>
<tr>
<td>DataTypeMap (inherited from TCustomDADataSet)</td>
<td>Used to set data type mapping rules.</td>
</tr>
<tr>
<td>Debug (inherited from TCustomDADataSet)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DetailFields (inherited from TCustomDADataSet)</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Disconnected (inherited from TCustomDADataSet)</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>DMLRefresh</td>
<td>Used to refresh record by the RETURNING clause when insert is performed.</td>
</tr>
<tr>
<td>Encryption</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FetchAll</td>
<td>Used to retrieve all records in a dataset.</td>
</tr>
<tr>
<td>FetchRows (inherited from TCustomDADataSet)</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL (inherited from TCustomDADataSet)</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL (inherited from TCustomDADataSet)</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>GeneratorMode</td>
<td>Used to specify which method is used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>GeneratorStep</td>
<td>Used to set the increment for increasing or decreasing current generator value when using the automatic key field value generation feature.</td>
</tr>
<tr>
<td>Handle</td>
<td>Used to specify the handle for the SQL statement of TCustomIBCDataSet.</td>
</tr>
<tr>
<td>IndexFieldNames (inherited from TMemDataSet)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery</td>
<td>Used to check if the SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive (inherited from TMemDataSet)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields (inherited from TCustomDADataSet)</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeyGenerator</td>
<td>Used 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.</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LockMode</td>
<td></td>
</tr>
<tr>
<td>MacroCount</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Macros</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>MasterFields</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>MasterSource</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td>ParamCheck</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>ParamCount</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Params</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Plan</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Prepared</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>RowsDeleted</td>
<td>Used to indicate the number of rows that were deleted during the last query operation.</td>
</tr>
<tr>
<td>RowsFetched</td>
<td>Used to get the number of the currently fetched rows.</td>
</tr>
<tr>
<td>RowsInserted</td>
<td>Used to indicate the number of rows that were inserted during the last query operation.</td>
</tr>
<tr>
<td>RowsUpdated</td>
<td>Used to indicate the number of rows that were updated during the last query operation.</td>
</tr>
<tr>
<td>SmartFetch</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>Used to specify the SQL statement that will be used when applying an insertion</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLLock</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>(inherited from TCustomDADataSet) Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLType</td>
<td>Used to get the typecode of the SQL statement being processed by the InterBase database server.</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>Transaction</td>
<td>Used to determine the transaction under which the query of this dataset executes.</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>(inherited from TCustomDADataSet) Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>(inherited from TMemDataSet) Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending</td>
<td>(inherited from TMemDataSet) Used to check the status of the cached updates buffer.</td>
</tr>
<tr>
<td>UpdateTransaction</td>
<td>Used to get or set the transaction for modifying a dataset.</td>
</tr>
</tbody>
</table>
5.12.1.1.2.1  AutoCommit Property

Used to automatically commit each update, insert or delete statement by database server.

Class

```
TCustomIBCDataSet
```

Syntax

```
property AutoCommit: boolean;
```

Remarks

When True and Connection.AutoCommit is True, each update, insert or delete statement is automatically committed by database server.

5.12.1.1.2.2  Connection Property

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

Class

```
TCustomIBCDataSet
```

Syntax

```
property Connection: TIBCConnection;
```

Remarks

Use the Connection property to specify the connection in which the dataset will be executed. If connection is not connected, the Open method calls Connection.Connect.
5.12.1.1.2.3 Cursor Property

Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.

Class

TCustomIBCDataSet

Syntax

```plaintext
property Cursor: string;
```

Remarks

Use the Cursor property for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause. FetchRows property must be set to 1 for using cursor.

5.12.1.1.2.4 DMLRefresh Property

Used to refresh record by the RETURNING clause when insert is performed.

Class

TCustomIBCDataSet

Syntax

```plaintext
property DMLRefresh: boolean;
```

Remarks

Use the DMLRefresh property to refresh record by the RETURNING clause when insert is performed. This feature is only for Firebird 2.0 and higher.
The default value is False.

**Note:** When the DMLRefresh property is set to True, the value of `TCustomDADataSet.RefreshOptions` is ignored to avoid refetching field values from the server.

### See Also
- Updating Data with IBDAC Dataset Components
- `TCustomDADataSet.RefreshOptions`

---

5.12.1.1.2.5 Encryption Property

**Used to specify encryption options in a dataset.**

**Class**

`TCustomIBCDataSet`

**Syntax**

```
property Encryption: TIBCEncryption;
```

**Remarks**

Set the Encryption options for using encryption in a dataset.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.1.2.6 FetchAll Property

**Used to retrieve all records in a dataset.**

**Class**

`TCustomIBCDataSet`

**Syntax**

```
property FetchAll: boolean;
```
5.12.1.1.2.7 GeneratorMode Property

Used to specify which method is used internally to generate a sequenced field.

Class

TCustomIBCDataSet

Syntax

property GeneratorMode: TGeneratorMode default gmPost;

Remarks

Set the GeneratorMode property to specify which method is used internally to generate a sequenced field.

See Also

• KeyGenerator

5.12.1.1.2.8 GeneratorStep Property

Used to set the increment for increasing or decreasing current generator value when using the automatic key field value generation feature.

Class

TCustomIBCDataSet

Syntax

property GeneratorStep: integer default 1;

Remarks

Use the GeneratorStep property to set the increment for increasing or decreasing current generator value when using the automatic key field value generation feature. The default value is 1.
Reserved.

5.12.1.2.9 Handle Property

Used to specify the handle for the SQL statement of TCustomIBCDataSet.

Class

TCustomIBCDataSet

Syntax

`property` Handle: TISC_STMT_HANDLE;

Remarks

Use the Handle property to specify the handle for the SQL statement of TCustomIBCDataSet.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.2.10 IsQuery Property

Used to check if the SQL statement returns rows.

Class

TCustomIBCDataSet

Syntax

`property` IsQuery: boolean;

Remarks

When the TCustomIBCDataSet component is prepared, returns True, if the SQL statement is a SELECT block that returns the REF CURSOR parameter.

Use the IsQuery property to check whether the SQL statement returns rows or not. TCustomIBCDataSet returns rows when the SQL statement is the SELECT or PL/SQL block with the REF CURSOR parameter. TCustomIBCDataSet must be prepared before.

IsQuery is a read-only property.

© 1997-2021 Devart. All Rights Reserved.
5.12.1.1.2.11 KeyGenerator Property

Used 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.

Class

TCustomIBCDataSet

Syntax

```property KeyGenerator: string;```

Remarks

Use the KeyGenerator property 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.

Note: KeyGenerator is used by TCustomIBCDataset only if TCustomDADataset.KeyFields property is assigned.

See Also

- GeneratorMode

© 1997-2021 Devart. All Rights Reserved.

5.12.1.1.2.12 LockMode Property

Used to indicate when to perform a locking of editing record.

Class

TCustomIBCDataSet

Syntax

```property LockMode: TLockMode default lmNone;```

Remarks

Use the LockMode property to define when to perform locking of an editing 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 realizes through execution of the SELECT FOR UPDATE statement in Firebird 1.5 and through the UPDATE operation in other database...
5.12.1.2.13  Options Property

Used to specify the behaviour of the TCustomIBCDataSet object.

Class

TCustomIBCDataSet

Syntax

property Options: TIBCDataSetOptions;

Remarks

Set the properties of Options to specify behaviour of a TCustomIBCDataSet object.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoClose</td>
<td>Used to for CustomIBCDataSet to close cursor after fetching all rows.</td>
</tr>
<tr>
<td>BooleanDomainFields</td>
<td>Used to create TBooleanField for fields that have domain of the integer data type, and the domain name contains 'BOOLEAN'.</td>
</tr>
<tr>
<td>CacheArrays</td>
<td>Used to allocate local memory buffer to hold a copy of the array content.</td>
</tr>
<tr>
<td>CacheBlobs</td>
<td>Used to allocate local memory buffer to hold a copy of the BLOB content.</td>
</tr>
<tr>
<td>ComplexArrayFields</td>
<td>Used to store array fields as TIBCArrayField objects.</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used for TCustomIBCDataSet to fill the DefaultExpression property of TField objects by the appropriate value.</td>
</tr>
<tr>
<td>DeferredArrayRead</td>
<td>Used for fetching all InterBase array values</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DeferredBlobRead</td>
<td>Used for fetching all InterBase BLOB values when they are explicitly requested.</td>
</tr>
<tr>
<td>DescribeParams</td>
<td>Used to specify whether to query TIBCParam properties from the server when executing the TCustomDADataSet.Prepare method.</td>
</tr>
<tr>
<td>FieldsAsString</td>
<td>Used to treat all non-BLOB fields as being of string datatype.</td>
</tr>
<tr>
<td>FullRefresh</td>
<td>Used to refresh fields from all tables of the query.</td>
</tr>
<tr>
<td>PrepareUpdateSQL</td>
<td>Used to automatically prepare update queries before execution.</td>
</tr>
<tr>
<td>QueryRowsAffected</td>
<td>Used to increase the performance of update operations.</td>
</tr>
<tr>
<td>SetDomainNames</td>
<td>Enables setting the TIBCFieldDesc.DomainName property for fields</td>
</tr>
<tr>
<td>SetEmptyStrToNull</td>
<td>Force replace of empty strings with NULL values in data. The default value is False.</td>
</tr>
<tr>
<td>StreamedBlobs</td>
<td>Used to save all edited BLOBs as streamed and to handle the streamed</td>
</tr>
<tr>
<td>StrictUpdate</td>
<td>Used for TCustomIBCDataSet to raise an exception when the number of the updated or deleted records is not equal 1.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDADataSet.Options
- BLOB Data Types

5.12.1.1.2.14 Plan Property

Used to get or set the PLAN clause of the SELECT statement.

Class
TCustomIBCDataSet

Syntax

```property Plan: string;```
Remarks
Use the Plan property to get or set the PLAN clause of the SELECT statement.

5.12.1.1.2.15 RowsDeleted Property

Used to indicate the number of rows that were deleted during the last query operation.

Class
TCustomIBCDataSet

Syntax

```delphi
property RowsDeleted: integer;
```

Remarks
Check RowsDeleted to determine how many rows were deleted during the last query operation. If RowsDeleted is -1, the query has not deleted any rows.

5.12.1.1.2.16 RowsFetched Property

Used to get the number of the currently fetched rows.

Class
TCustomIBCDataSet

Syntax

```delphi
property RowsFetched: integer;
```

Remarks
Use the RowsFetched property to get the number of the currently fetched rows.
5.12.1.1.2.17 RowsInserted Property

Used to indicate the number of rows that were inserted during the last query operation.

Class

TCustomIBCDataset

Syntax

\texttt{property} \ RowsInserted: \ integer;

Remarks

Check RowsInserted to determine how many rows were inserted during the last query operation. If RowsInserted is -1, the query has not inserted any rows.

5.12.1.1.2.18 RowsUpdated Property

Used to indicate the number of rows that were updated during the last query operation.

Class

TCustomIBCDataset

Syntax

\texttt{property} \ RowsUpdated: \ integer;

Remarks

Check RowsUpdated to determine how many rows were updated during the last query operation. If RowsUpdated is -1, the query has not updated any rows.

5.12.1.1.2.19 SmartFetch Property

The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.

Class
5.12.1.1.2.20 SQLType Property

Used to get the typecode of the SQL statement being processed by the InterBase database server.

Class

TCustomIBCDataSet

Syntax

```
property SQLType: integer;
```

Remarks

Read the SQLType property to get the typecode of the SQL statement being processed by the InterBase database server.

5.12.1.1.2.21 Transaction Property

Used to determine the transaction under which the query of this dataset executes.

Class

TCustomIBCDataSet

Syntax

```
property Transaction: TIBCTransaction stored IsTransactionStored;
```
Remarks
Use the Transaction property to determine the transaction under which the query of this dataset executes. You can separately set transaction for executing modifying queries with the `UpdateTransaction` property. By default the Transaction and the UpdateTransaction properties are the same.

See Also
• UpdateTransaction

5.12.1.1.2.22  UpdateObject Property

Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.

Class
TCustomIBCDataSet

Syntax

```pascal
property UpdateObject: TIBCUpdateSQL;
```

Remarks
The UpdateObject property specifies an update object component which provides SQL statements that perform updates of the read-only datasets when cached updates are enabled.

5.12.1.1.2.23  UpdateTransaction Property

Used to get or set the transaction for modifying a dataset.

Class
TCustomIBCDataSet

Syntax
**property** UpdateTransaction: TIBCTransaction;

Remarks

Use the UpdateTransaction property to set or get the transaction for modifying a dataset. By default UpdateTransaction is the same as Transaction.

See Also

- Transaction

© 1997-2021 Devart. All Rights Reserved.

5.12.1.1.3 Methods

Methods of the TCustomIBCDataSet class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from TCustomDADataset)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec (inherited from TCustomDADataset)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td><strong>CreateProcCall</strong></td>
<td>Assigns PL/SQL block that calls stored procedure to the SQL property.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from <strong>TMemDataset</strong>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from <strong>TMemDataset</strong>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from <strong>TMemDataset</strong>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>Execute</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td><strong>FindKey</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FindParam</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>GetArray</td>
<td>Retrieves a TIBCArr object for a field when only its name is known.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>Retrieves a TIBCBlob object for a field when only its name is known.</td>
</tr>
<tr>
<td>GetDataType (inherited from TCustomDADataset)</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td>GetFieldObject (inherited from TCustomDADataset)</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td>GetFieldPrecision (inherited from TCustomDADataset)</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>GetFieldScale (inherited from TCustomDADataset)</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames (inherited from TCustomDADataset)</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy (inherited from TCustomDADataset)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GotoCurrent (inherited from TCustomDADataset)</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td>Locate (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx (inherited from TMemDataSet)</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Lock (inherited from TCustomDADataset)</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td>MacroByName (inherited from TCustomDADataset)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Called to set or use parameter information for a</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td><strong>UnLock</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Frees the resources allocated for a previously</td>
</tr>
</tbody>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

**See Also**

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

---

### 5.12.1.1.3.1 CreateProcCall Method

Assigns PL/SQL block that calls stored procedure to the SQL property.

#### Class

**TCustomIBCDataSet**

#### Syntax

```delphi
procedure CreateProcCall(const Name: string);
```

#### Parameters

- `Name`  
  Holds the name of the stored procedure

#### Remarks

Call the CreateProcCall method to assign PL/SQL block that calls stored procedure specified by Name to the SQL property. Retrieves the information about parameters of the procedure from InterBase. After calling CreateProcCall you can execute stored procedure by Execute method.

#### See Also
5.12.1.1.3.2 FindParam Method

Determines if a parameter with the specified name exists in a dataset.

Class

TCustomIBCDataSet

Syntax

function FindParam(const Value: string): TIBCFParam;

Parameters

Value

Holds the name of the param for which to search.

Return Value

The TIBCFParam object for the specified Name, if a param with a matching name was found. Nil otherwise.

Remarks

Call the FindParam method to determine if a parameter with the specified name 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 the TIBCFParam object for the specified Name. Otherwise it returns nil.

See Also

• TCustomDADataSet.Params
• ParamByName

© 1997-2021
Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback
5.12.1.1.3.3 GetArray Method

Retrieves a TIBCArray object for a field when only its name is known.

**Class**

`TCustomIBCDataset`

**Syntax**

```pascal
function GetArray(FieldDesc: TFieldDesc): TIBCArray;
overload;
function GetArray(const FieldName: string): TIBCArray;
overload;
```

**Parameters**

*FieldName*

Holds the field name of an existing field.

**Return Value**

The TIBCArray object, if a field with a matching name was found.

**Remarks**

Call the GetArray method to retrieve a TIBCArray object for a field when only its name is known. FieldName is the name of an existing field. The field should have the ftIBCArray type.

**See Also**

- [TIBCArray](#)

© 1997-2021 Devart. All Rights Reserved.

5.12.1.1.3.4 GetBlob Method

Retrieves a TIBCBlob object for a field when only its name is known.

**Class**

`TCustomIBCDataset`

**Syntax**

```pascal
function GetBlob(const FieldName: string): TIBCBlob;
```

**Parameters**

*FieldName*

© 2021 Devart
Holds the field name of an existing field.

**Return Value**

The TIBCBlob object, if a field with a matching name was found.

**Remarks**

Call the GetBlob method to retrieve a TIBCBlob object for a field when only its name is known. FieldName is the name of an existing field. The field should have the ftIBCBlob type.

**See Also**

- TIBCBlob

© 1997-2021 Devart. All Rights Reserved.
5.12.1.2 TCustomIBCQuery Class

A base class for defining functionality for descendant classes which access database using SQL statements.

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

Unit
IBC

Syntax

TCustomIBCQuery = class(TCustomIBCDataSet);

Remarks

TCustomIBCQuery is a base class that defines functionality for descendant classes which access database using SQL statements. Applications never use the TCustomIBCQuery objects directly. Instead they use descendants of TCustomIBCQuery, such as TIBCQuery, TIBCStoredProc and TIBCTable.

TCustomIBCQuery implements functionality to update database tables using DML SQL statements. Put SQL statements into SQLInsert, SQLDelete, SQLUpdate properties. There is no restriction on their syntax, so any SQL statements are allowed. Usually you need to use INSERT, DELETE and UPDATE statements but also you can use stored procedures in more diverse cases.

SQLInsert, SQLDelete, SQLUpdate, SQLLock, SQLRefresh properties support automatic binding of parameters which have names identical to fields captions. To retrieve the value of a field as it was before operation use the field name with 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement. Use TCustomDADataset.BeforeUpdateExecute event to assign value to additional parameters and TCustomDADataset.AfterUpdateExecute event for reading them.

TCustomIBCQuery is read-only when none of SQLInsert, SQLDelete, SQLUpdate properties are defined.

Inheritance Hierarchy
See Also
- **TCustomIBCDataSet**
- **TIBCQuery**
- **TIBCStoredProc**
- **TIBCTable**

© 1997-2021 Devart. All Rights Reserved.

5.12.1.2.1 Members

**TCustomIBCQuery** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoCommit</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td><strong>BaseSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td><strong>Cursor</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> (inherited from <strong>TCustomDADatSet</strong>)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong> (inherited from <strong>TCustomDADatSet</strong>)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong> (inherited from <strong>TCustomDADatSet</strong>)</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong> (inherited from <strong>TCustomDADatSet</strong>)</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to refresh record by the RETURNING clause when insert is performed.</td>
</tr>
<tr>
<td><strong>Encryption</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>FetchAll</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to retrieve all records in a dataset.</td>
</tr>
<tr>
<td><strong>FetchRows</strong> (inherited from <strong>TCustomDADatSet</strong>)</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong> (inherited from <strong>TCustomDADatSet</strong>)</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong> (inherited from <strong>TCustomDADatSet</strong>)</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td><strong>GeneratorMode</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to specify which method is used internally to generate a sequenced field.</td>
</tr>
<tr>
<td><strong>GeneratorStep</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to set the increment for increasing or decreasing current generator value when using the automatic key field value generation feature.</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to specify the handle</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>(inherited from TMemDataSet) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery</td>
<td>(inherited from TCustomIBCDataSet) Used to check if the SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from TMemDataSet) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>(inherited from TCustomDADataSet) Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeyGenerator</td>
<td>(inherited from TCustomIBCDataSet) Used 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.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>(inherited from TMemDataSet) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>(inherited from TMemDataSet) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>LockMode</td>
<td>(inherited from TCustomIBCDataSet) Used to indicate when to perform a locking of editing record.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>(inherited from TCustomDADataSet) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>(inherited from TCustomDADataSet) Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>(inherited from TCustomDADataSet) Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong> (inherited from TCustomDADataset)</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong> (inherited from TCustomIBCDataset)</td>
<td>Used to specify the behaviour of the TCustomIBCDataset object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong> (inherited from TCustomDADataset)</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong> (inherited from TCustomDADataset)</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from TCustomDADataset)</td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Plan</strong> (inherited from TCustomIBCDataset)</td>
<td>Used to get or set the PLAN clause of the SELECT statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from TMemDataSet)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from TMemDataSet)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong> (inherited from TCustomDADataset)</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong> (inherited from TCustomDADataset)</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong> (inherited from TCustomDADataset)</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsDeleted</strong> (inherited from TCustomIBCDataset)</td>
<td>Used to indicate the number of rows that were deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsFetched</strong> (inherited from TCustomIBCDataset)</td>
<td>Used to get the number of the currently fetched rows.</td>
</tr>
<tr>
<td><strong>RowsInserted</strong> (inherited from TCustomIBCDataset)</td>
<td>Used to indicate the number of rows that were inserted during the last query.</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td><strong>RowsUpdated</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
<tr>
<td><strong>SmartFetch</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>SQLType</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
<tr>
<td><strong>SQLUpdate</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>Transaction</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
</tbody>
</table>
query of this dataset executes.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Inherited From</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniDirectional</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>UpdateObject</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>UpdatesPending</td>
<td>Used to check the status of the cached updates buffer.</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>UpdateTransaction</td>
<td>Used to get or set the transaction for modifying a dataset.</td>
<td>TCustomIBCDataSet</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Inherited From</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>ApplyRange</td>
<td>Applies a range to the dataset.</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>BreakExec</td>
<td>Breaks execution of the SQL statement on the server.</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>CancelRange</td>
<td>Removes any ranges currently in effect for a dataset.</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>Clears the cached updates buffer.</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td>TCustomDADataSet</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>CreateProcCall</td>
<td>TCustomIBCDataSet</td>
<td>Assigns PL/SQL block that calls stored procedure to the SQL property.</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>TMemDataSet</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>TCustomDADataSet</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>TMemDataSet</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>TMemDataSet</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>Execute</td>
<td>TCustomDADataSet</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>TCustomDADataSet</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched</td>
<td>TCustomDADataSet</td>
<td>Used to find out whether TCustomDADataSet has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>TCustomDADataSet</td>
<td>Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>TCustomDADataSet</td>
<td>Used to learn whether TCustomDADataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>TCustomDADataSet</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>TCustomDADataSet</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>TCustomDADataSet</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the specified value.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>FindParam</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
<td></td>
</tr>
<tr>
<td><strong>GetArray</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Retrieves a TIBCArray object for a field when only its name is known.</td>
<td></td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Retrieves a TIBCBlob object for a field when only its name is known.</td>
<td></td>
</tr>
<tr>
<td><strong>GetDataType</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
<td></td>
</tr>
<tr>
<td><strong>GetFieldObject</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Returns a multireference shared object from field.</td>
<td></td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Retrieves the precision of a number field.</td>
<td></td>
</tr>
<tr>
<td><strong>GetFieldScale</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Retrieves the scale of a number field.</td>
<td></td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Provides a list of available key field names.</td>
<td></td>
</tr>
<tr>
<td><strong>GetOrderBy</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
<td></td>
</tr>
<tr>
<td><strong>GotoCurrent</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
<td></td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
<td></td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Excludes features that don't need to be included to the <strong>TMemDataSet.Locate</strong> method of <strong>TDataSet</strong>.</td>
<td></td>
</tr>
<tr>
<td><strong>Lock</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Locks the current record.</td>
<td></td>
</tr>
<tr>
<td><strong>MacroByName</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Finds a macro with the specified name.</td>
<td></td>
</tr>
<tr>
<td><strong>ParamByName</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Called to set or use parameter information for a</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td><code>TCustomDADataSet</code></td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong></td>
<td><code>TCustomDADataSet</code></td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong></td>
<td><code>TCustomDADataSet</code></td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td><code>TMemDataSet</code></td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td><code>TMemDataSet</code></td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong></td>
<td><code>TCustomDADataSet</code></td>
<td>Saves the SQL property value to <code>BaseSQL</code>.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td><code>TMemDataSet</code></td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td><code>TCustomDADataSet</code></td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td><code>TMemDataSet</code></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong></td>
<td><code>TMemDataSet</code></td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong></td>
<td><code>TMemDataSet</code></td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong></td>
<td><code>TCustomDADataSet</code></td>
<td>Determines if the SQL property value was saved to the <code>BaseSQL</code> property.</td>
</tr>
<tr>
<td><strong>UnLock</strong></td>
<td><code>TCustomDADataSet</code></td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td><code>TMemDataSet</code></td>
<td>Frees the resources allocated for a previously prepared cursor.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>UpdateResult</strong> (inherited from TMemDataSet)**</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
<td></td>
</tr>
<tr>
<td><strong>UpdateStatus</strong> (inherited from TMemDataSet)**</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
<td></td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong> (inherited from TCustomDADataset)**</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td><strong>AfterFetch</strong> (inherited from TCustomDADataset)**</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong> (inherited from TCustomDADataset)**</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong> (inherited from TCustomDADataset)**</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong> (inherited from TCustomDADataset)**</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td><strong>OnUpdateError</strong> (inherited from TMemDataSet)**</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td><strong>OnUpdateRecord</strong> (inherited from TMemDataSet)**</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>
table without writing SQL statements.

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

Unit

IBC

Syntax

TCustomIBCTable = class(TCustomIBCQuery);

Remarks

TCustomIBCTable component is inherited from TCustomIBCQuery class. Applications never use TCustomIBCTable objects directly. Instead they use descendants of TCustomIBCTable such as TIBCTable.

It allows to retrieve and update data in single table without writing SQL statements. Use TableName to specify the name of a table. TIBCTable uses KeyFields property to build SQL statements for updating table data. KeyFields is a string containing a semicolon-delimited list of field names. If KeyFields is not defined before opening, TIBCTable uses primary or unique key.

Inheritance Hierarchy

TMemDataSet
  TCustomDADataset
    TCustomIBCDataset
      TCustomIBCQuery
        TCustomIBCTable

See Also

- TIBCTable
- TCustomIBCDataset
- TIBCQuery
- Master/Detail Relationships
- Updating Data with IBDAC Dataset Components

© 1997-2021
Devart. All Rights Reserved.
5.12.1.3.1 Members

**TCustomIBCTable** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoCommit</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td><strong>BaseSQL</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td><strong>Cursor</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) Used to refresh record by the RETURNING clause</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>Encryption</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>Exists</td>
<td></td>
</tr>
<tr>
<td>FetchAll</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>FetchRows</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GeneratorMode</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>GeneratorStep</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>Handle</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>IsQuery</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>KeyFields</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeyGenerator</td>
<td>Used 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.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>LockMode</td>
<td>Used to indicate when to perform a locking of editing record.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of the TCustomIBCDataSet object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Used to indicate how many parameters are there in the</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Plan</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) Used to get or set the PLAN clause of the SELECT statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsDeleted</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) Used to indicate the number of rows that were deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsFetched</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) Used to get the number of the currently fetched rows.</td>
</tr>
<tr>
<td><strong>RowsInserted</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) Used to indicate the number of rows that were inserted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsUpdated</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) Used to indicate the number of rows that were updated during the last query operation.</td>
</tr>
<tr>
<td><strong>SmartFetch</strong></td>
<td>(inherited from <strong>TCustomIBCDataSet</strong>) The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLType</td>
<td>Used to get the typecode of the SQL statement being processed by the InterBase database server.</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>Transaction</td>
<td>Used to determine the transaction under which the query of this dataset executes.</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>Used to indicate the update status for the current record when cached updates are</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong></td>
<td>(inherited from <a href="#">TMemDataSet</a>) Used to check the status of the cached updates buffer.</td>
</tr>
<tr>
<td><strong>UpdateTransaction</strong></td>
<td>(inherited from <a href="#">TCustomIBCDataSet</a>) Used to get or set the transaction for modifying a dataset.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</strong></td>
<td>(inherited from <a href="#">TCustomDADataSet</a>) Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td><strong>ApplyRange</strong></td>
<td>(inherited from <a href="#">TMemDataSet</a>) Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>(inherited from <a href="#">TMemDataSet</a>) Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td>(inherited from <a href="#">TCustomDADataSet</a>) Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>(inherited from <a href="#">TMemDataSet</a>) Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>(inherited from <a href="#">TMemDataSet</a>) Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>(inherited from <a href="#">TMemDataSet</a>) Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong></td>
<td>(inherited from <a href="#">TCustomDADataSet</a>) Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td><strong>CreateProcCall</strong></td>
<td>(inherited from <a href="#">TCustomIBCDataSet</a>) Assigns PL/SQL block that calls stored procedure to the SQL property.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>(inherited from <a href="#">TMemDataSet</a>) Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteTable</strong></td>
<td>Deletes a table from a database.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td>(inherited from <a href="#">TCustomDADataSet</a>) Removes WHERE clause from the SQL property and...</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>EmptyTable</strong></td>
<td>Truncates the current table content on the server.</td>
</tr>
<tr>
<td><strong>Execute</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td><strong>FindKey</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td><strong>FindParam</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td><strong>GetArray</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Retrieves a TIBCArray object for a field when only its name is known.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Retrieves a TIBCBlob object for a field when only its name is known.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>GetDataType</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldObject</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><strong>GetFieldScale</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td><strong>GetOrderBy</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td><strong>GotoCurrent</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td><strong>Lock</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td><strong>MacroByName</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong> (inherited from <strong>TCustomIBCDataset</strong>)</td>
<td>Called to set or use parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td><strong>UnLock</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Indicates the current update status for the dataset when</td>
</tr>
</tbody>
</table>
cached updates are enabled.

## Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  

5.12.1.3.2 Properties

Properties of the `TCustomIBCTable` class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCommit</td>
<td>Used to automatically commit each update, insert</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BaseSQL</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions</td>
<td>Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td>Cursor</td>
<td>Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.</td>
</tr>
<tr>
<td>DataTypeMap</td>
<td>Used to set data type mapping rules.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DetailFields</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>DMLRefresh</td>
<td>Used to refresh record by the RETURNING clause when insert is performed.</td>
</tr>
<tr>
<td>Encryption</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td>Exists</td>
<td>Indicates whether a table with the name passed in TableName exists in the database.</td>
</tr>
<tr>
<td>FetchAll</td>
<td>Used to retrieve all records.</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>FetchRows</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GeneratorMode</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>GeneratorStep</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>Handle</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>IsQuery</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>KeyFields</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>KeyGenerator</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong> (inherited from TMemDataSet)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong> (inherited from TMemDataSet)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>LockMode</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used to indicate when to perform a locking of editing record.</td>
</tr>
<tr>
<td><strong>MacroCount</strong> (inherited from TCustomDADataSet)</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong> (inherited from TCustomDADataSet)</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong> (inherited from TCustomDADataSet)</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong> (inherited from TCustomDADataSet)</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used to specify the behaviour of the TCustomIBCDataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong> (inherited from TCustomDADataSet)</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong> (inherited from TCustomDADataSet)</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from TCustomDADataSet)</td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Plan</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used to get or set the PLAN clause of the SELECT statement.</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td><em>TMemDataSet</em></td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td><em>TMemDataSet</em></td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td><em>TCustomDADataSet</em></td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td><em>TCustomDADataSet</em></td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td><em>TCustomDADataSet</em></td>
</tr>
<tr>
<td><strong>RowsDeleted</strong></td>
<td><em>TCustomIBCDataSet</em></td>
</tr>
<tr>
<td><strong>RowsFetched</strong></td>
<td><em>TCustomIBCDataSet</em></td>
</tr>
<tr>
<td><strong>RowsInserted</strong></td>
<td><em>TCustomIBCDataSet</em></td>
</tr>
<tr>
<td><strong>RowsUpdated</strong></td>
<td><em>TCustomIBCDataSet</em></td>
</tr>
<tr>
<td><strong>SmartFetch</strong></td>
<td><em>TCustomIBCDataSet</em></td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td><em>TCustomDADataSet</em></td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td><em>TCustomDADataSet</em></td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td><em>TCustomDADataSet</em></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>SQLLock</strong> (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong> (inherited from TCustomDADataset)</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong> (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td><strong>SQLType</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used to get the typecode of the SQL statement being processed by the InterBase database server.</td>
</tr>
<tr>
<td><strong>SQLUpdate</strong> (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td><strong>Transaction</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used to determine the transaction under which the query of this dataset executes.</td>
</tr>
<tr>
<td><strong>UniDirectional</strong> (inherited from TCustomDADataset)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td><strong>UpdateObject</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong> (inherited from TMemDataSet)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong> (inherited from TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
<tr>
<td><strong>UpdateTransaction</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used to get or set the transaction for modifying a dataset.</td>
</tr>
</tbody>
</table>
5.12.1.3.2.1 Exists Property

Indicates whether a table with the name passed in TableName exists in the database.

Class

TCustomIBCTable

Syntax

```pascal
property Exists: Boolean;
```

Remarks

Use the Exists property to determine whether a table with the name passed in TableName exists in the database.

5.12.1.3.3 Methods

Methods of the TCustomIBCTable class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from TCustomDADataset)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong> (inherited from TCustomDADataset)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong> (inherited from TCustomDADataset)</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td><strong>CreateProcCall</strong> (inherited from TCustomIBCDataset)</td>
<td>Assigns PL/SQL block that calls stored procedure to the SQL property.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from TMemDataSet)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteTable</strong></td>
<td>Deletes a table from a database.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong> (inherited from TCustomDADataset)</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>EmptyTable</strong></td>
<td>Truncates the current table content on the server.</td>
</tr>
<tr>
<td><strong>Execute</strong> (inherited from TCustomDADataset)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong> (inherited from TCustomDADataset)</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Fetched</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Fetching</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FindKey</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FindMacro</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FindNearest</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FindParam</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>GetArray</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>GetBlob</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>GetDataType</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>GotoCurrent</strong></td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td><strong>Locate</strong></td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong></td>
<td>Overloaded. Excludes features that don’t need to be included to the <code>TMemDataSet.Locate</code> method of <code>TDataSet</code>.</td>
</tr>
<tr>
<td><strong>Lock</strong></td>
<td>Locks the current record.</td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Called to set or use parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong></td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong></td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong></td>
<td>Saves the SQL property value to <code>BaseSQL</code>.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td>Method</td>
<td>Inheritance</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>SetRange</td>
<td>(inherited from TMemDataSet)</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>(inherited from TMemDataSet)</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>(inherited from TMemDataSet)</td>
</tr>
<tr>
<td>SQLSaved</td>
<td>(inherited from TCustomDADataset)</td>
</tr>
<tr>
<td>UnLock</td>
<td>(inherited from TCustomDADataset)</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>(inherited from TMemDataSet)</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>(inherited from TMemDataSet)</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>(inherited from TMemDataSet)</td>
</tr>
</tbody>
</table>

See Also
- TCustomIBCTable Class
- TCustomIBCTable Class Members

5.12.1.3.3.1 DeleteTable Method

Deletes a table from a database.
**TCustomIBCTable**

**Syntax**

```delphi
procedure DeleteTable;
```

**Remarks**

Call the `DeleteTable` method to delete a table from the database.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.3.3.2 EmptyTable Method

Truncates the current table content on the server.

**Class**

**TCustomIBCTable**

**Syntax**

```delphi
procedure EmptyTable;
```

**Remarks**

Call the `EmptyTable` method to truncate the current table content on the server.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.4 TIBCArray Class

A class representing the value of the InterBase array data type.

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

**Unit**

**IBC**

**Syntax**

```delphi
TIBCArray = class(TCustomIBCArray);
```
Remarks

TIBCArray represents the value of the InterBase array data type. You can get a TIBCArray object by the TCustomIBCDataSet.GetArray method after fetching rows contained in an array field. You should explicitly add the T:Devart.IbDac.Units.IBCArray unit to 'uses' list to use the TIBCArray class.

Inheritance Hierarchy

TSharedObject
  TDBObject
    TCustomIBCArray
      TIBCArray

See Also

- TCustomIBCDataSet.GetArray
- TIBCParam.AsArray

© 1997-2021
Devart. All Rights Reserved.

5.12.1.4.1 Members

**TIBCArray** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ArrayDimensions</strong> (inherited from TCustomIBCArray)</td>
<td>Contains the array dimensions count.</td>
</tr>
<tr>
<td><strong>ArrayHighBound</strong> (inherited from TCustomIBCArray)</td>
<td>Used to get or set the upper boundary of the defined dimension subscript.</td>
</tr>
<tr>
<td><strong>ArrayID</strong> (inherited from TCustomIBCArray)</td>
<td>Contains the array ID.</td>
</tr>
<tr>
<td><strong>ArrayLowBound</strong> (inherited from TCustomIBCArray)</td>
<td>Used to get or set the lower boundary of the defined dimension subscript.</td>
</tr>
<tr>
<td><strong>ArraySize</strong> (inherited from TCustomIBCArray)</td>
<td>Used to determine the size of the whole array data in bytes.</td>
</tr>
<tr>
<td><strong>AsString</strong> (inherited from TCustomIBCArray)</td>
<td>Used to return array as string.</td>
</tr>
<tr>
<td>Member</td>
<td>Inherited From</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>Cached</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>CachedDimensions</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>CachedHighBound</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>CachedLowBound</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>CachedSize</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>ColumnName</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>DbHandle</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>IsNull</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>Items</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>ItemScale</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>ItemSize</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>ItemType</td>
<td></td>
</tr>
<tr>
<td>Modified</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>RefCount</td>
<td>TSharedObject</td>
</tr>
<tr>
<td>TableName</td>
<td>TCustomIBCArray</td>
</tr>
<tr>
<td>TrHandle</td>
<td>TCustomIBCArray</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign (inherited from TCustomIBCArray)</td>
<td>Copies the Source object content to the current one.</td>
</tr>
<tr>
<td>ClearArray (inherited from TCustomIBCArray)</td>
<td>Clears all array values on the server if Cached is set to False.</td>
</tr>
<tr>
<td>CreateTemporaryArray (inherited from TCustomIBCArray)</td>
<td>Creates a temporary array on the InterBase server.</td>
</tr>
<tr>
<td>GetArrayInfo (inherited from TCustomIBCArray)</td>
<td>Used to get array descriptor.</td>
</tr>
<tr>
<td>GetItemAsDateTime (inherited from TCustomIBCArray)</td>
<td>Reads the value of an array item into an object or variable of the TDateTime type.</td>
</tr>
<tr>
<td>GetItemAsFloat (inherited from TCustomIBCArray)</td>
<td>Reads the value of an array item into a floating-point number.</td>
</tr>
<tr>
<td>GetItemAsInteger (inherited from TCustomIBCArray)</td>
<td>Reads the value of an array item into an integer.</td>
</tr>
<tr>
<td>GetItemAsSmallInt (inherited from TCustomIBCArray)</td>
<td>Reads the value of an array item into a short integer.</td>
</tr>
<tr>
<td>GetItemAsString (inherited from TCustomIBCArray)</td>
<td>Reads the value of an array item into a string.</td>
</tr>
<tr>
<td>GetItemAsWideString (inherited from TCustomIBCArray)</td>
<td>Reads the value of an array item into a WideString.</td>
</tr>
<tr>
<td>GetItemsSlice (inherited from TCustomIBCArray)</td>
<td>Returns the array slice items' values.</td>
</tr>
<tr>
<td>GetItemValue (inherited from TCustomIBCArray)</td>
<td>Returns the array item value.</td>
</tr>
<tr>
<td>ReadArray (inherited from TCustomIBCArray)</td>
<td>Reads an array from the database to memory.</td>
</tr>
<tr>
<td>ReadArrayItem (inherited from TCustomIBCArray)</td>
<td>Reads the array item specified by indices from the database to memory.</td>
</tr>
</tbody>
</table>
### Properties of the `TIBCArray` class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArrayDimensions</code></td>
<td>Contains the array dimensions count.</td>
</tr>
<tr>
<td><code>ArrayHighBound</code></td>
<td>Used to get or set the upper boundary of the defined</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ArrayID</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Contains the array ID.</td>
</tr>
<tr>
<td><strong>ArrayLowBound</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to get or set the lower boundary of the defined dimension subscript.</td>
</tr>
<tr>
<td><strong>ArraySize</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to determine the size of the whole array data in bytes.</td>
</tr>
<tr>
<td><strong>AsString</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to return array as string.</td>
</tr>
<tr>
<td><strong>Cached</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Indicates whether to cache array data on the client side.</td>
</tr>
<tr>
<td><strong>CachedDimensions</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Contains cached array dimensions count.</td>
</tr>
<tr>
<td><strong>CachedHighBound</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to get the upper boundary of defined dimension subscript of cached array elements.</td>
</tr>
<tr>
<td><strong>CachedLowBound</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to get the lower boundary of the defined dimension subscript of cached array elements.</td>
</tr>
<tr>
<td><strong>CachedSize</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to get the cached array data size in bytes.</td>
</tr>
<tr>
<td><strong>ColumnName</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to get or set the name of the table column that has array type.</td>
</tr>
<tr>
<td><strong>DbHandle</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Contains the handle of a database where the array is stored.</td>
</tr>
<tr>
<td><strong>IsNull</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to define whether the array field in the database is null.</td>
</tr>
<tr>
<td><strong>Items</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to get or set array items.</td>
</tr>
<tr>
<td><strong>ItemScale</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Used to get or set the scale for array items for the NUMERIC and DECIMAL datatypes.</td>
</tr>
<tr>
<td><strong>ItemSize</strong></td>
<td>(inherited from <strong>TCustomIBCArray</strong>) Contains the size of an array item.</td>
</tr>
<tr>
<td><strong>ItemType</strong></td>
<td>Used to return the type of an array element.</td>
</tr>
</tbody>
</table>
Modified (inherited from TCustomIBCArray) | Used to indicate if the modifications done in cache were saved to the database.
---|---
RefCount (inherited from TSharedObject) | Used to return the count of reference to a TSharedObject object.
TableName (inherited from TCustomIBCArray) | Used to set the name of the table containing an array field.
TrHandle (inherited from TCustomIBCArray) | Contains the handle of the transaction in which the array is read or written.

See Also
- TIBCArray Class
- TIBCArray Class Members

5.12.1.4.2.1 ItemType Property

Used to return the type of an array element.

Class
TIBCArray

Syntax

```property
type ItemType: TFieldType;
```

Remarks

Read the ItemType property to return the type of an array element.

5.12.1.5 TIBCArrayField Class

A class encapsulating the fundamental behavior common to the InterBase array fields.

For a list of all members of this type, see TIBCArrayField members.
Unit

IBC

Syntax

\[ \text{TIBCArrayField} = \textbf{class} (\text{TField}); \]

Remarks

TIBCArrayField encapsulates the fundamental behavior common to the InterBase array fields. The InterBase array fields can contain multiple data items of the same datatypes. TIBCArrayField introduces new AsArray property for accessing array data.

See Also

- TIBCArray

© 1997-2021 Devart. All Rights Reserved.

5.12.1.5.1 Members

TIBCArrayField class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsArray</td>
<td>Used to specify the value of the field when it represents the value of the InterBase array type.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.12.1.5.2 Properties

Properties of the TIBCArrayField class.

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

Public
### AsArray Property

**Used to specify the value of the field when it represents the value of the InterBase array type.**

#### Class

**TIBCArryField**

#### Syntax

```delphi
property AsArray: TIBCArry;
```

#### Remarks

Use the AsArray property to specify the value of the field when it represents the value of the InterBase array type.

#### TIBCConnection Class

A component for setting and controlling connections to an InterBase database.

For a list of all members of this type, see **TIBCConnection** members.

#### Unit

**IBC**

#### Syntax
TIBCConnection = class(TCustomDAConnection);

Remarks
The TIBCConnection component is used to maintain connection to the InterBase database. After setting the Username, Password, and Database properties, you can establish a connection to the database by calling the Open method or setting the Connected property to True.

The TIBCConnection component contains internal transaction component that is accessible through the TIBCConnection.DefaultTransaction property. It is possible to create applications without manual adding TIBCTransaction components. But you can use external transaction components instead of internal transaction. To do it create a TIBCTransaction component and assign the TIBCConnection.DefaultTransaction property to this transaction. If you need to restore internal transaction - just reset the TIBCConnection.DefaultTransaction property to nil.

All components which are dedicated to perform data access, such as TIBCQuery, TIBCSQL, TIBCScript, must have their Connection property assigned with one of the TIBCConnection instances.

Inheritance Hierarchy
TCustomDAConnection
   TIBCConnection

See Also
- TIBCConnection.DefaultTransaction
- TCustomIBCDataSet.Connection
- TIBCSQL.Connection

© 1997-2021 Devart. All Rights Reserved.

5.12.1.6.1 Members

TIBCConnection class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCommit</td>
<td>Used to permit or prevent permanent updates,</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>ClientLibrary</td>
<td>Used to set or get the client library location.</td>
</tr>
<tr>
<td>ConnectDialog</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td>Connected</td>
<td>Used to establish a database connection.</td>
</tr>
<tr>
<td>ConnectPrompt</td>
<td>Used to provide login support.</td>
</tr>
<tr>
<td>ConnectString</td>
<td>Used to specify the connection information, such as: UserName, Password, Server, etc.</td>
</tr>
<tr>
<td>ConvertEOL</td>
<td>Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td>Database</td>
<td>Used to set the name of the database to associate with TIBConnection component.</td>
</tr>
<tr>
<td>DatabaseInfo</td>
<td>Used to get information about the connected database.</td>
</tr>
<tr>
<td>DBSQLDialect</td>
<td>Shows the database SQL dialect.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display SQL statements being executed with their parameter values and data types.</td>
</tr>
<tr>
<td>DefaultTransaction</td>
<td>Used to access default database connection transaction.</td>
</tr>
<tr>
<td>Handle</td>
<td>Used to make calls directly to the InterBase API.</td>
</tr>
<tr>
<td>InTransaction</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td>LastError</td>
<td>Used to get the error code which resulted from the previous call to the InterBase server.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from <strong>TCustomDACConnection</strong>)</td>
<td>Specifies whether a login dialog appears immediately before opening a new connection.</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of the TIBCConnection object.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to specify the connection parameters.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Used to specify the password for a connection.</td>
</tr>
<tr>
<td><strong>Pooling</strong> (inherited from <strong>TCustomDACConnection</strong>)</td>
<td>Enables or disables using connection pool.</td>
</tr>
<tr>
<td><strong>PoolingOptions</strong> (inherited from <strong>TCustomDACConnection</strong>)</td>
<td>Specifies the behaviour of connection pool.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Used to specify the port number for connections to the server.</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Used to supply the server name to handle server's request for a login.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to execute any SQL statement.</td>
</tr>
<tr>
<td><strong>SQLDialect</strong></td>
<td>Used to set or return SQL Dialect used by InterBase client.</td>
</tr>
<tr>
<td><strong>SSLOptions</strong></td>
<td>Used to set up the SSL options.</td>
</tr>
<tr>
<td><strong>TransactionCount</strong></td>
<td>Used to return the number of transactions currently associated with this TIBCConnection component.</td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td>Used to specify a transaction for the given index.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Used to provide a user name.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddTransaction</td>
<td>Associates a TIBCTransaction component with the database component.</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td>AssignConnect</td>
<td>Shares database connection between the TIBCCConnection components.</td>
</tr>
<tr>
<td>Commit</td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td>CommitRetaining</td>
<td>Stores to the database server all changes of data associated with the default database transaction permanently and then retains the transaction context.</td>
</tr>
<tr>
<td>Connect</td>
<td>Establishes a connection to the server.</td>
</tr>
<tr>
<td>CreateDatabase</td>
<td>Creates an InterBase database.</td>
</tr>
<tr>
<td>CreateSQL</td>
<td>Creates a component for queries execution.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Performs disconnect.</td>
</tr>
<tr>
<td>DropDatabase</td>
<td>Used to drop database and remove database file from server.</td>
</tr>
<tr>
<td>ExecProc</td>
<td>Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td>ExecProcEx</td>
<td>Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td>ExecSQL</td>
<td>Executes a SQL statement with parameters.</td>
</tr>
<tr>
<td>ExecSQLEx</td>
<td>Executes any SQL statement outside the TQuery or TSQl components.</td>
</tr>
<tr>
<td>FindDefaultTransaction</td>
<td>Returns the default transaction for the database.</td>
</tr>
<tr>
<td>GetDatabaseNames</td>
<td>Returns a database list from the server.</td>
</tr>
</tbody>
</table>
**TCustomDAConnection**

GetKeyFieldNames (inherited from TCustomDAConnection)  Provides a list of available key field names.

GetStoredProcNames (inherited from TCustomDAConnection)  Returns a list of stored procedures from the server.

GetTableNames (inherited from TCustomDAConnection)  Provides a list of available tables names.

MonitorMessage (inherited from TCustomDAConnection)  Sends a specified message through the TCustomDASQLMonitor component.

ParamByName  Provides access to the OUT parameters and their values after processing SQL statement.

Ping (inherited from TCustomDAConnection)  Used to check state of connection to the server.

RemoveFromPool (inherited from TCustomDAConnection)  Marks the connection that should not be returned to the pool after disconnect.

RemoveTransaction  Disassociates a specified transaction from the database.

Rollback (inherited from TCustomDAConnection)  Discards all current data changes and ends transaction.

RollbackRetaining  Rolls back all changes of data associated with the default database transaction to the database server and then retains the transaction context.

StartTransaction (inherited from TCustomDAConnection)  Begins a new user transaction.

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnectionLost (inherited from TCustomDAConnection)</td>
<td>This event occurs when connection was lost.</td>
</tr>
</tbody>
</table>
### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ConnectDialog</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td><strong>ConnectString</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Used to specify the connection information, such as: UserName, Password, Server, etc.</td>
</tr>
<tr>
<td><strong>ConvertEOL</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td><strong>DatabaseInfo</strong></td>
<td>Used to get information about the connected database.</td>
</tr>
<tr>
<td><strong>DBSQLDialect</strong></td>
<td>Shows the database SQL dialect.</td>
</tr>
<tr>
<td><strong>Handle</strong></td>
<td>Used to make calls directly to the InterBase API.</td>
</tr>
<tr>
<td><strong>InTransaction</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td><strong>LastError</strong></td>
<td>Used to get the error code which resulted from the previous call to the InterBase server.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Specifies whether a login dialog appears immediately</td>
</tr>
</tbody>
</table>
before opening a new connection.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pooling</strong> (inherited from <strong>TCustomDACConnection</strong>)</td>
<td>Enables or disables using connection pool.</td>
</tr>
<tr>
<td><strong>PoolingOptions</strong> (inherited from <strong>TCustomDACConnection</strong>)</td>
<td>Specifies the behaviour of connection pool.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to execute any SQL statement.</td>
</tr>
<tr>
<td><strong>TransactionCount</strong></td>
<td>Used to return the number of transactions currently associated with this TIBCCConnection component.</td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td>Used to specify a transaction for the given index.</td>
</tr>
</tbody>
</table>

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoCommit</strong></td>
<td>Used to permit or prevent permanent updates, insertions, and deletions of data associated with the current transaction against the database server.</td>
</tr>
<tr>
<td><strong>ClientLibrary</strong></td>
<td>Used to set or get the client library location.</td>
</tr>
<tr>
<td><strong>Connected</strong></td>
<td>Used to establish a database connection.</td>
</tr>
<tr>
<td><strong>ConnectPrompt</strong></td>
<td>Used to provide login support.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Used to set the name of the database to associate with TIBCCConnection component.</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>Used to display SQL statements being executed with their parameter values and data types.</td>
</tr>
<tr>
<td><strong>DefaultTransaction</strong></td>
<td>Used to access default database connection transaction.</td>
</tr>
</tbody>
</table>
### Options

**Used to specify the behaviour of the TIBCConnection object.**

#### Params

**Used to specify the connection parameters.**

#### Password

**Used to specify the password for a connection.**

#### Port

**Used to specify the port number for connections to the server.**

#### Server

**Used to supply the server name to handle server's request for a login.**

#### SQLDialect

**Used to set or return SQL Dialect used by InterBase client.**

#### SSLOptions

**Used to set up the SSL options.**

#### Username

**Used to provide a user name.**

### See Also

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

5.12.1.6.2.1  AutoCommit Property

**Used to permit or prevent permanent updates, insertions, and deletions of data associated with the current transaction against the database server.**

### Class

**TIBConnection**

### Syntax

```pascal
property AutoCommit: boolean;
```

### Remarks

Use the AutoCommit property to permit or prevent permanent updates, insertions, and
deletions of data associated with the current transaction 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.

AutoCommit property in TIBConnection has higher precedence over the same properties in dataset components.

The default value is True.

Note: The AutoCommit property in TIBConnection globally specifies whether all queries to modify database are implicitly committed or not. Components which descend from TCustomDADataset and TCustomDASQL classes inherit their AutoCommit properties. This allows them to selectively specify their implicit transaction committing behavior after each data modifying access.

See Also
- TCustomDACConnection.Commit

5.12.1.6.2.2 ClientLibrary Property

Used to set or get the client library location.

Class
- TIBConnection

Syntax

```
property ClientLibrary: string;
```

Remarks

Use the ClientLibrary property to set or get the client library location.
5.12.1.6.2.3 Connected Property

Used to establish a database connection.

Class

TIBCConnection

Syntax

property Connected stored IsConnectedStored;

Remarks

Indicates whether the database connection is active. Set this property to True to establish a database connection. Setting this property to False will close a connection.

See Also
• TCustomDACConnection.Connect
• TCustomDACConnection.Disconnect

5.12.1.6.2.4 ConnectPrompt Property

Used to provide login support.

Class

TIBCConnection

Syntax

property ConnectPrompt: boolean stored False default True;

Remarks

Set ConnectPrompt to True to provide login support when establishing a connection. When ConnectPrompt is True, a dialog appears to prompt a user for a name and a password.

When ConnectPrompt is False, an application must supply user name and password values programmatically.

Warning: Storing hard-coded user name and password entries as property values or in code...
for an OnLogin event handler can compromise the server security.

See Also

- Password
- Server
- Username

© 1997-2021 Devart. All Rights Reserved.

5.12.1.6.2.5 Database Property

Used to set the name of the database to associate with TIBCConnection component.

Class

TIBCConnection

Syntax

```property Database: string;```

Remarks

Use the Database property to set the name of the database to associate with TIBCConnection component. For local InterBase databases it is a filename. For remote databases use following syntax.

To connect to an InterBase database on a remote server using TCP/IP the syntax is `<server_name>:<filename>`.

To connect to an InterBase database on a remote server using NetBEUI, the syntax is: `\<server_name>\<filename>`.

To connect to an InterBase database on a remote server using SPX, the syntax is: `<server_name>@<filename>`.

**Note:** You can set connection parameters not only by Database property. You can use Server property to set server name and Options property to set connection protocol. If Database property contains local filename and server property is empty, it will be a local connection. If Server property is not empty, Server and Options properties will be used for the connection. If both Database and Server property contain server name and they are the same, server name and connection protocol will be taken from the Database property. If they
are not the same, the server name and connection protocol will be taken from Server and Options properties and entire string of Database property will be processed as a database name without server name. In that case you may encounter errors.

See Also
- Options
- Server

5.12.1.6.2.6 DatabaseInfo Property

Used to get information about the connected database.

Class
TIBCConnection

Syntax

```
property DatabaseInfo: TGDSDatabaseInfo;
```

Remarks

Use the DatabaseInfo property to get information about the connected database. You should explicitly add the T:Devart.IbDac.Units.IBCClasses unit to the 'uses' list to use this property.

See Also
- TGDSDatabaseInfo

5.12.1.6.2.7 DBSQLDialect Property

Shows the database SQL dialect.

Class
TIBCConnection

Syntax

**property** DBSQLDialect: Integer;

Remarks
The DBSQLDialect property is used to show the database SQL dialect.

See Also
- SQLDialect

5.12.1.6.2.8 Debug Property

Used to display SQL statements being executed with their parameter values and data types.

Class
_TIBCConnection_

Syntax

```
property Debug: boolean;
```

Remarks
Set the Debug property to True to display SQL statements being executed with their parameter values and data types.

**Note:** To use this property you should explicitly include unit IBDACVcl (IBDACClx under Linux) to your project.

5.12.1.6.2.9 DefaultTransaction Property

Used to access default database connection transaction.

Class
_TIBCConnection_

Syntax
### DefaultTransaction Property

**property** `property DefaultTransaction: TIBCTransaction;`

**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.

**See Also**

- [Transactions](#)

### Handle Property

Used to make calls directly to the InterBase API.

**Class**

`TIBCConnection`

**Syntax**

**property** `property Handle: TISC_DB_HANDLE;`

**Remarks**

Use the Handle property to make calls directly to the InterBase API. Many of the InterBase API functions require a database handle as an argument.

### LastError Property

Used to get the error code which resulted from the previous call to the InterBase server.

**Class**

`TIBCConnection`

**Syntax**

**property** `property LastError: integer;`
Remarks

Use the LastError property to get the error code which resulted from the previous call to the InterBase server.

Class

TIBCConnection

Syntax

property Options: TIBCConnectionOptions;

Remarks

Set properties of Options to specify the behaviour of a TIBCConnection object.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharLength</td>
<td>Used to specify the size in bytes of a single character.</td>
</tr>
<tr>
<td>Charset</td>
<td>Used to set character set that IBDAC uses to read and write character data.</td>
</tr>
<tr>
<td>EnableBCD</td>
<td>Used to enable currency type.</td>
</tr>
<tr>
<td>EnableFMTBCD</td>
<td>Used to enable using FMTBCD instead of float for large integer numbers to keep precision.</td>
</tr>
<tr>
<td>EnableMemos</td>
<td>Used to enable creating TMemoField and TWideMemoField for BLOB fields with subtype 1.</td>
</tr>
<tr>
<td>IPVersion</td>
<td>Used to specify Internet Protocol Version.</td>
</tr>
<tr>
<td>NoDBTriggers</td>
<td>Used to disable all database triggers.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Used to specify the Network protocol of connection with InterBase server.</td>
</tr>
<tr>
<td>Role</td>
<td>Used to specify the InterBase connection role.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TrustedAuthentication</td>
<td>Used to apply Windows &quot;Trusted User&quot; security for authenticating Firebird users.</td>
</tr>
<tr>
<td>UseUnicode</td>
<td>Used to enable or disable Unicode support.</td>
</tr>
</tbody>
</table>

See Also

- TCustomDAConnection.Options
- Unicode Character Data

Class

TIBCConnection

Syntax

```pascal
property Params: TStrings;
```

Remarks

Use the Params property to specify the connection parameters, such as user name, password, role etc. For example:

```plaintext
user_name=sysdba password=masterkey sql_role_name=role1 lc_cctype=WIN1252
```

However, note that TIBCConnection has separate Username and Password properties. Therefore, to specify a user name and a password for connection, you should use either Username and Password properties or Params, but not both.

If you are using the Params property for creating a new database with the CreateDatabase function, you should set params like it is shown below:

```plaintext
USER "SYSDBA" PASSWORD "masterkey" PAGE_SIZE 4096
```

See Also

- CreateDatabase

© 1997-2021

Devart. All Rights Reserved.
5.12.1.6.2.14  Password Property

Used to specify the password for a connection.

Class

TIBCConnection

Syntax

```
property Password: string;
```

Remarks

Use the Password property to specify a password for connection.

When property is being changed TIBCConnection calls the Disconnect method.

See Also

- Username
- Server
- TCustomDACConnection.Connect

© 1997-2021 Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback

5.12.1.6.2.15  Port Property

Used to specify the port number for connections to the server.

Class

TIBCConnection

Syntax

```
property Port: string;
```

Remarks

Use the Port property to specify the port number for connections to InterBase and Firebird.

© 1997-2021 Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback

© 2021 Devart
5.12.1.6.2.16 Server Property

Used to supply the server name to handle server's request for a login.

Class

TIBCConnection

Syntax

```property server: string;```

Remarks

Use the Server property to supply the server name to handle server's request for a login.

When property is being changed TIBCConnection calls the Disconnect method.

See Also

- Username
- Password
- TCustomDAConnection.Connect

5.12.1.6.2.17 SQL Property

Used to execute any SQL statement.

Class

TIBCConnection

Syntax

```property SQL: TIBCSQL;```

Remarks

You can use the embedded TIBCSQL object to execute any SQL statement.

See Also

- TIBCSQL
- TCustomDAConnection.ExecSQL
5.12.1.6.2.18 SQLDialect Property

Used to set or return SQL Dialect used by InterBase client.

Class

TIBCConnection

Syntax

```pascal
property SQLDialect: Integer default 3;
```

Remarks

Use the SQLDialect property 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 property will be downgraded to match the database SQL dialect.

See Also

- DBSQLDialect

5.12.1.6.2.19 SSLOptions Property

Used to set up the SSL options.

Class

TIBCConnection

Syntax

```pascal
property SSLOptions: TIBCSSLConnectionOptions;
```

Remarks
Use the SSLOptions property to set up the SSL options.

5.12.1.6.2.20 TransactionCount Property

Used to return the number of transactions currently associated with this TIBCConnection component.

Class

TIBCConnection

Syntax

```property TransactionCount: integer;```

Remarks

Use the TransactionCount property to return the number of transactions currently associated with this TIBCConnection component.

See Also

- Transactions

5.12.1.6.2.21 Transactions Property(Indexer)

Used to specify a transaction for the given index.

Class

TIBCConnection

Syntax

```property Transactions[Index: Integer]: TIBCTransaction;```

Parameters

- `Index`
  - Holds the specified index.
Remarks
Use the Transactions property to specify a transaction for the given index.

5.12.1.6.2.22 Username Property

Used to provide a user name.

Class
TIBCConnection

Syntax

```pascal
property Username: string;
```

Remarks
Use the Username property to supply the user name to handle server's request for a login.

When property is being changed TIBCConnection calls Disconnect method.

See Also
- Password
- Server
- TCustomDACConnection.Connect

Methods of the TIBCConnection class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 1997-2021 Devart. All Rights Reserved.  

© 2021 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddTransaction</strong></td>
<td>Associates a TIBCTransaction component with the database component.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from TCustomDACConnection)</td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td><strong>AssignConnect</strong></td>
<td>Shares database connection between the TIBCCConnection components.</td>
</tr>
<tr>
<td><strong>Commit</strong> (inherited from TCustomDACConnection)</td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td><strong>CommitRetaining</strong></td>
<td>Stores to the database server all changes of data associated with the default database transaction permanently and then retains the transaction context.</td>
</tr>
<tr>
<td><strong>Connect</strong> (inherited from TCustomDACConnection)</td>
<td>Establishes a connection to the server.</td>
</tr>
<tr>
<td><strong>CreateDatabase</strong></td>
<td>Creates an InterBase database.</td>
</tr>
<tr>
<td><strong>CreateSQL</strong> (inherited from TCustomDACConnection)</td>
<td>Creates a component for queries execution.</td>
</tr>
<tr>
<td><strong>Disconnect</strong> (inherited from TCustomDACConnection)</td>
<td>Performs disconnect.</td>
</tr>
<tr>
<td><strong>DropDatabase</strong></td>
<td>Used to drop database and remove database file from server.</td>
</tr>
<tr>
<td><strong>ExecProc</strong> (inherited from TCustomDACConnection)</td>
<td>Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td><strong>ExecProcEx</strong> (inherited from TCustomDACConnection)</td>
<td>Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td><strong>ExecSQL</strong> (inherited from TCustomDACConnection)</td>
<td>Executes a SQL statement with parameters.</td>
</tr>
<tr>
<td><strong>ExecSQLEX</strong> (inherited from TCustomDACConnection)</td>
<td>Executes any SQL statement outside the TQuery or TSQl components.</td>
</tr>
<tr>
<td><strong>FindDefaultTransaction</strong></td>
<td>Returns the default transaction for the database.</td>
</tr>
<tr>
<td><strong>GetDatabaseNames</strong> (inherited from TCustomDACConnection)</td>
<td>Returns a database list from the server.</td>
</tr>
</tbody>
</table>
**TCustomDAConnection**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetStoredProcNames</td>
<td>Returns a list of stored procedures from the server.</td>
</tr>
<tr>
<td>GetTableNames</td>
<td>Provides a list of available tables names.</td>
</tr>
<tr>
<td>MonitorMessage</td>
<td>Sends a specified message through the TCustomDASQLMonitor component.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Provides access to the OUT parameters and their values after processing SQL statement.</td>
</tr>
<tr>
<td>Ping</td>
<td>Used to check state of connection to the server.</td>
</tr>
<tr>
<td>RemoveFromPool</td>
<td>Marks the connection that should not be returned to the pool after disconnect.</td>
</tr>
<tr>
<td>RemoveTransaction</td>
<td>Disassociates a specified transaction from the database.</td>
</tr>
<tr>
<td>Rollback</td>
<td>Discards all current data changes and ends transaction.</td>
</tr>
<tr>
<td>RollbackRetaining</td>
<td>Rolls back all changes of data associated with the default database transaction to the database server and then retains the transaction context.</td>
</tr>
<tr>
<td>StartTransaction</td>
<td>Begins a new user transaction.</td>
</tr>
</tbody>
</table>

**See Also**
- TIBCConnection Class
- TIBCConnection Class Members

© 1997-2021  Request Support  DAC Forum  Provide Feedback
5.12.1.6.3.1 AddTransaction Method

Associates a TIBCTransaction component with the database component.

Class

TIBCConnection

Syntax

function AddTransaction(TR: TIBCTransaction): Integer;

Parameters

TR
Holds the transaction that is being added.

Return Value
the index of the associated transaction in the transaction list.

Remarks

Use the AddTransaction method to associate a TIBCTransaction component with the database component. Returns the index of the associated transaction in the transaction list.

See Also

• Transactions

5.12.1.6.3.2 AssignConnect Method

Shares database connection between the TIBCConnection components.

Class

TIBCConnection

Syntax

procedure AssignConnect(Source: TIBCConnection);

Parameters

Source
Holds the source connection.

Remarks
Call the AssignConnect method to share database connection between the TIBCConnection components.

AssignConnect assumes that Source parameter points to a preconnected database component and sets for this instance of TIBCConnection Connected property to True. Note that AssignConnect doesn't make any references to the Source database. So before disconnecting parent TIBCConnection call AssignConnect(Nil) or Disconnect method for all assigned databases.

See Also
- TCustomDACConnection.Connect

5.12.1.6.3.3 CommitRetaining Method

Stores to the database server all changes of data associated with the default database transaction permanently and then retains the transaction context.

Class
TIBCConnection

Syntax

```pascal
procedure CommitRetaining;
```

Remarks
Call the CommitRetaining method to store to the database server all changes of data associated with the default database transaction permanently and then retain the transaction context.

See Also
- DefaultTransaction
5.12.1.6.3.4 CreateDatabase Method

Creates an InterBase database.

Class
TIBCConnection

Syntax

procedure CreateDatabase;

Remarks
Call the CreateDatabase method to create an InterBase database using Params as the rest of the CREATE DATABASE statement.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.6.3.5 DropDatabase Method

Used to drop database and remove database file from server.

Class
TIBCConnection

Syntax

procedure DropDatabase;

Remarks
Call the DropDatabase method to drop database and remove database file from server.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.6.3.6 FindDefaultTransaction Method

Returns the default transaction for the database.

Class
TIBCConnection
Syntax

```pascal
function FindDefaultTransaction: TIBCTransaction;
```

**Return Value**
the default transaction for the database.

**Remarks**
Call the FindDefaultTransaction method to return the default transaction for the database.

---

5.12.1.6.3.7  ParamByName Method

POrvides access to the OUT parameters and their values after processing SQL statement.

**Class**
*TIBConnection*

**Syntax**

```pascal
function ParamByName(const Name: string): TIBCParam;
```

**Parameters**

- **Name**
  Holds the parameter name.

**Return Value**

- the corresponding parameter.

**Remarks**
Call the ParamByName method to get access to the OUT parameters and their values after processing SQL statement with ExecSQL or stored procedure with ExecProc. Name should be equal to the parameter name as it occurred in SQL statement.

Implicitly calls the *TIBCSQL.ParamByName* function of *TIBCSQL*.

**See Also**
- [SQL](#)
- [TCustomDACConnection.ExecSQL](#)
- [TCustomDACConnection.ExecSQLEx](#)
5.12.1.6.3.8 RemoveTransaction Method

Disassociates a specified transaction from the database.

Class

TIBCConnection

Syntax

```pascal
procedure RemoveTransaction(TR: TIBCTransaction);
```

Parameters

- **TR**

Remarks

Call the RemoveTransaction method to disassociate a specified transaction from the database.

5.12.1.6.3.9 RollbackRetaining Method

Rolls back all changes of data associated with the default database transaction to the database server and then retains the transaction context.

Class

TIBCConnection

Syntax

```pascal
procedure RollbackRetaining;
```

Remarks

Call the RollbackRetaining method to roll back all changes of data associated with the default database transaction to the database server and then retain the transaction context.

See Also
5.12.1.7 TIBCConnectionOptions Class

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

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

Syntax

```
TIBCConnectionOptions = class(TDAConnectionOptions);
```

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowImplicitConnect</td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
<tr>
<td>CharLength</td>
<td>Used to specify the size in bytes of a single character.</td>
</tr>
<tr>
<td>Charset</td>
<td>Used to set character set that IBDAC uses to read and write character data.</td>
</tr>
<tr>
<td>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields. It is used</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DisconnectedMode</strong> (inherited from <strong>TDACConnectionOptions</strong>)</td>
<td>Used to open a connection only when needed for performing a server call and closes after performing the operation.</td>
</tr>
<tr>
<td><strong>EnableBCD</strong></td>
<td>Used to enable currency type.</td>
</tr>
<tr>
<td><strong>EnableFMTBCD</strong></td>
<td>Used to enable using FMTBCD instead of float for large integer numbers to keep precision.</td>
</tr>
<tr>
<td><strong>EnableMemos</strong></td>
<td>Used to enable creating TMemoField and TWideMemoField for BLOB fields with subtype 1.</td>
</tr>
<tr>
<td><strong>IPVersion</strong></td>
<td>Used to specify Internet Protocol Version.</td>
</tr>
<tr>
<td><strong>KeepDesignConnected</strong> (inherited from <strong>TDACConnectionOptions</strong>)</td>
<td>Used to prevent an application from establishing a connection at the time of startup.</td>
</tr>
<tr>
<td><strong>LocalFailover</strong> (inherited from <strong>TDACConnectionOptions</strong>)</td>
<td>If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
<tr>
<td><strong>NoDBTriggers</strong></td>
<td>Used to disable all database triggers.</td>
</tr>
<tr>
<td><strong>Protocol</strong></td>
<td>Used to specify the Network protocol of connection with InterBase server.</td>
</tr>
<tr>
<td><strong>Role</strong></td>
<td>Used to specify the InterBase connection role.</td>
</tr>
<tr>
<td><strong>TrustedAuthentication</strong></td>
<td>Used to apply Windows &quot;Trusted User&quot; security for authenticating Firebird users.</td>
</tr>
<tr>
<td><strong>UseUnicode</strong></td>
<td>Used to enable or disable</td>
</tr>
</tbody>
</table>
5.12.1.7.2 Properties

Properties of the `TIBCConnectionOptions` class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DefaultSortType</code> (inherited from <code>TDACConnectionOptions</code>)</td>
<td>Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the <code>TMemDataSet.IndexFieldNames</code> property of a dataset.</td>
</tr>
<tr>
<td><code>DisconnectedMode</code> (inherited from <code>TDACConnectionOptions</code>)</td>
<td>Used to open a connection only when needed for performing a server call and closes after performing the operation.</td>
</tr>
<tr>
<td><code>KeepDesignConnected</code> (inherited from <code>TDACConnectionOptions</code>)</td>
<td>Used to prevent an application from establishing a connection at the time of startup.</td>
</tr>
<tr>
<td><code>LocalFailover</code> (inherited from <code>TDACConnectionOptions</code>)</td>
<td>If True, the <code>TCustomDACConnection.OnConnectionLost</code> event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `AllowImplicitConnect` (inherited from) | Specifies whether to allow or
<table>
<thead>
<tr>
<th><strong>TDACConnectionOptions</strong></th>
<th>not implicit connection opening.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharLength</td>
<td>Used to specify the size in bytes of a single character.</td>
</tr>
<tr>
<td>Charset</td>
<td>Used to set character set that IBDAC uses to read and write character data.</td>
</tr>
<tr>
<td>EnableBCD</td>
<td>Used to enable currency type.</td>
</tr>
<tr>
<td>EnableFMTBCD</td>
<td>Used to enable using FMTBCD instead of float for large integer numbers to keep precision.</td>
</tr>
<tr>
<td>EnableMemos</td>
<td>Used to enable creating TMemoField and TWideMemoField for BLOB fields with subtype 1.</td>
</tr>
<tr>
<td>IPVersion</td>
<td>Used to specify Internet Protocol Version.</td>
</tr>
<tr>
<td>NoDBTriggers</td>
<td>Used to disable all database triggers.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Used to specify the Network protocol of connection with InterBase server.</td>
</tr>
<tr>
<td>Role</td>
<td>Used to specify the InterBase connection role.</td>
</tr>
<tr>
<td>TrustedAuthentication</td>
<td>Used to apply Windows &quot;Trusted User&quot; security for authenticating Firebird users.</td>
</tr>
<tr>
<td>UseUnicode</td>
<td>Used to enable or disable Unicode support.</td>
</tr>
</tbody>
</table>

See Also
- **TIBCConnectionOptions Class**
- **TIBCConnectionOptions Class Members**

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

© 2021 Devart
5.12.1.7.2.1 CharLength Property

Used to specify the size in bytes of a single character.

Class
TIBCConnectionOptions

Syntax

```property CharLength: TIBCCharLength default 0;```

Remarks
Use the CharLength property to specify 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 TIBCConnection to interrogate InterBase server for the actual character length.

The default value is 1.

© 1997-2021
Devart. All Rights Reserved.

5.12.1.7.2.2 Charset Property

Used to set character set that IBDAC uses to read and write character data.

Class
TIBCConnectionOptions

Syntax

```property Charset: string;```

Remarks
Use the Charset property to set character set that IBDAC uses to read and write character data.

© 1997-2021
Devart. All Rights Reserved.
5.12.1.7.2.3 EnableBCD Property

Used to enable currency type.

Class

TIBCConnectionOptions

Syntax

```pascal
property EnableBCD: boolean;
```

Remarks

Use the EnableBCD property to enable currency type.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.7.2.4 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class

TIBCConnectionOptions

Syntax

```pascal
property EnableFMTBCD: boolean;
```

Remarks

Use the EnableFMTBCD property to enable using FMTBCD instead of float for large integer numbers to keep precision.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.7.2.5 EnableMemos Property

Used to enable creating TMemoField and TWideMemoField for BLOB fields with subtype 1.

Class

TIBCConnectionOptions
Syntax

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

Remarks

Use the EnableMemos property to enable creating TMemoField and TWideMemoField for BLOB fields with subtype 1.

5.12.1.7.2.6  IPVersion Property

Used to specify Internet Protocol Version.

Class

`TIBCConnectionOptions`

Syntax

```plaintext
property IPVersion: TIPVersion default DefValIPVer;
```

Remarks

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.

**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.

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.
5.12.1.7.2.7 NoDBTriggers Property

Used to disable all database triggers.

Class

TIBCConnectionOptions

Syntax

property NoDBTriggers: boolean default False;

5.12.1.7.2.8 Protocol Property

Used to specify the Network protocol of connection with InterBase server.

Class

TIBCConnectionOptions

Syntax

property Protocol: TIBCProtocol default DefValProtocol;

Remarks

Use the Protocol property to specify the network protocol of connection with InterBase server. The default value is TCP.

5.12.1.7.2.9 Role Property

Used to specify the InterBase connection role.

Class

TIBCConnectionOptions

Syntax

property Role: string;
Remarks

Use the Role property to specify the InterBase connection role.

5.12.1.7.2.10  TrustedAuthentication Property

Used to apply Windows "Trusted User" security for authenticating Firebird users.

Class

TIBConnectionOptions

Syntax

property TrustedAuthentication: boolean default False;

Remarks

Use the TrustedAuthentication property to apply Windows "Trusted User" security for authenticating Firebird users on a Windows host. When the option is set to True, the Firebird security database is ignored during establishing a connection, and only Windows authentication is used.

The default value is False. More detailed information about this authentication mode is available at http://firebirdsql.org/rlsnotes/rlsnotes210.html#mfb210-wintrusted.

5.12.1.7.2.11  UseUnicode Property

Used to enable or disable Unicode support.

Class

TIBConnectionOptions

Syntax

property UseUnicode: boolean default DefValUseUnicode;

Remarks
Use the UseUnicode property to enable or disable Unicode support. Affects on character data fetched from the server. When set to True all character data is stored as WideStrings, and TStringField is replaced with TWideStringField.

### 5.12.1.8 TIBCDataSetOptions Class

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

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

**Unit**

**IBC**

**Syntax**

\[
\text{TIBCDataSetOptions = class(TDADatasetOptions)};
\]

**Inheritance Hierarchy**

- [TDADatasetOptions](#)
- [TIBCDataSetOptions](#)

### 5.12.1.8.1 Members

- **TIBCDataSetOptions** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoClose</td>
<td>Used to close CustomIBCDataSet cursor after fetching all rows.</td>
</tr>
<tr>
<td>AutoPrepare (inherited)</td>
<td>Used to execute automatic TCustomDADataset.Prepare on the query execution.</td>
</tr>
<tr>
<td>BooleanDomainFields</td>
<td>Used to create TBooleanField for fields that</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CacheArrays</td>
<td>Used to allocate local memory buffer to hold a copy of the array content.</td>
</tr>
<tr>
<td>CacheBlobs</td>
<td>Used to allocate local memory buffer to hold a copy of the BLOB content.</td>
</tr>
<tr>
<td>CacheCalcFields</td>
<td>Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td>ComplexArrayFields</td>
<td>Used to store array fields as TIBCArraryField objects.</td>
</tr>
<tr>
<td>CompressBlobMode</td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used for TCustomIBCDataSet to fill the DefaultExpression property of TField objects by the appropriate value.</td>
</tr>
<tr>
<td>DeferredArrayRead</td>
<td>Used for fetching all InterBase array values when they are explicitly requested.</td>
</tr>
<tr>
<td>DeferredBlobRead</td>
<td>Used for fetching all InterBase BLOB values when they are explicitly requested.</td>
</tr>
<tr>
<td>DescribeParams</td>
<td>Used to specify whether to query TIBCParam properties from the server when executing the TCustomDADDataSet.Prepare method.</td>
</tr>
<tr>
<td>DetailDelay (inherited from TDADatasetOptions)</td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td>FieldsAsString</td>
<td>Used to treat all non-BLOB fields as being of string datatype.</td>
</tr>
<tr>
<td>FieldsOrigin (inherited from TDADatasetOptions)</td>
<td>Used for</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TCUSTOMDADATASET TO FILL THE ORIGIN PROPERTY OF THE TFIELD OBJECTS BY APPROPRIATE VALUE WHEN OPENING A DATASET.</td>
<td>FlatBuffers (inherited from TDADATASETOPTIONS) Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td>FULLREFRESH</td>
<td>Used to refresh fields from all tables of the query.</td>
</tr>
<tr>
<td>INSERTALLSETFIELDS (INHERITED FROM TDADATASETOPTIONS)</td>
<td>Used to include all set dataset fields in the generated INSERT statement</td>
</tr>
<tr>
<td>LOCALMASTERDETAIL (INHERITED FROM TDADATASETOPTIONS)</td>
<td>Used for TCUSTOMDADATASET to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.</td>
</tr>
<tr>
<td>LONGSTRINGS (INHERITED FROM TDADATASETOPTIONS)</td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td>MASTERFIELDSCALLABLE (INHERITED FROM TDADATASETOPTIONS)</td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td>NUMBERRANGE (INHERITED FROM TDADATASETOPTIONS)</td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.</td>
</tr>
<tr>
<td>PREPAREUPDATESQL</td>
<td>Used to automatically prepare update queries before execution.</td>
</tr>
<tr>
<td>QUERYRECOUNT (INHERITED FROM TDADATASETOPTIONS)</td>
<td>Used for TCUSTOMDADATASET to perform additional query to get the record count for this SELECT, so the...</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RecordCount property</td>
<td>reflects the actual number of records.</td>
</tr>
<tr>
<td>QueryRowsAffected</td>
<td>Used to increase the performance of update operations.</td>
</tr>
<tr>
<td>QuoteNames (inherited from TDDataSetOptions)</td>
<td>Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td>RemoveOnRefresh (inherited from TDDataSetOptions)</td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td>RequiredFields (inherited from TDDataSetOptions)</td>
<td>Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td>ReturnParams (inherited from TDDataSetOptions)</td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td>SetDomainNames</td>
<td>Enables setting the TIBCFieldDesc.DomainName property for fields</td>
</tr>
<tr>
<td>SetEmptyStrToNull</td>
<td>Force replace of empty strings with NULL values in data. The default value is False.</td>
</tr>
<tr>
<td>SetFieldsReadOnly (inherited from TDDataSetOptions)</td>
<td>Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.</td>
</tr>
<tr>
<td>StreamedBlobs</td>
<td>Used to save all edited BLOBs as streamed and to handle the streamed</td>
</tr>
<tr>
<td>StrictUpdate</td>
<td>Used for TCustomIBCDataset to raise an exception when the number of the updated or deleted records is not equal 1.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TrimFixedChar</td>
<td>(inherited from TDADataSetOptions) Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
<tr>
<td>UpdateAllFields</td>
<td>(inherited from TDADataSetOptions) Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td>UpdateBatchSize</td>
<td>(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.</td>
</tr>
</tbody>
</table>

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

5.12.1.8.2 Properties

Properties of the TIBCDataSetOptions class.

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

Public

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPrepare</td>
<td>(inherited from TDADataSetOptions) Used to execute automatic TCustomDADataSet.Prepare on the query execution.</td>
</tr>
<tr>
<td>CacheCalcFields</td>
<td>(inherited from TDADataSetOptions) Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td>CompressBlobMode</td>
<td>(inherited from TDADataSetOptions) Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td>DetailDelay</td>
<td>(inherited from TDADataSetOptions) Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td>FieldsOrigin</td>
<td>(inherited from TDADataSetOptions) Used for TCustomDADataSet to fill the Origin property of the</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>FlatBuffers</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td><strong>InsertAllSetFields</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td><strong>LocalMasterDetail</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td><strong>LongStrings</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.</td>
</tr>
<tr>
<td><strong>MasterFieldsNullable</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td><strong>NumberRange</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td><strong>QueryRecCount</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.</td>
</tr>
<tr>
<td><strong>QuoteNames</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used for TCustomDADataset to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
</tbody>
</table>

© 2021 Devart
### InterBase Data Access Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoveOnRefresh (inherited from TDADatasetOptions)</td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td>RequiredFields (inherited from TDADatasetOptions)</td>
<td>Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td>ReturnParams (inherited from TDADatasetOptions)</td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td>SetFieldsReadOnly (inherited from TDADatasetOptions)</td>
<td>Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.</td>
</tr>
<tr>
<td>TrimFixedChar (inherited from TDADatasetOptions)</td>
<td>Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
<tr>
<td>UpdateAllFields (inherited from TDADatasetOptions)</td>
<td>Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td>UpdateBatchSize (inherited from TDADatasetOptions)</td>
<td>Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoClose</td>
<td>Used to for CustomIBCDataset to close cursor after fetching all rows.</td>
</tr>
<tr>
<td>BooleanDomainFields</td>
<td>Used to create TBooleanField for fields that have domain of the integer data type, and the domain name contains 'BOOLEAN'.</td>
</tr>
<tr>
<td>CacheArrays</td>
<td>Used to allocate local</td>
</tr>
<tr>
<td>Key</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CacheBlobs</td>
<td>Used to allocate local memory buffer to hold a copy of the array content.</td>
</tr>
<tr>
<td>ComplexArrayFields</td>
<td>Used to store array fields as TIBCArryField objects.</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used for TCustomBCDataSet to fill the DefaultValue property of TField objects by the appropriate value.</td>
</tr>
<tr>
<td>DeferredArrayRead</td>
<td>Used for fetching all InterBase array values when they are explicitly requested.</td>
</tr>
<tr>
<td>DeferredBlobRead</td>
<td>Used for fetching all InterBase BLOB values when they are explicitly requested.</td>
</tr>
<tr>
<td>DescribeParams</td>
<td>Used to specify whether to query TIBCParam properties from the server when executing the TCustomDADataset.Prepare method.</td>
</tr>
<tr>
<td>FieldsAsString</td>
<td>Used to treat all non-BLOB fields as being of string datatype.</td>
</tr>
<tr>
<td>FullRefresh</td>
<td>Used to refresh fields from all tables of the query.</td>
</tr>
<tr>
<td>PrepareUpdateSQL</td>
<td>Used to automatically prepare update queries before execution.</td>
</tr>
<tr>
<td>QueryRowsAffected</td>
<td>Used to increase the performance of update operations.</td>
</tr>
<tr>
<td>SetDomainNames</td>
<td>Enables setting the TIBCFieldDesc.DomainName property for fields</td>
</tr>
<tr>
<td>SetEmptyStrToNull</td>
<td>Force replace of empty strings with NULL values in data. The default value is False.</td>
</tr>
<tr>
<td>StreamedBlobs</td>
<td>Used to save all edited...</td>
</tr>
</tbody>
</table>
### TIBCDatasetOptions Class Members

#### 5.12.1.8.2.1  AutoClose Property

*Used to for CustomIBCDataset to close cursor after fetching all rows.*

**Class**

TIBCDatasetOptions

**Syntax**

```
property AutoClose: boolean default False;
```

**Remarks**

Use the AutoClose property for CustomIBCDataset to close cursor after fetching all rows. Allows to reduce the number of opened cursors on the server.

#### 5.12.1.8.2.2  BooleanDomainFields Property

*Used to create TBooleanField for fields that have domain of the integer data type, and the domain name contains 'BOOLEAN'.*

**Class**

TIBCDatasetOptions

<table>
<thead>
<tr>
<th>BLOBs as streamed and to handle the streamed</th>
<th>Used for TCustomIBCDataset to raise an exception when the number of the updated or deleted records is not equal 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StrictUpdate</strong></td>
<td></td>
</tr>
</tbody>
</table>

See Also

- TIBCDatasetOptions Class
- TIBCDatasetOptions Class Members

© 1997-2021 Devart. All Rights Reserved.
5.12.1.8.2.3 CacheArrays Property

Used to allocate local memory buffer to hold a copy of the array content.

Class
TIBCDataSetOptions

Syntax

```property BooleanDomainFields: boolean default False;```  

Remarks
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'.

5.12.1.8.2.4 CacheBlobs Property

Used to allocate local memory buffer to hold a copy of the BLOB content.

Class
TIBCDataSetOptions

Syntax

```property CacheArrays: boolean default True;```  

Remarks
If the CacheArray property is set to True (the default value), then local memory buffer is allocated to hold a copy of the array content.

**Note:** This option does not make sense if DefferedArrayRead is set to False because all BLOB values are fetched to the dataset in that case.
property CacheBlobs: boolean default True;

Remarks

If the CacheBlobs property is set to True (the default value), then local memory buffer is allocated to hold a copy of the BLOB content.

Note: CacheBlobs option controls the way streamed BLOB objects are handled. If False, application can access streamed BLOB values on server side without caching BLOBs on client size. Only requested portions of data are fetched. Setting CacheBlobs to False may bring up the following benefits for time-critical applications: reduced traffic over the network since only required data are fetched, less memory is needed on the client side because returned record sets do not hold contents of BLOB fields. This feature is available only for streamed BLOBs and only if StreamedBlobs option is set to True. This option doesn't make sense if DeffferedBlobRead is set to False because all BLOB values are fetched to the dataset in that case.

5.12.1.8.2.5 ComplexArrayFields Property

Used to store array fields as TIBCArrayField objects.

Class

TIBCDataSetOptions

Syntax

property ComplexArrayFields: boolean default True;

Remarks

If the ComplexArrayFields property is set to False, any array field is stored as one TIBCArrayField object. If true and ObjectView is true, array items are stored hierarchically. If true and ObjectView is false, all array items are stored as sibling fields.
5.12.1.8.2.6 DefaultValues Property

Used for TCustomIBCDataset to fill the DefaultExpression property of TField objects by the appropriate value.

Class

TIBCDatasetOptions

Syntax

```plaintext
property DefaultValues: boolean;
```

Remarks

If the DefaultValues property is set to True, TCustomIBCDataset fills the DefaultExpression property of TField objects by the appropriate value. Note that computed BLR fields are not detected and set to read-only if DefaultValues set to false.

5.12.1.8.2.7 DeferredArrayRead Property

Used for fetching all InterBase array values when they are explicitly requested.

Class

TIBCDatasetOptions

Syntax

```plaintext
property DeferredArrayRead: boolean default True;
```

Remarks

If the DeferredArrayRead property is set to True, all InterBase array values are only fetched when they are explicitly requested. Otherwise 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 CacheBlobs option.
5.12.1.8.2.8 DeferredBlobRead Property

Used for fetching all InterBase BLOB values when they are explicitly requested.

Class

TIBCDataSetOptions

Syntax

property DeferredBlobRead: boolean default False;

Remarks

If the DeferredBlobRead property is set to True, all InterBase BLOB values are only fetched 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 or not is controlled by CacheBlobs option.

© 1997-2021
Devart. All Rights Reserved.

5.12.1.8.2.9 DescribeParams Property

Used to specify whether to query TIBCParm properties from the server when executing the TCustomDADataset.Prepare method.

Class

TIBCDataSetOptions

Syntax

property DescribeParams: boolean default False;

Remarks

Specifies whether to query TIBCParm properties (Name, ParamType, DataType, Size, TableName) from the server when executing the TCustomDADataset.Prepare method. The default value is False.

© 1997-2021
Devart. All Rights Reserved.
5.12.1.8.2.10 FieldsAsString Property

Used to treat all non-BLOB fields as being of string datatype.

Class

TIBCDataSetOptions

Syntax

property FieldsAsString: boolean default False;

Remarks

If the FieldsAsString property is set to True, then all non-BLOB fields are treated as being of string datatype.

5.12.1.8.2.11 FullRefresh Property

Used to refresh fields from all tables of the query.

Class

TIBCDataSetOptions

Syntax

property FullRefresh: boolean;

Remarks

Use the FullRefresh property to refresh fields from all tables of the query.

5.12.1.8.2.12 PrepareUpdateSQL Property

Used to automatically prepare update queries before execution.

Class

TIBCDataSetOptions
Syntax

```pascal
property PrepareUpdateSQL: boolean;
```

Remarks
If True, update queries are automatically prepared before executing.

5.12.1.8.2.13 QueryRowsAffected Property

Used to increase the performance of update operations.

Class

`TIBCDatasetOptions`

Syntax

```pascal
property QueryRowsAffected: boolean default True;
```

5.12.1.8.2.14 SetDomainNames Property

Enables setting the `TIBCFieldDesc.DomainName` property for fields

Class

`TIBCDatasetOptions`

Syntax

```pascal
property SetDomainNames: boolean default False;
```

5.12.1.8.2.15 SetEmptyStrToNull Property

Force replace of empty strings with NULL values in data. The default value is False.

Class
**TIBCDataSetOptions**

**Syntax**

```property```
SetEmptyStrToNull: boolean;
```property```

**Remarks**

If the StreamedBlobs property is set to True, then all edited BLOBs are saved as streamed BLOBs and all streamed BLOBs are handled as streamed. Otherwise streamed BLOBs are handled as usual segmented BLOBs and all edited BLOBs are saved as segmented BLOBs. For more information on BLOBs see [BLOB Data Types](#). Setting this option to True allows to use benefits of the CacheBlobs option.

**StrictUpdate Property**

Used for TCustomIBCDataSet to raise an exception when the number of the updated or deleted records is not equal 1.

**Syntax**

```property```
StrictUpdate: boolean;
```property```
Remarks

TCustomIBCDataset raises exception when the number of the updated or deleted records is not equal 1. Setting this option also causes an exception if the RefreshRecord procedure returns more than one record. The exception does not occur when you use non-SQL block. The default value is True. If False, the AffectedRows 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 property to False.

5.12.1.9 TIBCDatasource Class

TIBCDatasource provides an interface between an IBDAC dataset components and data-aware controls on a form.

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

Unit

IBC

Syntax

TIBCDatasource = class(TCRDataSource);

Remarks

TIBCDatasource provides an interface between an IBDAC dataset components and data-aware controls on a form.

TIBCDatasource inherits its functionality directly from the TDataSource component.

At design-time assign individual data-aware components' DataSource properties from their drop-down listboxes.

Inheritance Hierarchy

TCRDataSource

TIBCDatasource
5.12.1.10 TIBCDbKeyField Class

A class representing the InterBase RDB$DB_KEY field.

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

Unit
IBC

Syntax

TIBCDbKeyField = class(TBytesField);

Remarks

This class represents the InterBase RDB$DB_KEY field. It was implemented for the text view of DB_KEY values and does not change TBytesField interface.

See Also

• Updating Data with IBDAC Dataset Components

5.12.1.11 TIBCEncryptor Class

The class that performs encrypting and decrypting of data.

For a list of all members of this type, see TIBCEncryptor members.
Unit
IBC

Syntax

TIBCEncryptor = class(TCREncryptor);

Inheritance Hierarchy

TCREncryptor
  TIBCEncryptor

© 1997-2021 Devart. All Rights Reserved.

5.12.1.11.1 Members

**TIBCEncryptor** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataHeader (inherited from TCREncryptor)</td>
<td>Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td>EncryptionAlgorithm (inherited from TCREncryptor)</td>
<td>Specifies the algorithm of data encryption.</td>
</tr>
<tr>
<td>HashAlgorithm (inherited from TCREncryptor)</td>
<td>Specifies the algorithm of generating hash data.</td>
</tr>
<tr>
<td>InvalidHashAction (inherited from TCREncryptor)</td>
<td>Specifies the action to perform on data fetching when hash data is invalid.</td>
</tr>
<tr>
<td>Password (inherited from TCREncryptor)</td>
<td>Used to set a password that is used to generate a key for encryption.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetKey (inherited from TCREncryptor)</td>
<td>Sets a key, using which data is encrypted.</td>
</tr>
</tbody>
</table>
5.12.1.12 TIBCMetaData Class

A component for obtaining metainformation about database objects from the server.

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

Unit

IBC

Syntax

TIBCMetaData = class(TDAMetaData);

Remarks

The TIBCMetaData 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

TIBCMetaData

See Also

- TCustomDADataset.Debug
- TCustomDASQL.Debug
- DBMonitor

© 1997-2021 Devart. All Rights Reserved.

5.12.1.12.1 Members

TIBCMetaData class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Connection (inherited from TDAMetaData)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>IndexFieldNames (inherited from TMemDataSet)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>KeyExclusive (inherited from TMemDataSet)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>LocalConstraints (inherited from TMemDataSet)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate (inherited from TMemDataSet)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MetaDataKind (inherited from TDAMetaData)</td>
<td>Used to specify which kind of metainformation to show.</td>
</tr>
<tr>
<td>Prepared (inherited from TMemDataSet)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged (inherited from TMemDataSet)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>Restrictions (inherited from TDAMetaData)</td>
<td>Used to provide one or more conditions restricting the list of objects to be described.</td>
</tr>
<tr>
<td>Transaction</td>
<td>Used to determine the transaction under which queries to the server are executed.</td>
</tr>
<tr>
<td>UpdateRecordTypes (inherited from TMemDataSet)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending (inherited from TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>GetMetaDataKinds</strong> (inherited from <strong>TDAMetaData</strong>)</td>
<td>Used to get values acceptable in the MetaDataKind property.</td>
</tr>
<tr>
<td><strong>GetRestrictions</strong> (inherited from <strong>TDAMetaData</strong>)</td>
<td>Used to find out which restrictions are applicable to a certain MetaDataKind.</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Excludes features that don't need to be included to the <strong>TMemDataSet.Locate</strong> method of TDataSet.</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Allocates resources and creates field components for</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Properties of the TIBCMetaData class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MetaDataKind</strong></td>
<td>Used to specify which kind of metainformation to show.</td>
</tr>
</tbody>
</table>
**InterBase Data Access Components**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prepared</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>Restrictions</strong> (inherited from <strong>TDAMetaData</strong>)</td>
<td>Used to provide one or more conditions restricting the list of objects to be described.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transaction</strong></td>
<td>Used to determine the transaction under which queries to the server are executed.</td>
</tr>
</tbody>
</table>

**See Also**

- **TIBCMetaData Class**
- **TIBCMetaData Class Members**

© 1997-2021 Devart. All Rights Reserved.
Use the Transaction property to determine the transaction under which queries to the server are executed.

5.12.1.13 TIBCParam 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 TIBCParam members.

Unit

IBC

Syntax

TIBCParam = class(TDAParam);

Remarks

Use the properties of TIBCParam to set the value of a parameter. Objects that use parameters create TIBCParam objects to represent these parameters. For example, TIBCParam objects are used by TIBCSQL, TCustomIBCDataSet.

TIBCParam 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 TIBCParam object. Conversely, TIBCParam includes properties that indicate how the field value is passed as a parameter.

Inheritance Hierarchy

TDAParam
  TIBParam

See Also

- TCustomIBCDataSet
- TIBCSQL
- TIBCParams
### TIBCPParam class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsArray</td>
<td>Used to specify the value of the parameter when it represents the value of the InterBase array type.</td>
</tr>
<tr>
<td>AsBlob</td>
<td>(inherited from TDAParam) Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td>AsBlobRef</td>
<td>(inherited from TDAParam) Used to set and read the value of the BLOB parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>(inherited from TDAParam) Used to assign the value for a float field to a parameter.</td>
</tr>
<tr>
<td>AsIbBlob</td>
<td>Used to specify the value of the parameter when it represents the value of BLOB type.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>(inherited from TDAParam) Used to assign the value for an integer field to the parameter.</td>
</tr>
<tr>
<td>AsLargeInt</td>
<td>(inherited from TDAParam) Used to assign the value for a LargeInteger field to the parameter.</td>
</tr>
<tr>
<td>AsMemo</td>
<td>(inherited from TDAParam) Used to assign the value for a memo field to the parameter.</td>
</tr>
<tr>
<td>AsMemoRef</td>
<td>(inherited from TDAParam) Used to set and read the value of the memo parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsSQLTimeStamp</td>
<td>(inherited from TDAParam) Used to specify the value of the parameter when it represents a SQL timestamp field.</td>
</tr>
<tr>
<td>AsString</td>
<td>(inherited from TDAParam) Used to assign the string</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>As WideString (inherited from TDAPram)</td>
<td>Used to assign the Unicode string value to the parameter.</td>
</tr>
<tr>
<td>DataType (inherited from TDAPram)</td>
<td>Indicates the data type of the parameter.</td>
</tr>
<tr>
<td>IsNull (inherited from TDAPram)</td>
<td>Used to indicate whether the value assigned to a parameter is NULL.</td>
</tr>
<tr>
<td>ParamType (inherited from TDAPram)</td>
<td>Used to indicate the type of use for a parameter.</td>
</tr>
<tr>
<td>Size (inherited from TDAPram)</td>
<td>Specifies the size of a string type parameter.</td>
</tr>
<tr>
<td>Value (inherited from TDAPram)</td>
<td>Used to represent the value of the parameter as Variant.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignField (inherited from TDAPram)</td>
<td>Assigns field name and field value to a param.</td>
</tr>
<tr>
<td>AssignFieldValue (inherited from TDAPram)</td>
<td>Assigns the specified field properties and value to a parameter.</td>
</tr>
<tr>
<td>LoadFromFile (inherited from TDAPram)</td>
<td>Places the content of a specified file into a TDAPram object.</td>
</tr>
<tr>
<td>LoadFromStream (inherited from TDAPram)</td>
<td>Places the content from a stream into a TDAPram object.</td>
</tr>
<tr>
<td>SetBlobData</td>
<td>Sets the parameter value from the memory buffer.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AsArray</td>
<td>Used to specify the value of the parameter when it represents the value of the InterBase array type.</td>
</tr>
<tr>
<td>AsBlob (inherited from TDAParam)</td>
<td>Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td>AsBlobRef (inherited from TDAParam)</td>
<td>Used to set and read the value of the BLOB parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsFloat (inherited from TDAParam)</td>
<td>Used to assign the value for a float field to a parameter.</td>
</tr>
<tr>
<td>AsIbBlob</td>
<td>Used to specify the value of the parameter when it represents the value of BLOB type.</td>
</tr>
<tr>
<td>AsInteger (inherited from TDAParam)</td>
<td>Used to assign the value for an integer field to the parameter.</td>
</tr>
<tr>
<td>AsLargeInt (inherited from TDAParam)</td>
<td>Used to assign the value for a LargeInteger field to the parameter.</td>
</tr>
<tr>
<td>AsMemo (inherited from TDAParam)</td>
<td>Used to assign the value for a memo field to the parameter.</td>
</tr>
<tr>
<td>AsMemoRef (inherited from TDAParam)</td>
<td>Used to set and read the value of the memo parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsSQLTimeStamp (inherited from TDAParam)</td>
<td>Used to specify the value of the parameter when it represents a SQL timestamp field.</td>
</tr>
<tr>
<td>AsString (inherited from TDAParam)</td>
<td>Used to assign the string value to the parameter.</td>
</tr>
<tr>
<td>AsWideString (inherited from TDAParam)</td>
<td>Used to assign the Unicode string value to the parameter.</td>
</tr>
<tr>
<td>IsNull (inherited from TDAParam)</td>
<td>Used to indicate whether the value assigned to a parameter is NULL.</td>
</tr>
</tbody>
</table>
### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataType</strong></td>
<td>(inherited from <a href="#">TDAParam</a>) Indicates the data type of the parameter.</td>
</tr>
<tr>
<td><strong>ParamType</strong></td>
<td>(inherited from <a href="#">TDAParam</a>) Used to indicate the type of use for a parameter.</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>(inherited from <a href="#">TDAParam</a>) Specifies the size of a string type parameter.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>(inherited from <a href="#">TDAParam</a>) Used to represent the value of the parameter as Variant.</td>
</tr>
</tbody>
</table>

#### See Also
- [TIBCParam Class](#)
- [TIBCParam Class Members](#)

© 1997-2021 Devart. All Rights Reserved.

### 5.12.1.13.2.1 AsArray Property

Used to specify the value of the parameter when it represents the value of the InterBase array type.

#### Class
- [TIBCParam](#)

#### Syntax

```plaintext
property AsArray: TIBCArray;
```

#### Remarks

Use the AsArray property to specify the value of the parameter when it represents the value of the InterBase array type.

Setting AsArray will set the DataType property to ftArray.
5.12.1.13.2.2  AsIbBlob Property

Used to specify the value of the parameter when it represents the value of BLOB type.

Class

TIBCParam

Syntax

property AsIbBlob: TIBCBlob;

Remarks

Use the AsIbBlob property to specify the value of the parameter when it represents the value of BLOB type.

Setting AsIBCBLOB will set the DataType property to ftIBCBLOB.

5.12.1.13.3  Methods

Methods of the TIBCParam class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignField (inherited from TDAParam)</td>
<td>Assigns field name and field value to a param.</td>
</tr>
<tr>
<td>AssignFieldValue (inherited from TDAParam)</td>
<td>Assigns the specified field properties and value to a parameter.</td>
</tr>
<tr>
<td>LoadFromFile (inherited from TDAParam)</td>
<td>Places the content of a specified file into a TDAParam object.</td>
</tr>
<tr>
<td>LoadFromStream (inherited from TDAParam)</td>
<td>Places the content from a stream into a TDAParam object.</td>
</tr>
<tr>
<td>SetBlobData</td>
<td>Sets the parameter value from the memory buffer.</td>
</tr>
</tbody>
</table>
5.12.1.13.3.1  SetBlobData Method

Sets the parameter value from the memory buffer.

Class
TIBCParam

Syntax

procedure SetBlobData(Buffer: IntPtr; Size: integer);

Parameters

Buffer
Holds the pointer to the buffer with data.

Size
Holds the buffer size.

Remarks

Call the SetBLOBData procedure to set the parameter value from the memory buffer. After this procedure call the DataType property is assigned to ftBLOB.

5.12.1.14 TIBCParams Class

Used to control TIBCParam objects.

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

Unit
IBC

Syntax
TIBCParams = class(TDAParams);

Remarks
Use TIBCParams to manage a list of TIBCParam objects for an object that uses field parameters. For example, TIBCStoredProc objects and TIBCQuery objects use TIBCParams objects to create and access their parameters.

Inheritance Hierarchy
TDAParams
   TIBCPrams

See Also
- TIBCPram
- TCustomDASQL.Params
- TCustomDADatSet.Params
- TIBCSQL.Params

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all field parameters.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindParam</td>
<td>Searches a parameter with the name passed in Value.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Searches a parameter with the name passed in Value.</td>
</tr>
</tbody>
</table>
Properties of the **TIBCPParams** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Items</strong></td>
<td>Used to iterate through all field parameters.</td>
</tr>
</tbody>
</table>

**See Also**

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

© 1997-2021

Devart. All Rights Reserved.

### 5.12.1.14.2.1 Items Property(Indexer)

Used to iterate through all field parameters.

**Class**

**TIBCPParams**

**Syntax**

```plaintext
property Items[Index: integer]: TIBCParam; default;
```

**Parameters**

- **Index**
  
  Holds the index in the range 0..Count - 1.

**Remarks**

Use the Items property to iterate through all field parameters. Index identifies the index in the range 0..Count - 1. Items can reference a particular parameter by its index, but the ParamByName method is preferred, so as to avoid depending on the order of the parameters.
### Methods

Methods of the **TIBCPParams** class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FindParam</code></td>
<td>Searches a parameter with the name passed in Value.</td>
</tr>
<tr>
<td><code>ParamByName</code></td>
<td>Searches a parameter with the name passed in Value.</td>
</tr>
</tbody>
</table>

### See Also

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

© 1997-2021 Devart. All Rights Reserved.

### 5.12.1.14.3.1 FindParam Method

Searches a parameter with the name passed in Value.

#### Class

**TIBCPParams**

#### Syntax

```pascal
function FindParam(const Value: string): TIBCPParam;
```

#### Parameters

- **Value**
  
  Holds the parameter name.

#### Return Value

- the parameter, if a match was found.

#### Remarks

Call the `FindParam` method to find a parameter with the name passed in `Value`. If a match is found, `FindParam` returns the parameter. Otherwise, it returns nil. Use this method rather
than a direct reference to the Items property to avoid depending on the order of the entries.

To locate more than one parameter at a time by name, use the GetParamList method instead. To get only the value of a named parameter, use the ParamValues property.

5.12.1.14.3.2  ParamByName Method

Searches a parameter with the name passed in Value.

Class

TIBCParams

Syntax

function ParamByName(const Value: string): TIBCParam;

Parameters

Value

Holds the parameter name.

Return Value

the parameter, if a match was found.

Remarks

Call the ParamByName method to find a parameter with the name passed in Value. If a match is found, ParamByName returns the parameter. Otherwise, an exception is raised. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

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

5.12.1.15  TIBCQuery 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 TIBCQuery members.
Unit

IBC

Syntax

\[
\text{TIBCQuery} = \text{class}(\text{TCustomIBCQuery})
\]

Remarks

TIBCQuery is a direct descendant of the TCustomIBCDataSet component. It publishes most of its inherited properties and events so that they can be manipulated at design-time.

Use TIBCQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. TIBCQuery 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, TIBCQuery will retrieve primary keys for UpdatingTable from metadata. TIBCQuery 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
    TCustomIBCDataSet
      TCustomIBCQuery
        TIBCQuery
See Also
- Updating Data with IBDAC Dataset Components
- Master/Detail Relationships
- TIBCStoredProc
- TIBCTable

© 1997-2021 Devart. All Rights Reserved.

5.12.1.15.1 Members

**TIBCQuery** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoCommit</strong></td>
<td>(inherited from TCustomIBCDataSet) Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td><strong>BaseSQL</strong></td>
<td>(inherited from TCustomDADataset) Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from TMemDataSet) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong></td>
<td>(inherited from TCustomDADataset) Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>(inherited from TCustomIBCDataSet) Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td><strong>Cursor</strong></td>
<td>(inherited from TCustomIBCDataSet) Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>(inherited from TCustomDADataset) Used to set data type mapping rules</td>
</tr>
</tbody>
</table>
| **Debug**          | (inherited from TCustomDADataset) Used to display the statement that is being executed and the values and
<table>
<thead>
<tr>
<th>Property</th>
<th>Inherited From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DetailFields</td>
<td>TCustomDADataset</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>TCustomDADataset</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>DMLRefresh</td>
<td>TCustomIBCDataSet</td>
<td>Used to refresh record by the RETURNING clause when insert is performed.</td>
</tr>
<tr>
<td>Encryption</td>
<td>TCustomIBCDataSet</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td>FetchAll</td>
<td></td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td>FetchRows</td>
<td>TCustomDADataset</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>TCustomDADataset</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>TCustomDADataset</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>GeneratorMode</td>
<td>TCustomIBCDataSet</td>
<td>Used to specify which method is used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>GeneratorStep</td>
<td>TCustomIBCDataSet</td>
<td>Used to set the increment for increasing or decreasing current generator value when using the automatic key field value generation feature.</td>
</tr>
<tr>
<td>Handle</td>
<td>TCustomIBCDataSet</td>
<td>Used to specify the handle for the SQL statement of TCustomIBCDataSet.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>TMemDataSet</td>
<td>Used to get or set the list of fields on which the recordset</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>IsQuery (inherited from <code>TCustomBCDataSet</code>)</td>
<td>Used to check if the SQL statement returns rows.</td>
<td></td>
</tr>
<tr>
<td>KeyExclusive (inherited from <code>TMemDataSet</code>)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
<td></td>
</tr>
<tr>
<td>KeyFields (inherited from <code>TCustomDADataset</code>)</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
<td></td>
</tr>
<tr>
<td>KeyGenerator (inherited from <code>TCustomBCDataSet</code>)</td>
<td>Used 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.</td>
<td></td>
</tr>
<tr>
<td>LocalConstraints (inherited from <code>TMemDataSet</code>)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
<td></td>
</tr>
<tr>
<td>LocalUpdate (inherited from <code>TMemDataSet</code>)</td>
<td>Used to prevent implicit update of rows on database server.</td>
<td></td>
</tr>
<tr>
<td>LockMode</td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
<td></td>
</tr>
<tr>
<td>MacroCount (inherited from <code>TCustomDADataset</code>)</td>
<td>Used to get the number of macros associated with the Macros property.</td>
<td></td>
</tr>
<tr>
<td>Macros (inherited from <code>TCustomDADataset</code>)</td>
<td>Makes it possible to change SQL queries easily.</td>
<td></td>
</tr>
<tr>
<td>MasterFields (inherited from <code>TCustomDADataset</code>)</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
<td></td>
</tr>
<tr>
<td>MasterSource (inherited from <code>TCustomDADataset</code>)</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Options</td>
<td>TCustomIBCDataSet</td>
<td>Used to specify the behaviour of the TCustomIBCDataSet object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>TCustomDADataSet</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>TCustomDADataSet</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>TCustomDADataSet</td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td>Plan</td>
<td>TCustomIBCDataSet</td>
<td>Used to get or set the PLAN clause of the SELECT statement.</td>
</tr>
<tr>
<td>Prepared</td>
<td>TMemDataSet</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>TMemDataSet</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>TCustomDADataSet</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>TCustomDADataSet</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>TCustomDADataSet</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>RowsDeleted</td>
<td>TCustomIBCDataSet</td>
<td>Used to indicate the number of rows that were deleted during the last query operation.</td>
</tr>
<tr>
<td>RowsFetched</td>
<td>TCustomIBCDataSet</td>
<td>Used to get the number of the currently fetched rows.</td>
</tr>
<tr>
<td>RowsInserted</td>
<td>TCustomIBCDataSet</td>
<td>Used to indicate the number of rows that were inserted during the last query operation.</td>
</tr>
<tr>
<td>RowsUpdated</td>
<td>TCustomIBCDataSet</td>
<td>Used to indicate the number of rows that were updated during the last query.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SmartFetch</strong></td>
<td><strong>TCustomIBCDataSet</strong></td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td><strong>SQLType</strong></td>
<td><strong>TCustomIBCDataSet</strong></td>
<td>Used to get the typecode of the SQL statement being processed by the InterBase database server.</td>
</tr>
<tr>
<td><strong>SQLUpdate</strong></td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td><strong>Transaction</strong></td>
<td><strong>TCustomIBCDataSet</strong></td>
<td>Used to determine the transaction under which the query of this dataset executes.</td>
</tr>
<tr>
<td><strong>UniDirectional</strong></td>
<td><strong>TCustomDADataset</strong></td>
<td>Used if an application does not need bidirectional operation.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from TCustomDADataset)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec (inherited from TCustomDADataset)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>Function Name</td>
<td>Inherited From</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>CreateProcCall</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Execute</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Executing</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Fetched</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Fetching</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FindKey</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FindMacro</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FindNearest</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td>FindParam</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>GetArray</td>
<td>Retrieves a TIBCArray object for a field when only its name is known.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>Retrieves a TIBCBlob object for a field when only its name is known.</td>
</tr>
<tr>
<td>GetDataType</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td>specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MacroByName</strong> (inherited from TCustomDADataSet)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong> (inherited from TCustomIBCDataSet)</td>
<td>Called to set or use parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from TCustomDADataSet)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong> (inherited from TCustomDADataSet)</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong> (inherited from TCustomDADataSet)</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong> (inherited from TMemDataSet)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong> (inherited from TMemDataSet)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong> (inherited from TCustomDADataSet)</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong> (inherited from TCustomDADataSet)</td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong> (inherited from TMemDataSet)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong> (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong> (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong> (inherited from TCustomDADataSet)</td>
<td>Determines if the SQL property value was saved to</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UnLock</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>
Properties of the TIBCQuery class.

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

## Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoCommit</strong></td>
<td>Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td><strong>BaseSQL</strong></td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong></td>
<td>Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td><strong>Cursor</strong></td>
<td>Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>Used to set data type mapping rules.</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong></td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>DMLRefresh</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>Encryption</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>FetchRows</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>GeneratorMode</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>GeneratorStep</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>Handle</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>IsQuery</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>KeyFields</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>KeyGenerator</td>
<td>TCustomIBCDataSet</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>LocalConstraints</code></td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><code>LocalUpdate</code></td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><code>MacroCount</code></td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><code>Macros</code></td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><code>MasterFields</code></td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><code>MasterSource</code></td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><code>Options</code></td>
<td>Used to specify the behaviour of the TCustomIBCDataSet object.</td>
</tr>
<tr>
<td><code>ParamCheck</code></td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><code>ParamCount</code></td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><code>Params</code></td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><code>Plan</code></td>
<td>Used to get or set the PLAN clause of the SELECT statement.</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>RowsDeleted</strong></td>
<td><strong>TCustomIBCDataSet</strong></td>
</tr>
<tr>
<td><strong>RowsFetched</strong></td>
<td><strong>TCustomIBCDataSet</strong></td>
</tr>
<tr>
<td><strong>RowsInserted</strong></td>
<td><strong>TCustomIBCDataSet</strong></td>
</tr>
<tr>
<td><strong>RowsUpdated</strong></td>
<td><strong>TCustomIBCDataSet</strong></td>
</tr>
<tr>
<td><strong>SmartFetch</strong></td>
<td><strong>TCustomIBCDataSet</strong></td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify a SQL statement that will be used to refresh current record by calling the <code>TCustomDADataset.RefreshRecord</code> procedure.</td>
</tr>
<tr>
<td><strong>SQLType</strong></td>
<td>(inherited from <code>TCustomIBCDataSet</code>) Used to get the typecode of the SQL statement being processed by the InterBase database server.</td>
</tr>
<tr>
<td><strong>SQLUpdate</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td><strong>Transaction</strong></td>
<td>(inherited from <code>TCustomIBCDataSet</code>) Used to determine the transaction under which the query of this dataset executes.</td>
</tr>
<tr>
<td><strong>UniDirectional</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td><strong>UpdateObject</strong></td>
<td>(inherited from <code>TCustomIBCDataSet</code>) Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Used to check the status of the cached updates buffer.</td>
</tr>
<tr>
<td><strong>UpdateTransaction</strong></td>
<td>(inherited from <code>TCustomIBCDataSet</code>) Used to get or set the transaction for modifying a dataset.</td>
</tr>
</tbody>
</table>
## Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FetchAll</td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td>LockMode</td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
<tr>
<td>UpdatingTable</td>
<td>Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.</td>
</tr>
</tbody>
</table>

### See Also
- [TIBCQuery Class](#)
- [TIBCQuery Class Members](#)

© 1997-2021 Devart. All Rights Reserved.

### 5.12.1.15.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

#### Class
- [TIBCQuery](#)

#### Syntax

```delphi
property FetchAll: boolean;
```

#### Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to
False if initial response time is important.

When the FetchAll property is False, the first call to `TMemDataSet.Locate` and `TMemDataSet.LocateEx` methods may take a lot of time to retrieve additional records to the client side.

### 5.12.1.15.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

**Class**

**TIBCQuery**

**Syntax**

```delphi
property LockMode: TLockMode default lmNone;
```

**Remarks**

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is lmNone.

**See Also**

- `TIBCStoredProc.LockMode`
- `TIBCTable.LockMode`

### 5.12.1.15.2.3 UpdatingTable Property

Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

**Class**

TIBCQuery

Syntax

```plaintext
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 TCustomIBCDataSet.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.

5.12.1.16 TIBCSQL 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 TIBCSQL members.

Unit

IBC

Syntax

```plaintext
TIBCSQL = class(TCustomDASQL);
```

Remarks

The TIBCSQL component is a direct descendant of the TCustomDASQL class.

Use The TIBCSQL 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.
### TIBCSQL class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>Enables or disables changing screen cursor when executing commands in the NonBlocking mode.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to set or return the connection associated with the query.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DescribeParams</td>
<td>Used to specify whether to query TIBCParam properties from the server when executing the TCustomDASQL.Prepare method.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return a SQL statement with expanded macros.</td>
</tr>
<tr>
<td>Handle</td>
<td>Used to specify the handle for the SQL statement of TIBCSQL.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
</tbody>
</table>
### ParamCheck (inherited from TCustomDASQL)
Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

### ParamCount (inherited from TCustomDASQL)
Indicates the number of parameters in the Params property.

### Params
Contains the parameters for 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.

### SQLType
Used to provide the typecode of the SQL statement being processed by the InterBase server.

### Transaction
Used to set or return the transaction to be used by the component.

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BreakExec (inherited from TCustomDASQL)</td>
<td>Breaks execution of an SQL statement on the server.</td>
</tr>
<tr>
<td>CreateProcCall</td>
<td>Assigns SQL block that calls stored procedure specified by Name to SQL property.</td>
</tr>
</tbody>
</table>
### Public

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>ExecuteNext</strong></td>
<td>Provides data that is returned by SQL statement through the out parameters.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>Checks whether TCustomDASQL still executes a SQL statement.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Searches a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Searches a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td>Frees the resources allocated for a previously prepared query on the server.</td>
</tr>
<tr>
<td><strong>WaitExecuting</strong></td>
<td>Waits until TCustomDASQL executes a SQL statement.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong></td>
<td>Occurs after a SQL statement has been executed.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devar. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback

5.12.1.16.2 Properties

Properties of the TIBCSQL class.

For a complete list of the TIBCSQL class members, see the TIBCSQL Members topic.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>Enables or disables changing screen cursor when executing commands in the NonBlocking mode.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return a SQL statement with expanded macros.</td>
</tr>
<tr>
<td>Handle</td>
<td>Used to specify the handle for the SQL statement of TIBCSQL.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Indicates the number of parameters in the Params property.</td>
</tr>
<tr>
<td>ParamValues</td>
<td>Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is</td>
</tr>
</tbody>
</table>
### TIBCSQL Class

**Property**

```
property Connection: TIBCConnection;
```

**Called.**

Used to provide the typecode of the SQL statement being processed by the InterBase server.

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Used to set or return the connection associated with the query.</td>
</tr>
<tr>
<td>DescribeParams</td>
<td>Used to specify whether to query TIBCParam properties from the server when executing the TCustomDASQL.Prepare method.</td>
</tr>
<tr>
<td>Params</td>
<td>Contains the parameters for SQL statement.</td>
</tr>
<tr>
<td>Transaction</td>
<td>Used to set or return the transaction to be used by the component.</td>
</tr>
</tbody>
</table>

**See Also**

- TIBCSQL Class
- TIBCSQL Class Members

© 1997-2021 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback

5.12.1.16.2.1 Connection Property

Used to set or return the connection associated with the query.

**Class**

TIBCSQL

**Syntax**

```
property Connection: TIBCConnection;
```
Remarks

Use the Connection property to set or return the connection associated with the query.

Class

TIBCSQL

Syntax

```property`` DescribeParams: boolean `default` False;
```

Remarks

Specifies whether to query TIBCParam properties (Name, ParamType, DataType, Size, TableTypeName) from the server when executing the TCustomDASQL.Prepare method. The default value is False.

5.12.1.16.2.3 Handle Property

Used to specify the handle for the SQL statement of TIBCSQL.

Class

TIBCSQL

Syntax

```property`` Handle: TISC_STMT_HANDLE;
```

Remarks

Use the Handle property to specify the handle for the SQL statement of TIBCSQL.
5.12.1.16.2.4  Params Property

Contains the parameters for SQL statement.

Class
TIBCSQL

Syntax

```
property Params: TIBCParams stored False;
```

Remarks

The Params property is used to hold the parameters for 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
• TIBCParam
• FindParam

5.12.1.16.2.5  SQLType Property

Used to provide the typecode of the SQL statement being processed by the InterBase server.

Class
TIBCSQL

Syntax

```
property SQLType: integer;
```

Remarks
Use the SQLType property to get the typecode of the SQL statement being processed by the InterBase server.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.16.2.6 Transaction Property

Used to set or return the transaction to be used by the component.

Class

TIBCSQL

Syntax

```plaintext
property Transaction: TIBCTransaction stored IsTransactionStored;
```

Remarks

Use the Transaction property to set or return the transaction to be used by the component.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.16.3 Methods

Methods of the TIBCSQL class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BreakExec (inherited from TCustomDASQL)</td>
<td>Breaks execution of an SQL statement on the server.</td>
</tr>
<tr>
<td>CreateProcCall</td>
<td>Assigns SQL block that calls stored procedure specified by Name property.</td>
</tr>
<tr>
<td>Execute (inherited from TCustomDASQL)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>ExecuteNext</td>
<td>Provides data that is returned by SQL statement through the out parameters.</td>
</tr>
</tbody>
</table>
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 a parameter with the specified name.

MacroByName (inherited from TCustomDASQL) Finds a macro with the specified name.

ParamByName Searches 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
- TIBCSQL Class
- TIBCSQL Class Members

Assigns SQL block that calls stored procedure specified by Name to SQL property.

Class
TIBCSQL

Syntax

```procedure CreateProcCall(const Name: string);```

Parameters

Name
Holds the stored procedure name.

Remarks
Call CreateProcCall to assign SQL block that calls stored procedure specified by Name to SQL property. Retrieves the information about parameters of the procedure from InterBase. After calling CreateProcCall you can execute stored procedure by Execute method.

See Also
- TCustomDASQL.Execute
- TCustomDAConnection.ExecProc
- TIBCStoredProc

5.12.1.16.3.2 ExecuteNext Method

Provides data that is returned by SQL statement through the out parameters.

Class
TIBCSQL

Syntax

```pascal
function ExecuteNext: boolean;
```

Return Value

True, when the data were read and False when it reaches the end of the dataset.

Remarks

Call the ExecuteNext method to get data that is returned by SQL statement through the out parameters. That can be select statement or execution of stored procedure that returns dataset. ExecuteNext returns True when the data were read and False when it reaches the end of dataset.

5.12.1.16.3.3 FindParam Method

Searches a parameter with the specified name.

Class
TIBCSQL
Syntax

```go
function FindParam(const Value: string): TIBCParam;
```

Parameters

**Value**

Holds the parameter name.

Return Value

the parameter, if a match was found.

Remarks

Call the FindParam method to find a parameter with the name passed in the Name argument. If a match is found, FindParam returns the parameter. Otherwise, it returns nil.

See Also

- TIBCParam
- ParamByName

© 1997-2021 Devart. All Rights Reserved.
If a match is found, ParamByName returns the parameter. Otherwise, an exception is raised.

See Also
- TIBCParm
- FindParam

5.12.1.17 TIBCSSLConnectionOptions Class

A class for setting up the SSL options.

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

Unit
IBC

Syntax

```
TIBCSSLConnectionOptions = class(TPersistent);
```

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientCertFile</td>
<td>Holds the path to the client certificate file.</td>
</tr>
<tr>
<td>ClientPassPhrase</td>
<td>Holds the private key passphrase.</td>
</tr>
<tr>
<td>ClientPassPhraseFile</td>
<td>Holds the path to the text file containing the client private key passphrase.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Enables or disables SSL connections.</td>
</tr>
<tr>
<td>ServerPublicFile</td>
<td>Holds the path to the certificate authority file.</td>
</tr>
</tbody>
</table>
5.12.1.17.2 Properties

Properties of the TIBCSSLConnectionOptions class.

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

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ServerPublicPath</strong></td>
<td>Holds the path to the directory with the certificate authority files.</td>
</tr>
<tr>
<td><strong>ClientCertFile</strong></td>
<td>Holds the path to the client certificate file.</td>
</tr>
<tr>
<td><strong>ClientPassPhrase</strong></td>
<td>Holds the private key passphrase.</td>
</tr>
<tr>
<td><strong>ClientPassPhraseFile</strong></td>
<td>Holds the path to the text file containing the client private key passphrase.</td>
</tr>
<tr>
<td><strong>Enabled</strong></td>
<td>Enables or disables SSL connections.</td>
</tr>
<tr>
<td><strong>ServerPublicFile</strong></td>
<td>Holds the path to the certificate authority file.</td>
</tr>
<tr>
<td><strong>ServerPublicPath</strong></td>
<td>Holds the path to the directory with the certificate authority files.</td>
</tr>
</tbody>
</table>

See Also

- [TIBCSSLConnectionOptions Class](#)
- [TIBCSSLConnectionOptions Class Members](#)
5.12.1.17.2.1 ClientCertFile Property

Holds the path to the client certificate file.

Class

TIBCSSLConnectionOptions

Syntax

```property` ClientCertFile: string;```

Remarks

Use the ClientCertFile property to specify the path the client certificate file. The file must be in the PEM format and contain both the client certificate and the private key.

© 1997-2021
Devart. All Rights Reserved.

5.12.1.17.2.2 ClientPassPhrase Property

Holds the private key passphrase.

Class

TIBCSSLConnectionOptions

Syntax

```property` ClientPassPhrase: string;```

Remarks

Use the ClientPassPhrase property to specify the private key passphrase. You can use either this property or the ClientPassPhraseFile property.

© 1997-2021
Devart. All Rights Reserved.

5.12.1.17.2.3 ClientPassPhraseFile Property

Holds the path to the text file containing the client private key passphrase.

Class
**TIBCSSLConnectionOptions**

**Syntax**

```
property ClientPassPhraseFile: string;
```

**Remarks**
Use the ClientPassPhraseFile property to specify the path to the text file containing the client private key passphrase. You can use either this property or the `ClientPassPhrase` property.

---

**5.12.1.17.2.4 Enabled Property**

Enables or disables SSL connections.

**Class**

**TIBCSSLConnectionOptions**

**Syntax**

```
property Enabled: boolean;
```

**Remarks**
Use the Enabled property to enable or disable SSL connections.

---

**5.12.1.17.2.5 ServerPublicFile Property**

Holds the path to the certificate authority file.

**Class**

**TIBCSSLConnectionOptions**

**Syntax**

```
property ServerPublicFile: string;
```

**Remarks**
Use the ServerPublicFile property to specify the path to the CA certificate file in the PEM format.

5.12.1.17.2.6 ServerPublicPath Property

Holds the path to the directory with the certificate authority files.

Class

TIBCSSLConnectionOptions

Syntax

property ServerPublicPath: string;

Remarks

Use the ServerPublicPath property to specify the path to 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 ServerPublicFile instead. If you specify both, ServerPublicFile will be used.

5.12.1.18 TIBCStoredProc Class

A component for accessing and executing stored procedures and functions.

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

Unit

IBC

Syntax

TIBCStoredProc = class(TCustomIBCQuery);

Remarks
Use TIBCStoredProc to access stored procedures on the database server.

You need only to define the StoredProcName property, and the SQL statement to call the stored procedure will be generated automatically.

Use the Execute method at runtime to generate request that instructs server to execute procedure and PrepareSQL to describe parameters at run time.

Inheritance Hierarchy

```
TMemDataSet
  TCustomDADataSet
    TCustomIBCDataSet
      TCustomIBCQuery
        TIBCStoredProc
```

See Also
- TIBCQuery
- TIBCSQL
- Updating Data with IBDAC Dataset Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCommit</td>
<td>(inherited from TCustomIBCDataSet) Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td>BaseSQL</td>
<td>(inherited from TCustomDADataSet) Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates</td>
<td>(inherited from TMemDataSet) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td><strong>Cursor</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to refresh record by the RETURNING clause when insert is performed.</td>
</tr>
<tr>
<td><strong>Encryption</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>FetchAll</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to retrieve all records in a dataset.</td>
</tr>
<tr>
<td><strong>FetchRows</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>GeneratorMode</td>
<td>TCustomIBCDataset</td>
</tr>
<tr>
<td>GeneratorStep</td>
<td>TCustomIBCDataset</td>
</tr>
<tr>
<td>Handle</td>
<td>TCustomIBCDataset</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>IsQuery</td>
<td>TCustomIBCDataset</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>KeyFields</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>KeyGenerator</td>
<td>TCustomIBCDataset</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LockMode</td>
<td></td>
</tr>
<tr>
<td>MacroCount</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Macros</strong></td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong></td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong></td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of the TCustomIBDataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Plan</strong></td>
<td>Used to get or set the PLAN clause of the SELECT statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>RowsDeleted</strong></td>
<td>Used to indicate the number of rows that were deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsFetched</strong></td>
<td>Used to get the number of the currently fetched rows.</td>
</tr>
<tr>
<td><strong>RowsInserted</strong></td>
<td>Used to indicate the number of rows that were inserted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsUpdated</strong></td>
<td>Used to indicate the number of rows that were updated during the last query operation.</td>
</tr>
<tr>
<td><strong>SmartFetch</strong></td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
</tbody>
</table>
### SQLType (inherited from TCustomIBCDataSet)

Used to get the typecode of the SQL statement being processed by the InterBase database server.

### 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 TCustomIBCDataSet)

Used to determine the transaction under which the query of this dataset executes.

### UniDirectional (inherited from TCustomDADataSet)

Used if an application does not need bidirectional access to records in the result set.

### UpdateObject (inherited from TCustomIBCDataSet)

Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.

### UpdateRecordTypes (inherited from TMemDataSet)

Used to indicate the update status for the current record when cached updates are enabled.

### UpdatesPending (inherited from TMemDataSet)

Used to check the status of the cached updates buffer.

### UpdateTransaction (inherited from TCustomIBCDataSet)

Used to get or set the transaction for modifying a dataset.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from TCustomDADataSet)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong></td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td><strong>CreateProcCall</strong></td>
<td>Assigns PL/SQL block that calls stored procedure to the SQL property.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>ExecProc</strong></td>
<td>Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong></td>
<td>Used to find out whether TCustomDADataset has</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Fetching</strong></td>
<td>(inherited from TCustomDADataSet) Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong></td>
<td>(inherited from TCustomDADataSet) Used to learn whether TCustomDADataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td><strong>FindKey</strong></td>
<td>(inherited from TCustomDADataSet) Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>(inherited from TCustomDADataSet) Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</strong></td>
<td>(inherited from TCustomDADataSet) Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>(inherited from TCustomIBCDataSet) Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td><strong>GetArray</strong></td>
<td>(inherited from TCustomIBCDataSet) Retrieves a TIBCArray object for a field when only its name is known.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>(inherited from TCustomIBCDataSet) Retrieves a TIBCBlob object for a field when only its name is known.</td>
</tr>
<tr>
<td><strong>GetDataType</strong></td>
<td>(inherited from TCustomDADataSet) Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldObject</strong></td>
<td>(inherited from TCustomDADataSet) Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong></td>
<td>(inherited from TCustomDADataSet) Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><strong>GetFieldScale</strong></td>
<td>(inherited from TCustomDADataSet) Retrieves the scale of a number field.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong></td>
<td>(inherited from TCustomDADataSet) Provides a list of available key field names.</td>
</tr>
<tr>
<td><strong>GetOrderBy</strong></td>
<td>(inherited from TCustomDADataSet) Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
</tbody>
</table>
| **GotoCurrent**     | (inherited from TCustomDADataSet) Sets the current record in this dataset similar to the
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Locate</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Excludes features that don't need to be included to the <code>TMemDataSet.Locate</code> method of <code>TDataSet</code>.</td>
</tr>
<tr>
<td><strong>Lock</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td><strong>MacroByName</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong> (inherited from <code>TCustomIBCDataset</code>)</td>
<td>Called to set or use parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Describes the stored procedure parameters.</td>
</tr>
<tr>
<td><strong>PrepareSQL</strong></td>
<td>Describes the stored procedure parameters.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
</tbody>
</table>
SetRange (inherited from TMemDataSet)

Sets the starting and ending values of a range, and applies it.

SetRangeEnd (inherited from TMemDataSet)

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

SetRangeStart (inherited from TMemDataSet)

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

SQLSaved (inherited from TCustomDADataset)

Determines if the SQL property value was saved to the BaseSQL property.

UnLock (inherited from TCustomDADataset)

Releases a record lock.

UnPrepare (inherited from TMemDataSet)

Frees the resources allocated for a previously prepared query on the server and client sides.

UpdateResult (inherited from TMemDataSet)

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

UpdateStatus (inherited from TMemDataSet)

Indicates the current update status for the dataset when cached updates are enabled.

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch</td>
<td>Occurs before dataset is</td>
</tr>
</tbody>
</table>
### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoCommit</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td><strong>BaseSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td><strong>Cursor</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used for positioned UPDATE and DELETE operations.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> <em>(inherited from TCustomDADataset)</em></td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong> <em>(inherited from TCustomDADataset)</em></td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong> <em>(inherited from TCustomDADataset)</em></td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong> <em>(inherited from TCustomDADataset)</em></td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong> <em>(inherited from TCustomIBCDataSet)</em></td>
<td>Used to refresh record by the RETURNING clause when insert is performed.</td>
</tr>
<tr>
<td><strong>Encryption</strong> <em>(inherited from TCustomIBCDataSet)</em></td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>FetchAll</strong> <em>(inherited from TCustomIBCDataSet)</em></td>
<td>Used to retrieve all records in a dataset.</td>
</tr>
<tr>
<td><strong>FetchRows</strong> <em>(inherited from TCustomDADataset)</em></td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong> <em>(inherited from TCustomDADataset)</em></td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong> <em>(inherited from TCustomDADataset)</em></td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td><strong>GeneratorMode</strong> <em>(inherited from TCustomIBCDataSet)</em></td>
<td>Used to specify which method is used internally to generate a sequenced field.</td>
</tr>
<tr>
<td><strong>GeneratorStep</strong> <em>(inherited from TCustomIBCDataSet)</em></td>
<td>Used to set the increment for increasing or decreasing current generator value when</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used to specify the handle for the SQL statement of TCustomIBCDataSet.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong> (inherited from TMemDataSet)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>IsQuery</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used to check if the SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from TMemDataSet)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</strong> (inherited from TCustomDADataset)</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td><strong>KeyGenerator</strong> (inherited from TCustomIBCDataSet)</td>
<td>Used 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.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong> (inherited from TMemDataSet)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong> (inherited from TMemDataSet)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MacroCount</strong> (inherited from TCustomDADataset)</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong> (inherited from TCustomDADataset)</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong> (inherited from TCustomDADataset)</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MasterSource</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>(inherited from <strong>TCustomIBCDataset</strong>) Used to specify the behaviour of the TCustomIBCDataset object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Plan</strong></td>
<td>(inherited from <strong>TCustomIBCDataset</strong>) Used to get or set the PLAN clause of the SELECT statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsDeleted</strong></td>
<td>(inherited from <strong>TCustomIBCDataset</strong>) Used to indicate the number of rows that were deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>RowsFetched</strong></td>
<td>(inherited from <strong>TCustomIBCDataset</strong>) Used to get the number of the currently fetched rows.</td>
</tr>
<tr>
<td><strong>RowsInserted</strong></td>
<td>(inherited from <strong>TCustomIBCDataset</strong>) Used to indicate the number of rows that were inserted during the last query.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RowsUpdated</td>
<td>Used to indicate the number of rows that were updated during the last query operation.</td>
</tr>
<tr>
<td>SmartFetch</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLType</td>
<td>Used to get the typecode of the SQL statement being processed by the InterBase database server.</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>Transaction</td>
<td>Used to determine the transaction under which the operation occurred.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>(inherited from TCustomDADataset) Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject</td>
<td>(inherited from TCustomIBCDataSet) Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>(inherited from TMemDataSet) Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending</td>
<td>(inherited from TMemDataSet) Used to check the status of the cached updates buffer.</td>
</tr>
<tr>
<td>UpdateTransaction</td>
<td>(inherited from TCustomIBCDataSet) Used to get or set the transaction for modifying a dataset.</td>
</tr>
</tbody>
</table>

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LockMode</td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
<tr>
<td>StoredProcName</td>
<td>Used to specify the name of the stored procedure to call on the server.</td>
</tr>
</tbody>
</table>

**See Also**
- TIBCStoredProc Class
- TIBCStoredProc Class Members

---

5.12.1.18.2.1  LockMode Property

Used to specify what kind of lock will be performed when editing a record.
**TIBCStoredProc**

**Syntax**

```
property LockMode: TLockMode default lmNone;
```

**Remarks**

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is lmNone.

**See Also**

- **TIBCQuery.LockMode**
- **TIBCTable.LockMode**

© 1997-2021 Devart. All Rights Reserved.

5.12.1.18.2.2 StoredProcName Property

Used to specify the name of the stored procedure to call on the server.

**Class**

**TIBCStoredProc**

**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.

© 1997-2021 Devart. All Rights Reserved.
Methods of the TIBCStoredProc class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from TCustomDADataset)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec (inherited from TCustomDADataset)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream (inherited from TCustomDADataset)</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>CreateProcCall (inherited from TCustomIBCDataSet)</td>
<td>Assigns PL/SQL block that calls stored procedure to the SQL property.</td>
</tr>
<tr>
<td>DeferredPost (inherited from TMemDataSet)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere (inherited from TCustomDADataset)</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>ExecProc</td>
<td>Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td>FindParam</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>GetArray</td>
<td>Retrieves a TIBCArray object for a field when only its name is known.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>Retrieves a TIBCBlob object for a field when only its name is known.</td>
</tr>
<tr>
<td>GetDataType</td>
<td>Returns internal field types</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Called to set or use parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Describes the stored procedure parameters.</td>
</tr>
<tr>
<td>PrepareSQL</td>
<td>Describes the stored procedure parameters.</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveSQL</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetOrderBy</td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td>SetRange</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SQLSaved</td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td>UnLock</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>Indicates the current update status for the dataset when</td>
</tr>
</tbody>
</table>
ExecProc Method

Executes a SQL statement on the server.

Class
TIBCStoredProc

Syntax

```delphi
procedure ExecProc;
```

Remarks

ExecProc is similar to the TCustomDADataset.Execute method. It is included for compatibility with TStoredProc.

See Also

- TStoredProc Class
- TIBCSqlProcedure Class Members

Prepare Method

Describes the stored procedure parameters.

Class
TIBCStoredProc

Syntax

```delphi
procedure Prepare; override;
```
Remarks

Call the Prepare method to describe the parameters of stored procedure. You can define parameters at design time if ParametersEditor is opened. Prepare method prepares the EXECUTE PROCEDURE statement. To prepare the SELECT statement use the `PrepareSQL` method.

See Also

- `PrepareSQL`
- `TCustomDADataset.Execute`

5.12.1.18.3.3  PrepareSQL Method

Describes the stored procedure parameters.

Class

`TIBCStoredProc`

Syntax

```plaintext
procedure PrepareSQL(IsQuery: boolean = False);
```

Parameters

- `IsQuery`
  
  True, if the SELECT statement should be prepared.

Remarks

Use the PrepareSQL method to describe the parameters of stored procedure. The Execute or Open method calls it automatically if it is necessary. You can define the parameters at design time if ParametersEditor is opened. Set `IsQuery` parameter to True to prepare SELECT statement. Set it to False or omit it to prepare EXECUTE PROCEDURE statement.

See Also

- `Prepare`
- `TCustomDADataset.Execute`
Reserved.

5.12.1.19 TIBCTable 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 TIBCTable members.

Unit

IBC

Syntax

TIBCTable = class(TCustomIBCTable);

Remarks

The TIBCTable component allows retrieving and updating data in a single table without writing SQL statements. Use TIBCTable to access data in a table. Use the TableName property to specify table name. TIBCTable uses the KeyFields property to build SQL statements for updating table data. KeyFields is a string containing a semicolon-delimited list of the field names.

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
    TCustomBCDataSet
      TCustomBCQuery
        TCustomIBCTable
          TIBCTable

See Also

- Updating Data with IBDAC Dataset Components
- Master/Detail Relationships
- TCustomIBCTable

© 1997-2021 Devart. All Rights Reserved.
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCommit</td>
<td>Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td>BaseSQL</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions</td>
<td>Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td>Cursor</td>
<td>Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.</td>
</tr>
<tr>
<td>DataTypeMap</td>
<td>Used to set data type mapping rules.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DetailFields</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>DMLRefresh</td>
<td>Used to refresh record by the RETURNING clause when insert is performed.</td>
</tr>
<tr>
<td>Encryption</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>Exists</strong> (inherited from <strong>TCustomIBCTable</strong>)</td>
<td>Indicates whether a table with the name passed in TableName exists in the database.</td>
</tr>
<tr>
<td><strong>FetchAll</strong></td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td><strong>FetchRows</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td><strong>GeneratorMode</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to specify which method is used internally to generate a sequenced field.</td>
</tr>
<tr>
<td><strong>GeneratorStep</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to set the increment for increasing or decreasing current generator value when using the automatic key field value generation feature.</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to specify the handle for the SQL statement of TCustomIBCDataSet.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>IsQuery</strong> (inherited from <strong>TCustomIBCDataSet</strong>)</td>
<td>Used to check if the SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KeyGenerator</td>
<td>Used 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.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>LockMode</td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of the TCustomIBCDataset object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td><code>TCustomDADataSet</code></td>
</tr>
<tr>
<td><strong>Plan</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td><code>TCustomDADataSet</code></td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td><code>TCustomDADataSet</code></td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td><code>TCustomDADataSet</code></td>
</tr>
<tr>
<td><strong>RowsDeleted</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
<tr>
<td><strong>RowsFetched</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
<tr>
<td><strong>RowsInserted</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
<tr>
<td><strong>RowsUpdated</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
<tr>
<td><strong>SmartFetch</strong></td>
<td><code>TCustomIBCDataSet</code></td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td><code>TCustomDADataSet</code></td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td><code>TCustomDADataSet</code></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLInsert (inherited from TCustomDADataSet)</td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock (inherited from TCustomDADataSet)</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount (inherited from TCustomDADataSet)</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh (inherited from TCustomDADataSet)</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLType (inherited from TCustomIBCDataSet)</td>
<td>Used to get the typecode of the SQL statement being processed by the InterBase database server.</td>
</tr>
<tr>
<td>SQLUpdate (inherited from TCustomDADataSet)</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>TableName</td>
<td>Used to specify the name of the database table this component encapsulates.</td>
</tr>
<tr>
<td>Transaction (inherited from TCustomIBCDataSet)</td>
<td>Used to determine the transaction under which the query of this dataset executes.</td>
</tr>
<tr>
<td>UniDirectional (inherited from TCustomDADataSet)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject (inherited from TCustomIBCDataSet)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
</tr>
</tbody>
</table>
**UpdateRecordTypes** (inherited from **TMemDataSet**)  
Used to indicate the update status for the current record when cached updates are enabled.

**UpdatesPending** (inherited from **TMemDataSet**)  
Used to check the status of the cached updates buffer.

**UpdateTransaction** (inherited from **TCustomIBCData**
Set)  
Used to get or set the transaction for modifying a dataset.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **AddWhere**          | (inherited from **TCustomDADataset**)  
Adds condition to the WHERE clause of SELECT statement in the SQL property.                                                                  |
| **ApplyRange**        | (inherited from **TMemDataSet**)  
Applies a range to the dataset.                                                                                                               |
| **ApplyUpdates**      | (inherited from **TMemDataSet**)  
Overloaded. Writes dataset's pending cached updates to a database.                                                                               |
| **BreakExec**         | (inherited from **TCustomDADataset**)  
Breaks execution of the SQL statement on the server.                                                                                           |
| **CancelRange**       | (inherited from **TMemDataSet**)  
Removes any ranges currently in effect for a dataset.                                                                                         |
| **CancelUpdates**     | (inherited from **TMemDataSet**)  
Clears all pending cached updates from cache and restores dataset in its prior state.                                                          |
| **CommitUpdates**     | (inherited from **TMemDataSet**)  
Clears the cached updates buffer.                                                                                                               |
| **CreateBlobStream**  | (inherited from **TCustomDADataset**)  
Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.                              |
| **CreateProcCall**    | (inherited from **TCustomIBCData**
Set)  
Assigns PL/SQL block that calls stored procedure to the SQL property.                                                                           |
| **DeferredPost**      | (inherited from **TMemDataSet**)  
Makes permanent changes to the database server.                                                                                                  |
<table>
<thead>
<tr>
<th>Method</th>
<th>Inherited From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteTable</td>
<td>TCustomIBCTable</td>
<td>Deletes a table from a database.</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>TCustomDADataset</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>TMemDataSet</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>TMemDataSet</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>EmptyTable</td>
<td>TCustomIBCTable</td>
<td>Truncates the current table content on the server.</td>
</tr>
<tr>
<td>Execute</td>
<td>TCustomDADataset</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>TCustomDADataset</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched</td>
<td>TCustomDADataset</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>TCustomDADataset</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>TCustomDADataset</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>TCustomDADataset</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>TCustomDADataset</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>TCustomDADataset</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td>FindParam</td>
<td>TCustomIBCDataset</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>GetArray</td>
<td>TCustomIBCDataset</td>
<td>Retrieves a TIBCArray</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>GetBlob</td>
<td>object for a field when only its name is known.</td>
<td></td>
</tr>
<tr>
<td>GetDataType</td>
<td>Retrieves a TIBCBlob object for a field when only its name is known.</td>
<td></td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>Returns a multireference shared object from field.</td>
<td></td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>Retrieves the precision of a number field.</td>
<td></td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>Retrieves the scale of a number field.</td>
<td></td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
<td></td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
<td></td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
<td></td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
<td></td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
<td></td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the current record.</td>
<td></td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
<td></td>
</tr>
<tr>
<td>ParamByName</td>
<td>Called to set or use parameter information for a specific parameter based on its name.</td>
<td></td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates, opens, and parses cursor for a query.</td>
<td></td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>Actualizes field values for the current record.</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong></td>
<td><strong>TCustomDADataSet</strong></td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td><strong>TMemDataSet</strong></td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td><strong>TMemDataSet</strong></td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong></td>
<td><strong>TCustomDADataSet</strong></td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td><strong>TMemDataSet</strong></td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td><strong>TCustomDADataSet</strong></td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td><strong>TMemDataSet</strong></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong></td>
<td><strong>TMemDataSet</strong></td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong></td>
<td><strong>TMemDataSet</strong></td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong></td>
<td><strong>TCustomDADataSet</strong></td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td><strong>UnLock</strong></td>
<td><strong>TCustomDADataSet</strong></td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td><strong>TMemDataSet</strong></td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong></td>
<td><strong>TMemDataSet</strong></td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are</td>
</tr>
</tbody>
</table>
Properties of the **TIBCTable** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateStatus</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>AutoCommit</strong></td>
<td>(inherited from TCustomIBCDataSet) Used to automatically commit each update, insert or delete statement by database server.</td>
</tr>
<tr>
<td><strong>BaseSQL</strong></td>
<td>(inherited from TCustomDADataset) Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from TMemDataSet) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong></td>
<td>(inherited from TCustomDADataset) Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>(inherited from TCustomIBCDataSet) Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td><strong>Cursor</strong></td>
<td>(inherited from TCustomIBCDataSet) Used for positioned UPDATE and DELETE statements made for the data retrieved with the SELECT statements with the FOR UPDATE clause.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>(inherited from TCustomDADataset) Used to set data type mapping rules.</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>(inherited from TCustomDADataset) Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong></td>
<td>(inherited from TCustomDADataset) Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td>(inherited from TCustomDADataset) Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong></td>
<td>(inherited from TCustomIBCDataSet) Used to refresh record by the RETURNING clause when insert is performed.</td>
</tr>
<tr>
<td><strong>Encryption</strong></td>
<td>(inherited from TCustomIBCDataSet) Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>Exists</strong></td>
<td>(inherited from TCustomIBCTable) Indicates whether a table with the name passed in TableName exists in the</td>
</tr>
</tbody>
</table>
**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.

**GeneratorMode** (inherited from **TCustomIBCDataSet**)  
Used to specify which method is used internally to generate a sequenced field.

**GeneratorStep** (inherited from **TCustomIBCDataSet**)  
Used to set the increment for increasing or decreasing current generator value when using the automatic key field value generation feature.

**Handle** (inherited from **TCustomIBCDataSet**)  
Used to specify the handle for the SQL statement of TCustomIBCDataSet.

**IndexFieldNames** (inherited from **TMemDataSet**)  
Used to get or set the list of fields on which the recordset is sorted.

**IsQuery** (inherited from **TCustomIBCDataSet**)  
Used to check if the 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.

**KeyGenerator** (inherited from **TCustomIBCDataSet**)  
Used 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.
<table>
<thead>
<tr>
<th>Property</th>
<th>Inherited From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalConstraints</td>
<td><strong>TMemDataSet</strong></td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td><strong>TMemDataSet</strong></td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MacroCount</td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td><strong>TCustomDADataset</strong></td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td><strong>TCustomIBCDataSet</strong></td>
<td>Used to specify the behaviour of the TCustomIBCDataSet object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td><strong>TCustomDADataset</strong></td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td>Plan</td>
<td><strong>TCustomIBCDataSet</strong></td>
<td>Used to get or set the PLAN clause of the SELECT statement.</td>
</tr>
<tr>
<td>Prepared</td>
<td><strong>TMemDataSet</strong></td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Variable</td>
<td>Inherited From</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Ranged</td>
<td>TFormDataSet</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>TCustomDADataSet</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>TCustomDADataSet</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>TCustomDADataSet</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>RowsDeleted</td>
<td>TCustomIBCDataSet</td>
<td>Used to indicate the number of rows that were deleted during the last query operation.</td>
</tr>
<tr>
<td>RowsFetched</td>
<td>TCustomIBCDataSet</td>
<td>Used to get the number of the currently fetched rows.</td>
</tr>
<tr>
<td>RowsInserted</td>
<td>TCustomIBCDataSet</td>
<td>Used to indicate the number of rows that were inserted during the last query operation.</td>
</tr>
<tr>
<td>RowsUpdated</td>
<td>TCustomIBCDataSet</td>
<td>Used to indicate the number of rows that were updated during the last query operation.</td>
</tr>
<tr>
<td>SmartFetch</td>
<td>TCustomIBCDataSet</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td>SQL</td>
<td>TCustomDADataSet</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>TCustomDADataSet</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>TCustomDADataSet</td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock</td>
<td>TCustomDADataSet</td>
<td>Used to specify a SQL statement that will be used</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>SQLRecCount (inherited from TCustomDDADataSet)</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
<td></td>
</tr>
<tr>
<td>SQLRefresh (inherited from TCustomDDADataSet)</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDDADataSet.RefreshRecord procedure.</td>
<td></td>
</tr>
<tr>
<td>SQLType (inherited from TCustomDubCastDataSet)</td>
<td>Used to get the typecode of the SQL statement being processed by the InterBase database server.</td>
<td></td>
</tr>
<tr>
<td>SQLUpdate (inherited from TCustomDDADataSet)</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
<td></td>
</tr>
<tr>
<td>Transaction (inherited from TCustomDubCastDataSet)</td>
<td>Used to determine the transaction under which the query of this dataset executes.</td>
<td></td>
</tr>
<tr>
<td>UniDirectional (inherited from TCustomDDADataSet)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
<td></td>
</tr>
<tr>
<td>UpdateObject (inherited from TCustomDubCastDataSet)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
<td></td>
</tr>
<tr>
<td>UpdateRecordTypes (inherited from TMemDataSet)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
<td></td>
</tr>
<tr>
<td>UpdatesPending (inherited from TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
<td></td>
</tr>
<tr>
<td>UpdateTransaction (inherited from TCustomDubCastDataSet)</td>
<td>Used to get or set the transaction for modifying a dataset.</td>
<td></td>
</tr>
</tbody>
</table>
Name | Description
---|---
FetchAll | Defines whether to request all records of the query from database server when the dataset is being opened.
LockMode | Used to specify what kind of lock will be performed when editing a record.
TableName | Used to specify the name of the database table this component encapsulates.

See Also
- TIBCTable Class
- TIBCTable Class Members

5.12.1.19.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class
- TIBCTable

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.
5.12.1.19.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class
TIBCTable

Syntax

```
property LockMode: TLockMode default lmNone;
```

Remarks

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is lmNone.

See Also
- TIBCStoredProc.LockMode
- TIBCQuery.LockMode

5.12.1.19.2.3 TableName Property

Used to specify the name of the database table this component encapsulates.

Class
TIBCTable

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 and correct connection settings are provided, select a valid table name from the TableName drop-down list in Object Inspector.

5.12.1.20 TIBCTransaction Class

A component for managing transactions in an application.

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

Unit
IBC

Syntax

TIBCTransaction = class(TDATransaction);

Remarks

The TIBCTransaction component is used to provide discrete transaction control over connection. It can be used for manipulating simple local and global transactions.

All components which are dedicated to perform data access, such as TIBCQuery, TIBCSQL, TIBCScript, must have their Transaction property assigned with one of TIBCTransaction instances.

Inheritance Hierarchy

TDATransaction
    TIBCTransaction

See Also

- TCustomDACConnection.StartTransaction
- TCustomDACConnection.Commit
- TCustomDACConnection.Rollback
- TIBCConnection.DefaultTransaction
- TIBCConnection.Transactions
- TCustomIBCDataSet.Transaction
**TIBCTransaction** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong></td>
<td>Determines if the transaction is active or not.</td>
</tr>
<tr>
<td><strong>Connections</strong></td>
<td>Used to specify a connection for the given index.</td>
</tr>
<tr>
<td><strong>ConnectionsCount</strong></td>
<td>Used to get the number of connections associated with the transaction component.</td>
</tr>
<tr>
<td><strong>DefaultCloseAction</strong></td>
<td>Used to specify the transaction behaviour when connection closes.</td>
</tr>
<tr>
<td><strong>DefaultConnection</strong></td>
<td>Used to access the default connection of the transaction.</td>
</tr>
<tr>
<td><strong>Handle</strong></td>
<td>Used to specify the handle of the transaction.</td>
</tr>
<tr>
<td><strong>IsolationLevel</strong></td>
<td>Used to get or set the transaction isolation level and access mode.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to access transaction parameters of the transaction parameter buffer.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddConnection</strong></td>
<td>Binds a TCustomDACConnection object with the transaction component.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Commit</strong></td>
<td>Stores all changes of data associated with the transaction to the database server permanently.</td>
</tr>
<tr>
<td><strong>CommitRetaining</strong></td>
<td>Stores to the database server all changes of data associated with the transaction permanently and then retains the transaction context.</td>
</tr>
<tr>
<td><strong>FindDefaultConnection</strong></td>
<td>Returns the default connection for the transaction.</td>
</tr>
<tr>
<td><strong>ReleaseSavepoint</strong></td>
<td>Destroys the specified savepoint without affecting any work that has been performed after its creation.</td>
</tr>
<tr>
<td><strong>RemoveConnection</strong></td>
<td>Disassociates the specified connections from the transaction.</td>
</tr>
<tr>
<td><strong>Rollback</strong></td>
<td>Rolls back all changes of data associated with the transaction.</td>
</tr>
<tr>
<td><strong>RollbackRetaining</strong></td>
<td>Rolls back all data changes associated with the transaction and retains the transaction context.</td>
</tr>
<tr>
<td><strong>RollbackSavepoint</strong></td>
<td>Cancels all updates for the current transaction and restores its state up to the moment of the last defined savepoint.</td>
</tr>
<tr>
<td><strong>StartSavepoint</strong></td>
<td>Defines a point in the transaction to which you can roll back later.</td>
</tr>
<tr>
<td><strong>StartTransaction</strong></td>
<td>(inherited from <a href="#">TDATransaction</a>) Begins a new transaction.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnCommit</strong></td>
<td>(inherited from <a href="#">TDATransaction</a>) Occurs after the transaction has been successfully committed.</td>
</tr>
</tbody>
</table>
### Properties of the TIBCTransaction class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>Used to specify a connection for the given index.</td>
</tr>
<tr>
<td>ConnectionsCount</td>
<td>Used to get the number of connections associated with the transaction component.</td>
</tr>
<tr>
<td>Handle</td>
<td>Used to specify the handle of the transaction.</td>
</tr>
</tbody>
</table>

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

© 2021 Devart
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Determines if the transaction is active or not.</td>
</tr>
<tr>
<td>DefaultCloseAction</td>
<td>Used to specify the transaction behaviour when connection closes.</td>
</tr>
<tr>
<td>DefaultConnection</td>
<td>Used to access the default connection of the transaction.</td>
</tr>
<tr>
<td>IsolationLevel</td>
<td>Used to get or set the transaction isolation level and access mode.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to access transaction parameters of the transaction parameter buffer.</td>
</tr>
</tbody>
</table>

See Also

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

5.12.1.20.2.1 Active Property

Determines if the transaction is active or not.

Class

[TIBCTransaction](#)

Syntax

```pascal
property Active: Boolean stored IsActiveStored default False;
```

Remarks

The Active property is used to indicate whether transaction is active or not.
5.12.1.20.2.2 Connections Property (Indexer)

Used to specify a connection for the given index.

Class

TIBCTransaction

Syntax

[property] Connections[Index: integer]: TIBCConnection;

Parameters

Index
Holds the index for which to specify a connection.

Remarks

Use the Connections property to specify a connection for the given index.

See Also

- ConnectionsCount
- AddConnection
- RemoveConnection

5.12.1.20.2.3 ConnectionsCount Property

Used to get the number of connections associated with the transaction component.

Class

TIBCTransaction

Syntax

[property] ConnectionsCount: integer;

Remarks

Use the ConnectionsCount property for getting the number of connections associated with the transaction component.
5.12.1.20.2.4 DefaultCloseAction Property

Used to specify the transaction behaviour when connection closes.

**Class**

**TIBCTransaction**

**Syntax**

```plaintext
property DefaultCloseAction: TIBCTransactionAction default taRollback;
```

**Remarks**

Use the DefaultCloseAction property to specify the transaction behaviour when connection closes.

5.12.1.20.2.5 DefaultConnection Property

Used to access the default connection of the transaction.

**Class**

**TIBCTransaction**

**Syntax**

```plaintext
property DefaultConnection: TIBCConnection stored IsInternalTrStored;
```

**Remarks**

Use the DefaultConnection property to access the default connection of the transaction.
5.12.1.20.2.6 Handle Property

Used to specify the handle of the transaction.

Class

TIBCTransaction

Syntax

property Handle: TISC_TR_HANDLE;

Remarks

Use the Handle property to specify the handle of the transaction. Use Handle Property to make calls directly to the InterBase API. Some of the InterBase API functions require a transaction handle as an argument.

5.12.1.20.2.7 IsolationLevel Property

Used to get or set the transaction isolation level and access mode.

Class

TIBCTransaction

Syntax

property IsolationLevel: TIBCIsolationLevel default iblReadCommitted;

Remarks

Use the IsolationLevel property to get or set the transaction isolation level and access mode. You should add IBCClasses unit to the uses list to use this property.
5.12.1.20.2.8 Params Property

Used to access transaction parameters of the transaction parameter buffer.

Class

**TIBCTransaction**

Syntax

```
property Params: TStrings;
```

Remarks

Use the Params property to access transaction parameters of the transaction parameter buffer. Refer to InterBase API Guide for more information on this parameters.

© 1997-2021
Devart. All Rights Reserved.

5.12.1.20.3 Methods

Methods of the **TIBCTransaction** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddConnection</strong></td>
<td>Binds a TCustomDAConnection object with the transaction component.</td>
</tr>
<tr>
<td><strong>Commit</strong></td>
<td>Stores all changes of data associated with the transaction to the database server permanently.</td>
</tr>
<tr>
<td><strong>CommitRetaining</strong></td>
<td>Stores to the database server all changes of data associated with the transaction permanently and then retains the transaction context.</td>
</tr>
<tr>
<td><strong>FindDefaultConnection</strong></td>
<td>Returns the default</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ReleaseSavepoint</td>
<td>Destroys the specified savepoint without affecting any work that has been performed after its creation.</td>
</tr>
<tr>
<td>RemoveConnection</td>
<td>Disassociates the specified connections from the transaction.</td>
</tr>
<tr>
<td>Rollback</td>
<td>Rolls back all changes of data associated with the transaction.</td>
</tr>
<tr>
<td>RollbackRetaining</td>
<td>Rolls back all data changes associated with the transaction and retains the transaction context.</td>
</tr>
<tr>
<td>RollbackSavepoint</td>
<td>Cancels all updates for the current transaction and restores its state up to the moment of the last defined savepoint.</td>
</tr>
<tr>
<td>StartSavepoint</td>
<td>Defines a point in the transaction to which you can roll back later.</td>
</tr>
<tr>
<td>StartTransaction</td>
<td>Begins a new transaction.</td>
</tr>
</tbody>
</table>

**See Also**
- [TIBCTransaction Class](#)
- [TIBCTransaction Class Members](#)

**Syntax**

```plaintext
Binds a TCustomDACConnection object with the transaction component.

Class

TIBCTransaction

Syntax
```
function AddConnection(Connection: TIBCConnection): integer;

Parameters

Connection
Holds a TCustomDACConnection object to associate with the transaction component.

Return Value
the index of associated connection in the connection list.

Remarks
Use the AddConnection method to associate a TCustomDACConnection object with the transaction component.

See Also
• RemoveConnection

© 1997-2021
Devart. All Rights Reserved.

5.12.1.20.3.2 Commit Method

Stores all changes of data associated with the transaction to the database server permanently.

Class
TIBCTransaction

Syntax

procedure Commit; override;

Remarks
Call the Commit method to store to the database server all changes of data associated with the transaction permanently.

© 1997-2021
Devart. All Rights Reserved.

5.12.1.20.3.3 CommitRetaining Method

Stores to the database server all changes of data associated with the transaction permanently and then retains the transaction context.

© 2021 Devart
Class
TIBCTransaction

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.

5.12.1.20.3.4  FindDefaultConnection Method

Returns the default connection for the transaction.

Class
TIBCTransaction

Syntax

function FindDefaultConnection: TIBCConnection;

Remarks
Call the FindDefaultConnection method to return the default connection for the transaction.

5.12.1.20.3.5  ReleaseSavepoint Method

Destroys the specified savepoint without affecting any work that has been performed after its
creation.

Class
TIBCTransaction

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
- StartSavepoint
- RollbackSavepoint

© 1997-2021 Devart. All Rights Reserved.

5.12.1.20.3.6 RemoveConnection Method

Disassociates the specified connections from the transaction.

Class
TIBCTransaction

Syntax

procedure RemoveConnection(Connection: TIBCConnection);

Parameters

Connection
Holds the connections to disassociate.

Remarks
Use the RemoveConnection method to disassociate the specified connections from the transaction.

See Also
- AddConnection

© 1997-2021 Devart. All Rights Reserved.
5.12.1.20.3.7 Rollback Method

Rolls back all changes of data associated with the transaction.

Class

TIBCTransaction

Syntax

```pascal
procedure Rollback; override;
```

Remarks

Call the Rollback method to roll back all changes of data associated with the transaction.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.20.3.8 RollbackRetaining Method

Rolls back all data changes associated with the transaction and retains the transaction context.

Class

TIBCTransaction

Syntax

```pascal
procedure RollbackRetaining;
```

Remarks

Call the RollbackRetaining method to roll back all changes of data associated with the transaction and retain the transaction context.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.20.3.9 RollbackSavepoint Method

Cancels all updates for the current transaction and restores its state up to the moment of the last defined savepoint.

Class
**TIBCTransaction**

Syntax

```pascal
procedure RollbackSavepoint(const Name: string);
```

**Parameters**

- **Name**
  
  Holds the defined savepoint name.

**Remarks**

Call the RollbackSavepoint to cancel all updates for the current transaction and restore its state up to the moment of the last defined savepoint.

**See Also**

- [Rollback](#)
- [StartSavepoint](#)
- [ReleaseSavepoint](#)

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.12.1.20.3.10 StartSavepoint Method

Defines a point in the transaction to which you can roll back later.

**Class**

**TIBCTransaction**

Syntax

```pascal
procedure StartSavepoint(const Name: string);
```

**Parameters**

- **Name**
  
  Holds the savepoint name.

**Remarks**

Call the StartSavepoint method to define a point in the transaction to which you can roll back later. As the parameter, you can pass any valid name to identify the savepoint.

To roll back to the last savepoint call [RollbackSavepoint](#).
See Also
- RollbackSavepoint
- ReleaseSavepoint

Events of the TIBCTransaction class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnCommit (inherited from TDATransaction)</td>
<td>Occurs after the transaction has been successfully committed.</td>
</tr>
<tr>
<td>OnCommitRetaining (inherited from TDATransaction)</td>
<td>Occurs after CommitRetaining has been executed.</td>
</tr>
<tr>
<td>OnRollback (inherited from TDATransaction)</td>
<td>Occurs after the transaction has been successfully rolled back.</td>
</tr>
<tr>
<td>OnRollbackRetaining (inherited from TDATransaction)</td>
<td>Occurs after RollbackRetaining has been executed.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnError</td>
<td>Occurs when processing errors that are raised during executing transaction and savepoint control statements such as COMMIT, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT and others.</td>
</tr>
</tbody>
</table>
5.12.1.20.4.1 OnError Event

Occurs when processing errors that are raised during executing transaction and savepoint control statements such as COMMIT, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT and others.

Class

TIBCTransaction

Syntax

property OnError: TIBCTransactionErrorEvent;

Remarks

Write the OnError event handler to process errors that occur during executing transaction and savepoint control statements such as COMMIT, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT and others. Check the E parameter to get an error code.

Note: You should explicitly add T:Devart.IbDac.Units.IBCError unit to 'uses' list to use OnError event handler.

This event occur only when two or more connections are associated with the transaction. When only one connection is assigned to the transaction, then the OnError event of the TIBCConnection class arises.

5.12.1.21 TIBCUpdateSQL Class

A component for tuning update operations for the DataSet component.

For a list of all members of this type, see TIBCUpdateSQL members.
IBC

Syntax

TIBCUpdateSQL = class(TCustomDAUpdateSQL);

Remarks

Use the TIBCUpdateSQL 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
  TIBCUpdateSQL

© 1997-2021 Devart. All Rights Reserved.

5.12.1.21.1 Members

TIBCUpdateSQL class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSet</td>
<td>(inherited from TCustomDAUpdateSQL) Used to hold a reference to the TCustomDADataSet object that is being updated.</td>
</tr>
<tr>
<td>DeleteObject</td>
<td>(inherited from TCustomDAUpdateSQL) Provides ability to perform advanced adjustment of the delete operations.</td>
</tr>
<tr>
<td>DeleteSQL</td>
<td>(inherited from TCustomDAUpdateSQL) Used when deleting a record.</td>
</tr>
<tr>
<td>InsertObject</td>
<td>(inherited from TCustomDAUpdateSQL) Provides ability to perform advanced adjustment of insert operations.</td>
</tr>
<tr>
<td>InsertSQL</td>
<td>(inherited from TCustomDAUpdateSQL) Used when inserting a record.</td>
</tr>
<tr>
<td>LockObject</td>
<td>(inherited from TCustomDAUpdateSQL) Provides ability to perform advanced adjustment of lock operations.</td>
</tr>
</tbody>
</table>
LockSQL (inherited from TCustomDAUpdateSQL) | Used to lock the current record.
ModifyObject (inherited from TCustomDAUpdateSQL) | Provides ability to perform advanced adjustment of modify operations.
ModifySQL (inherited from TCustomDAUpdateSQL) | Used when updating a record.
RefreshObject (inherited from TCustomDAUpdateSQL) | Provides ability to perform advanced adjustment of refresh operations.
RefreshSQL (inherited from TCustomDAUpdateSQL) | Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataset.RefreshRecord procedure.
SQL (inherited from TCustomDAUpdateSQL) | Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply (inherited from TCustomDAUpdateSQL)</td>
<td>Sets parameters for a SQL statement and executes it to update a record.</td>
</tr>
<tr>
<td>ExecSQL (inherited from TCustomDAUpdateSQL)</td>
<td>Executes a SQL statement.</td>
</tr>
</tbody>
</table>

5.12.2 Types

Types in the IBC unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCTransactionErrorEvent</td>
<td>This type is used for the TIBCTransactionOnError event.</td>
</tr>
</tbody>
</table>
5.12.2.1 TIBCTransactionErrorEvent Procedure Reference

This type is used for the TIBCTransaction.OnError event.

Unit
IBC

Syntax

```
TIBCTransactionErrorEvent = procedure (Sender: TObject; E: EIBCError) of object;
```

Parameters

Sender
An object that raised the event.

E
Holds the error code.

5.12.3 Enumerations

Enumerations in the IBC unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGeneratorMode</td>
<td>Specifies the method used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>TIBCProtocol</td>
<td>Specifies the network protocol of connection with InterBase server.</td>
</tr>
<tr>
<td>TIBCTransactionAction</td>
<td>Specifies the transaction behaviour when connection closes.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.12.3.1 TGeneratorMode Enumeration

Specifies the method used internally to generate a sequenced field.

Unit
IBC

Syntax

\[
\text{TGeneratorMode} = (\text{gmInsert, gmPost});
\]

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmInsert</td>
<td>New record is inserted into the dataset where the first key field populated with a sequenced value. Application may modify this field before posting the record to the database.</td>
</tr>
<tr>
<td>gmPost</td>
<td>Database server populates the key field with a sequenced value when application posts the record to the database. However if user specified the key field value, the key generator will not be used.</td>
</tr>
</tbody>
</table>

5.12.3.2 TIBCProtocol Enumeration

Specifies the network protocol of connection with InterBase server.

Unit
IBC

Syntax

\[
\text{TIBCProtocol} = (\text{TCP, NetBEUI, SPX});
\]

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBEUI</td>
<td>Uses the NetBEUI protocol.</td>
</tr>
<tr>
<td>SPX</td>
<td>Uses the SPX protocol.</td>
</tr>
<tr>
<td>TCP</td>
<td>Uses the TCP protocol. The default value.</td>
</tr>
</tbody>
</table>
5.12.3.3 TIBCTransactionAction Enumeration

Specifies the transaction behaviour when connection closes.

Unit
IBC

Syntax

\[
\text{TIBCTransactionAction} = (\text{taCommit}, \text{taRollback}, \text{taCommitRetaining}, \text{taRollbackRetaining});
\]

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{taCommit}</td>
<td>Transaction is committed and is closed.</td>
</tr>
<tr>
<td>\text{taCommitRetaining}</td>
<td>Transaction is committed and remains opened.</td>
</tr>
<tr>
<td>\text{taRollback}</td>
<td>Transaction is rolled back and is closed.</td>
</tr>
<tr>
<td>\text{taRollbackRetaining}</td>
<td>Transaction is rolled back and remains opened.</td>
</tr>
</tbody>
</table>

5.12.4 Variables

Variables in the IBC unit.

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>Holds pointers to all TIBCConnection objects of an application.</td>
</tr>
<tr>
<td>DefConnection</td>
<td>Read this variable to get pointer to default connection object. Same as DefaultConnection function.</td>
</tr>
<tr>
<td>UseDefConnection</td>
<td>When set to true enables</td>
</tr>
</tbody>
</table>
5.12.4.1 Connections Variable

Holds pointers to all TIBCConnection objects of an application.

Unit
IBC

Syntax

Connections: TConnectionList;

5.12.4.2 DefConnection Variable

Read this variable to get pointer to default connection object. Same as DefaultConnection function.

Unit
IBC

Syntax

DefConnection: TIBCConnection;

5.12.4.3 UseDefConnection Variable

When set to true enables TCustomIBCDataSet and TIBCSQL components to use default connection if they are not attached to any connection.
5.12.5 Constants

Constants in the IBC unit.

### Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBDACVersion</td>
<td>Read this constant to get current version number for IBDAC.</td>
</tr>
</tbody>
</table>

5.12.5.1 IBDACVersion Constant

Read this constant to get current version number for IBDAC.

5.13 IBCAdmin

This unit contains implementation of components, used for InterBase/Firebird server administration.
## Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomIBCService</td>
<td>TCustomIBCService is the ancestor object from which all TIBCService components descend.</td>
</tr>
<tr>
<td>TIBCBackupRestoreService</td>
<td>TIBCBackupRestoreService is the base class from which TIBCBackupService and TIBCRestoreService are derived.</td>
</tr>
<tr>
<td>TIBCBackupService</td>
<td>Used to backup a database.</td>
</tr>
<tr>
<td>TIBCConfigParams</td>
<td>TIBCConfigParams holds the configuration information about an InterBase server.</td>
</tr>
<tr>
<td>TIBCConfigService</td>
<td>Use a TIBConfigService object to configure database parameters.</td>
</tr>
<tr>
<td>TIBCControlAndQueryService</td>
<td>TIBCControlAndQueryService is the base class from which the log, statistical, validation, security, and backup and restore TIBCService components descend.</td>
</tr>
<tr>
<td>TIBCDatabaseInfo</td>
<td>Describes an InterBase database.</td>
</tr>
<tr>
<td>TIBCJournalInformation</td>
<td>A class used to access the TIBCConfigService component properties.</td>
</tr>
<tr>
<td>TIBCLicenseInfo</td>
<td>Stores information about licensed users.</td>
</tr>
<tr>
<td>TIBCLicenseMaskInfo</td>
<td>Indicates the software activation certificate options enabled on the server.</td>
</tr>
<tr>
<td>TIBCLicensingService</td>
<td>TIBCLicensingService configures the licensing parameters</td>
</tr>
<tr>
<td>TIBCLimboTransactionInfo</td>
<td>TIBCLimboTransactionInfo stores information about a limbo transaction.</td>
</tr>
<tr>
<td>TIBCLogService</td>
<td>Returns the contents of the interbase.log file from</td>
</tr>
</tbody>
</table>
## Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCBackupOptions</td>
<td>Represents the set of TIBCBackupOption.</td>
</tr>
<tr>
<td>TIBCNBackupOptions</td>
<td>Represents the set of TIBCNBackupOption.</td>
</tr>
<tr>
<td>TIBCRestoreOptions</td>
<td>Represents the set of TIBCRestoreOption.</td>
</tr>
<tr>
<td>TIBCStatOptions</td>
<td>Represents the set of TIBCStatOption.</td>
</tr>
<tr>
<td>TIBCValidateOptions</td>
<td>Represents the set of TIBCValidateOption.</td>
</tr>
</tbody>
</table>

## Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCBackupOption</td>
<td>Allows you to build backup</td>
</tr>
</tbody>
</table>
### Classes

Classes in the `IBCAdmin` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomIBCService</td>
<td>TCustomIBCService is the ancestor object from which all TIBCService components descend.</td>
</tr>
<tr>
<td>TIBCBackupRestoreService</td>
<td>TIBCBackupRestoreService is the base class from which <code>TIBCBackupService</code> and</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TIBCBackupService</td>
<td>Used to backup a database.</td>
</tr>
<tr>
<td>TIBCConfigParams</td>
<td>TIBCConfigParams holds the configuration information about an InterBase server.</td>
</tr>
<tr>
<td>TIBCConfigService</td>
<td>Use a TIBCConfigService object to configure database parameters.</td>
</tr>
<tr>
<td>TIBCControlAndQueryService</td>
<td>TIBCControlAndQueryService is the base class from which the log, statistical, validation, security, and backup and restore TIBCServic components descend.</td>
</tr>
<tr>
<td>TIBCDatabaseInfo</td>
<td>Describes an InterBase database.</td>
</tr>
<tr>
<td>TIBCJournalInformation</td>
<td>A class used to access the TIBCConfigService component properties.</td>
</tr>
<tr>
<td>TIBCLicenseInfo</td>
<td>Stores information about licensed users.</td>
</tr>
<tr>
<td>TIBCLicenseMaskInfo</td>
<td>Indicates the software activation certificate options enabled on the server.</td>
</tr>
<tr>
<td>TIBCLicensingService</td>
<td>TIBCLicensingService configures the licensing parameters</td>
</tr>
<tr>
<td>TIBCLimboTransactionInfo</td>
<td>TIBCLimboTransactionInfo stores information about a limbo transaction.</td>
</tr>
<tr>
<td>TIBCLogService</td>
<td>Returns the contents of the interbase.log file from server.</td>
</tr>
<tr>
<td>TIBCRestoreService</td>
<td>Used to restore a database.</td>
</tr>
<tr>
<td>TIBCSecurityService</td>
<td>Used to manage user access to the InterBase server.</td>
</tr>
<tr>
<td>TIBCServerProperties</td>
<td>A class that returns database server information.</td>
</tr>
<tr>
<td>TIBCStatisticalService</td>
<td>TIBCStatisticalService is used to view database data.</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TIBCTraceService</td>
<td>This component is used for working with trace service added in Firebird 2.5.</td>
</tr>
<tr>
<td>TIBCUserInfo</td>
<td>TIBCUserInfo stores information about an InterBase user for the security service.</td>
</tr>
<tr>
<td>TIBCValidationService</td>
<td>Used to validate a database and reconcile database transactions.</td>
</tr>
<tr>
<td>TIBCVersionInfo</td>
<td>Represents the version information about an InterBase server.</td>
</tr>
</tbody>
</table>

5.13.1.1 TCustomIBCService Class

TCustomIBCService is the ancestor object from which all TIBCService components descend.

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

Unit

IBCAdmin

Syntax

```pascal
TCustomIBCService = class(TComponent);
```

Remarks

TCustomIBCService is the ancestor object from which all TIBCService components descend.
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>Handle</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td>LoginPrompt</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td>Server</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td>ServiceParamBySPB</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>ServiceStart</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttach</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>

Properties of the `TCustomIBCServices` class.
For a complete list of the **TCustomIBCService** class members, see the **TCustomIBCService Members** topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td>ServiceParamBySPB</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>LoginPrompt</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td>Server</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
</tbody>
</table>

### See Also

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

© 1997-2021 Devart. All Rights Reserved.
Syntax

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

Remarks

Use the `Active` property to set the service to active (True) or the default, inactive (False).

5.13.1.1.2.2 Handle Property

Used to return the database handle.

Class

`TCustomIBCSерvice`

Syntax

```plaintext
property Handle: TISC_SVC_HANDLE;
```

Remarks

Use the `Handle` property to return the database handle.

5.13.1.1.2.3 LoginPrompt Property

Used to display a login prompt before attaching to a database.

Class

`TCustomIBCService`

Syntax

```plaintext
property LoginPrompt: Boolean default DefValLoginPrompt;
```

Remarks

Use the `LoginPrompt` property to display (or not display) a login prompt before attaching to a database.
5.13.1.1.2.4  Params Property

Used to set or return database parameters.

Class

TCustomIBCSERVICE

Syntax

```
property Params: TStrings;
```

Remarks

Use the Params property to set or return database parameters.

5.13.1.1.2.5  Protocol Property

Used to select the network protocol.

Class

TCustomIBCSERVICE

Syntax

```
property Protocol: TIBCProtocol default DefValProtocol;
```

Remarks

Use the Protocol property to select the network protocol.

5.13.1.1.2.6  Server Property

Used to set the name of the server on which the services are to be run.

Class
TCustomIBCService

Syntax

```hashtag
property Server: string;
```

Remarks

Use the Server property to set the name of the server on which the services are to be run.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.1.2.7 ServiceParamBySPB Property(Indexer)

Used to return and sets SPB parameters.

Class

TCustomIBCService

Syntax

```hashtag
property ServiceParamBySPB[const Idx: Integer]: string;
```

Parameters

`Idx`

Holds the index of parameter.

Remarks

Use the ServiceParamBySPB property to inspect and set SPB parameters.

For example, DBParamBySPB[isc_SPB_user_name] sets and inspects the user name.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.1.3 Methods

Methods of the TCustomIBCService class.

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

Public

© 2021 Devart
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>ServiceStart</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

See Also
- TCustomIBCService Class
- TCustomIBCService Class Members

5.13.1.1.3.1 Attach Method

Attaches to the database.

Class
TCustomIBCService

Syntax

```
procedure Attach;
```

Remarks
Call the Attach method to attach to the database.

5.13.1.1.3.2 Detach Method

Detaches from the database.

Class
TCustomIBCService

Syntax

```
procedure Detach;
```
Remarks

Call the Detach method to detach from the database.

Class

TCustomIBCService

Syntax

```
procedure ServiceStart; virtual;
```

Events

Events of the TCustomIBCService class.

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

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttach</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>

See Also

- TCustomIBCService Class
- TCustomIBCService Class Members
5.13.1.1.4.1 OnAttach Event

Occurs when the database is attached.

Class

TCustomIBCService

Syntax

property OnAttach: TNotifyEvent;

Remarks

Write an OnAttach event handler to take specific actions when a database is attached. If an exception is raised in this event, the database is not attached.

5.13.1.2 TIBCBackupRestoreService Class

TIBCBackupRestoreService is the base class from which TIBCBackupService and TIBCRestoreService are derived.

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

Unit

IBCAadmin

Syntax

TIBCBackupRestoreService = class(TIBCControlAndQueryService);

Remarks

TIBCBackupRestoreService is the base class from which TIBCBackupService and TIBCRestoreService are derived.

Inheritance Hierarchy

TCustomIBCService

TIBCControlAndQueryService

TIBCBackupRestoreService
### TIBCBackupRestoreService class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from TCustomIBCServices)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BackupFile</strong></td>
<td>Holds the path of the backup file name.</td>
</tr>
<tr>
<td><strong>BufferSize</strong> (inherited from TIBCControlAndQueryService)</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>Eof</strong> (inherited from TIBCControlAndQueryService)</td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from TCustomIBCServices)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from TCustomIBCServices)</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>NBackupLevel</strong></td>
<td>Used to set backup level for nBackup.</td>
</tr>
<tr>
<td><strong>NBackupOptions</strong></td>
<td>Used to set backup options for nBackup.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from TCustomIBCServices)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> (inherited from TCustomIBCServices)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>Server</strong> (inherited from TCustomIBCServices)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td><strong>ServiceParamBySPB</strong> (inherited from TCustomIBCServices)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
<tr>
<td><strong>UseNBackup</strong></td>
<td>Used to enable or disable using nBackup service.</td>
</tr>
<tr>
<td><strong>Verbose</strong></td>
<td>Used to set or return the backup or restore in the verbose mode.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach (inherited from TCustomIBCSERVICE)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCSERVICE)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>GETNEXTCHUNK (inherited from TIBCCONTROLANDQUERYSERVICE)</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td>GETNEXTLINE (inherited from TIBCCONTROLANDQUERYSERVICE)</td>
<td>Returns the next line of data.</td>
</tr>
<tr>
<td>SERVICESTART (inherited from TCustomIBCSERVICE)</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONATTACH (inherited from TCustomIBCSERVICE)</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>

Properties of the TIBCBackupRestoreService class.

For a complete list of the TIBCBackupRestoreService class members, see the TIBCBackupRestoreService Members topic.
### NBackupOptions
Used to set backup options for nBackup.

### ServiceParamBySPB (inherited from TCustomIBCService)
Used to return and sets SPB parameters.

### UseNBackup
Used to enable or disable using nBackup service.

### Verbose
Used to set or return the backup or restore in the verbose mode.

## Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from TCustomIBCService)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BufferSize</strong> (inherited from TIBCCtrlAndQueryService)</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from TCustomIBCService)</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from TCustomIBCService)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> (inherited from TCustomIBCService)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>Server</strong> (inherited from TCustomIBCService)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
</tbody>
</table>

## See Also
- TIBCBackupRestoreService Class
- TIBCBackupRestoreService Class Members

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.13.1.2.2.1 BackupFile Property

Holds the path of the backup file name.

Class
TIBCBackupRestoreService

Syntax

```
property BackupFile: TStrings;
```

Remarks

The BackupFile property holds the path of the backup file name.

5.13.1.2.2.2  NBackupLevel Property

Used to set backup level for nBackup.

Class

TIBCBackupRestoreService

Syntax

```
property NBackupLevel: integer default 0;
```

Remarks

Use the NBackupLevel property to set backup level for nBackup. This property is used only when UseNBackup = True.

5.13.1.2.2.3  NBackupOptions Property

Used to set backup options for nBackup.

Class

TIBCBackupRestoreService

Syntax

```
property NBackupOptions: TIBCNBackupOptions default [];
```

Remarks
Use the NBackupOptions property to set backup options for nBackup. This property is used only when UseNBackup = True.

5.13.1.2.2.4 UseNBackup Property

Used to enable or disable using nBackup service.

Class
TIBCBackupRestoreService

Syntax

```
property UseNBackup: boolean default False;
```

Remarks

Use the UseNBackup property to enable or disable using nBackup service.

Set this property to True to use nBackup service instead of the standard backup/restore service.

Note: nBackup service is supported starting with Firebird 2.5.

5.13.1.2.2.5 Verbose Property

Used to set or return the backup or restore in the verbose mode.

Class
TIBCBackupRestoreService

Syntax

```
property Verbose: Boolean default False;
```

Remarks

Use the Verbose property to set or return the backup or restore in the verbose mode.
5.13.1.3 TIBCBackupService Class

Used to backup a database.

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

Unit
IBCAdmin

Syntax

TIBCBackupService = class(TIBCBackupRestoreService);

Remarks

Use a TIBCBackupService object to backup a database.

Inheritance Hierarchy

TCustomIBCService
  TIBCControlAndQueryService
    TIBCBackupRestoreService
      TIBCBackupService

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active (inherited from TCustomIBCService)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>BackupFile</td>
<td>Used to set or return the backup file name.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BlockingFactor</td>
<td>Used to set the blocking factor for the tape device as an integer.</td>
</tr>
<tr>
<td>BufferSize</td>
<td>(inherited from <strong>TIBCControlAndQueryService</strong>) Used to set or return the buffer size.</td>
</tr>
<tr>
<td>Database</td>
<td>Used to set or return the database name.</td>
</tr>
<tr>
<td>Eof</td>
<td>(inherited from <strong>TIBCControlAndQueryService</strong>) Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td>Handle</td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to return the database handle.</td>
</tr>
<tr>
<td>LoginPrompt</td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td>NBackupLevel</td>
<td>(inherited from <strong>TIBCBackupRestoreService</strong>) Used to set backup level for nBackup.</td>
</tr>
<tr>
<td>NBackupOptions</td>
<td>(inherited from <strong>TIBCBackupRestoreService</strong>) Used to set backup options for nBackup.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of a TIBCBackupService object.</td>
</tr>
<tr>
<td>Params</td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to set or return database parameters.</td>
</tr>
<tr>
<td>Protocol</td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to select the network protocol.</td>
</tr>
<tr>
<td>Server</td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td>ServiceParamBySPB</td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to return and sets SPB parameters.</td>
</tr>
<tr>
<td>UseNBackup</td>
<td>(inherited from <strong>TIBCBackupRestoreService</strong>) Used to enable or disable using nBackup service.</td>
</tr>
<tr>
<td>Verbose</td>
<td>Used to set or return the backup in the verbose mode.</td>
</tr>
</tbody>
</table>

**Methods**
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>GetNextChunk</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td>GetNextLine</td>
<td>Returns the next line of data.</td>
</tr>
<tr>
<td>ServiceStart</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttach</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>

5.13.1.3.2 Properties

Properties of the `TIBCBackupService` class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eof</td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td>Handle</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td>NBackupLevel</td>
<td>Used to set backup level for nBackup.</td>
</tr>
<tr>
<td>NBackupOptions</td>
<td>Used to set backup options for nBackup.</td>
</tr>
<tr>
<td><strong>ServiceParamBySPB</strong> (inherited from <strong>TCustomIBCServices</strong>)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
<tr>
<td><strong>UseNBackup</strong> (inherited from <strong>TIBCBackupRestoreService</strong>)</td>
<td>Used to enable or disable using nBackup service.</td>
</tr>
</tbody>
</table>

**Published**

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from <strong>TCustomIBCServices</strong>)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BackupFile</strong></td>
<td>Used to set or return the backup file name.</td>
</tr>
<tr>
<td><strong>BlockingFactor</strong></td>
<td>Used to set the blocking factor for the tape device as an integer.</td>
</tr>
<tr>
<td><strong>BufferSize</strong> (inherited from <strong>TIBCControlAndQueryService</strong>)</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Used to set or return the database name.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from <strong>TCustomIBCServices</strong>)</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of a TIBCBackupService object.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from <strong>TCustomIBCServices</strong>)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> (inherited from <strong>TCustomIBCServices</strong>)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>Server</strong> (inherited from <strong>TCustomIBCServices</strong>)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td><strong>Verbose</strong></td>
<td>Used to set or return the backup in the verbose mode.</td>
</tr>
</tbody>
</table>

**See Also**
- TIBCBackupService Class
- TIBCBackupService Class Members

© 2021 Devart
5.13.1.3.2.1 BackupFile Property

Used to set or return the backup file name.

Class

TIBCBackupService

Syntax

```pascal
property BackupFile: TStrings;
```

Remarks

Use the BackupFile property to set or return the backup file name.

5.13.1.3.2.2 BlockingFactor Property

Used to set the blocking factor for the tape device as an integer.

Class

TIBCBackupService

Syntax

```pascal
property BlockingFactor: Integer default 0;
```

Remarks

Use the BlockingFactor property to set the blocking factor for the tape device as an integer.

5.13.1.3.2.3 Database Property

Used to set or return the database name.

Class
**TIBCBackupService**

**Syntax**

```delphi
property Database: string;
```

**Remarks**

Use the Database property to set or return the database name to set properties on.

---

5.13.1.3.2.4 OPTIONS Property

Used to specify the behaviour of a TIBCBackupService object.

**Class**

**TIBCBackupService**

**Syntax**

```delphi
property Options: TIBCBackupOptions default [];
```

**Remarks**

Set the properties of Options to specify the behaviour of a TIBCBackupService object.

---

5.13.1.3.2.5 VERBOSE Property

Used to set or return the backup in the verbose mode.

**Class**

**TIBCBackupService**

**Syntax**

```delphi
property Verbose: Boolean;
```

**Remarks**
Use the Verbose property to set or return the backup in the verbose mode.

5.13.1.4 TIBCConfigParams Class

TIBCConfigParams holds the configuration information about an InterBase server.

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

Unit
IBCAdmin

Syntax

TIBCConfigParams = class(System.TObject);

Remarks

TIBCConfigParams holds the configuration information about an InterBase server. The TIBCConfigParams type stores server configuration settings.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseLocation</td>
<td>Used to determine the base location.</td>
</tr>
<tr>
<td>LockFileLocation</td>
<td>Used to determine the lock file location.</td>
</tr>
<tr>
<td>MessageFileLocation</td>
<td>Used to determine the message file location.</td>
</tr>
<tr>
<td>SecurityDatabaseLocation</td>
<td>Used to determine the security database location.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
Properties of the TIBCConfigParams class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseLocation</td>
<td>Used to determine the base location.</td>
</tr>
<tr>
<td>LockFileLocation</td>
<td>Used to determine the lock file location.</td>
</tr>
<tr>
<td>MessageFileLocation</td>
<td>Used to determine the message file location.</td>
</tr>
<tr>
<td>SecurityDatabaseLocation</td>
<td>Used to determine the security database location.</td>
</tr>
</tbody>
</table>

See Also
- TIBCConfigParams Class
- TIBCConfigParams Class Members

5.13.1.4.2.1 BaseLocation Property

Used to determine the base location.

Class

TIBCConfigParams

Syntax

```csharp
property BaseLocation: string;
```

Remarks

Use the BaseLocation property to determine the base location.
5.13.1.4.2.2 LockFileLocation Property

Used to determine the lock file location.

Class

TIBConfigParams

Syntax

```
property LockFileLocation: string;
```

Remarks

Use the LockFileLocation property to determine the lock file location.

5.13.1.4.2.3 MessageFileLocation Property

Used to determine the message file location.

Class

TIBConfigParams

Syntax

```
property MessageFileLocation: string;
```

Remarks

Use the MessageFileLocation to determine the message file location.

5.13.1.4.2.4 SecurityDatabaseLocation Property

Used to determine the security database location.

Class
**TIBCConfigParams**

Syntax

```property`` SecurityDatabaseLocation: `string``;``

Remarks

Use the SecurityDatabaseLocation to determine the security database location.

---

**5.13.1.5 TIBCConfigService Class**

Use a TIBConfigService object to configure database parameters.

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

**Unit**

`IBCAdmin`

Syntax

```TIBConfigService = class(TCustomIBCServic)``;``

Remarks

Use a TIBConfigService component to configure database parameters, including page buffers, access mode, and sweep interval.

**Note:** This feature is available in InterBase only.

**Inheritance Hierarchy**

`TCustomIBCServic`  
`TIBCConfigService`

---

**5.13.1.5.1 Members**

[TIBCConfigService class overview](#)
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Used to set or return the database name.</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td><strong>JournalInformation</strong></td>
<td>Holds information about journal system.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>Server</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td><strong>ServiceParamBySPB</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
<tr>
<td><strong>Transaction</strong></td>
<td>Used to determine the transaction under which the query of this dataset executes.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ActivateShadow</strong></td>
<td>Activates the database shadow.</td>
</tr>
<tr>
<td><strong>AlterJournal</strong></td>
<td>Alters a pre-existing journal system.</td>
</tr>
<tr>
<td><strong>Attach</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td><strong>BringDatabaseOnline</strong></td>
<td>Brings a database online.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CreateJournal</td>
<td>Creates a journal.</td>
</tr>
<tr>
<td>CreateJournalArchive</td>
<td>Creates an archive.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>DisableFlush</td>
<td>Disables database flushing.</td>
</tr>
<tr>
<td>DropJournal</td>
<td>Drops a journal system.</td>
</tr>
<tr>
<td>DropJournalArchive</td>
<td>Drops an archive.</td>
</tr>
<tr>
<td>FlushDatabase</td>
<td>Flushes a database.</td>
</tr>
<tr>
<td>GetJournalInformation</td>
<td>Retrieves journaling information.</td>
</tr>
<tr>
<td>ReclaimMemory</td>
<td>Used to reclaim memory.</td>
</tr>
<tr>
<td>ServiceStart</td>
<td>Is not currently used by TIBCConfigService.</td>
</tr>
<tr>
<td>SetAsyncMode</td>
<td>Changes the database write mode to asynchronous (buffered writes).</td>
</tr>
<tr>
<td>SetDBSqlDialect</td>
<td>Sets the SQL dialect for the database.</td>
</tr>
<tr>
<td>SetFlushInterval</td>
<td>Sets the interval for database flushing</td>
</tr>
<tr>
<td>SetGroupCommit</td>
<td>Sets group commit.</td>
</tr>
<tr>
<td>SetLingerInterval</td>
<td>Sets the linger interval.</td>
</tr>
<tr>
<td>SetPageBuffers</td>
<td>Sets the number of database page buffers.</td>
</tr>
<tr>
<td>SetReadOnly</td>
<td>Sets the database to read only.</td>
</tr>
<tr>
<td>SetReclaimInterval</td>
<td>Sets the reclaim interval.</td>
</tr>
<tr>
<td>SetReserveSpace</td>
<td>Reserves space for versioning.</td>
</tr>
<tr>
<td>SetSweepInterval</td>
<td>Sets the sweep interval for the database.</td>
</tr>
<tr>
<td>ShutdownDatabase</td>
<td>Shuts down the database.</td>
</tr>
<tr>
<td>SweepDatabase</td>
<td>Performs a database sweep.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### OnAttach

*(inherited from TCustomIBCService)*

Occurs when the database is attached.

#### Properties

**Properties of the TIBConfigService class.**

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle (inherited from TCustomIBCService)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td>ServiceParamBySPB (inherited from TCustomIBCService)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active (inherited from TCustomIBCService)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify the connection in which the dataset will be executed.</td>
</tr>
<tr>
<td>Database</td>
<td>Used to set or return the database name.</td>
</tr>
<tr>
<td>JournalInformation</td>
<td>Holds information about journal system.</td>
</tr>
<tr>
<td>LoginPrompt (inherited from TCustomIBCService)</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td>Params (inherited from TCustomIBCService)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td>Protocol (inherited from TCustomIBCService)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td>Server (inherited from TCustomIBCService)</td>
<td>Used to set the name of the server on which the services are executed.</td>
</tr>
</tbody>
</table>
5.13.1.5.2.1 Connection Property

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

Class

**TIBCConfigService**

Syntax

```
property Connection: TIBCConnection;
```

Remarks

Use the Connection property to specify the connection in which the service will be executed. If connection is not connected, the Open method calls Connection.Connect.

**Note:** You should install InterBase 6 to use this feature.

5.13.1.5.2.2 Database Property

Used to set or return the database name.

Class

**TIBCConfigService**

Syntax
property Database: string;

Remarks
Use the Database property to set or return the database name to set properties on.

Note: You should install InterBase 6 to use this feature.

5.13.1.5.2.3 JournalInformation Property

Holds information about journal system.

Class
TIBConfigService

Syntax
property JournalInformation: TIBCJournalInformation;

Remarks
The JournalInformation property holds information about a journal system. Gives access to the underlying IBCJournalInformation field.

Note: You should install InterBase 6 to use this feature.

5.13.1.5.2.4 Transaction Property

Used to determine the transaction under which the query of this dataset executes.

Class
TIBConfigService

Syntax
property Transaction: TIBCTransaction;

Remarks
Use the Transaction property to determine the transaction under which the query of this dataset executes. You can separately set transaction for executing modifying queries with the `TCustomIBCDataset.UpdateTransaction` property. By default the Transaction and the UpdateTransaction properties are the same.

Methods of the `TIBConfigService` class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ActivateShadow</code></td>
<td>Activates the database shadow.</td>
</tr>
<tr>
<td><code>AlterJournal</code></td>
<td>Alters a pre-existing journal system.</td>
</tr>
<tr>
<td><code>Attach</code> (inherited from <code>TCustomIBCService</code>)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td><code>BringDatabaseOnline</code></td>
<td>Brings a database online.</td>
</tr>
<tr>
<td><code>CreateJournal</code></td>
<td>Creates a journal.</td>
</tr>
<tr>
<td><code>CreateJournalArchive</code></td>
<td>Creates an archive.</td>
</tr>
<tr>
<td><code>Detach</code> (inherited from <code>TCustomIBCService</code>)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td><code>DisableFlush</code></td>
<td>Disables database flushing.</td>
</tr>
<tr>
<td><code>DropJournal</code></td>
<td>Drops a journal system.</td>
</tr>
<tr>
<td><code>DropJournalArchive</code></td>
<td>Drops an archive.</td>
</tr>
<tr>
<td><code>FlushDatabase</code></td>
<td>Flushes a database.</td>
</tr>
<tr>
<td><code>GetJournalInformation</code></td>
<td>Retrieves journaling information.</td>
</tr>
<tr>
<td><code>ReclaimMemory</code></td>
<td>Used to reclaim memory.</td>
</tr>
<tr>
<td><code>ServiceStart</code></td>
<td>Is not currently used by <code>TIBConfigService</code>.</td>
</tr>
<tr>
<td><code>SetAsyncMode</code></td>
<td>Changes the database write mode to asynchronous</td>
</tr>
</tbody>
</table>
### ActivateShadow Method

Activates the database shadow.

**Class**

**TIBConfigService**

**Syntax**

```plaintext
procedure ActivateShadow;
```

**Remarks**

Call the ActivateShadow method to activate the database shadow.

---

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetDBSqlDialect</td>
<td>Sets the SQL dialect for the database.</td>
</tr>
<tr>
<td>SetFlushInterval</td>
<td>Sets the interval for database flushing.</td>
</tr>
<tr>
<td>SetGroupCommit</td>
<td>Sets group commit.</td>
</tr>
<tr>
<td>SetLingerInterval</td>
<td>Sets the linger interval.</td>
</tr>
<tr>
<td>SetPageBuffers</td>
<td>Sets the number of database page buffers.</td>
</tr>
<tr>
<td>SetReadOnly</td>
<td>Sets the database to read only.</td>
</tr>
<tr>
<td>SetReclaimInterval</td>
<td>Sets the reclaim interval.</td>
</tr>
<tr>
<td>SetReserveSpace</td>
<td>Reserves space for versioning.</td>
</tr>
<tr>
<td>SetSweepInterval</td>
<td>Sets the sweep interval for the database.</td>
</tr>
<tr>
<td>ShutdownDatabase</td>
<td>Shuts down the database.</td>
</tr>
<tr>
<td>SweepDatabase</td>
<td>Performs a database sweep.</td>
</tr>
</tbody>
</table>
InterBase 6 lets you recover a database in case of disk failure, network failure, or accidental deletion of the database. This recovery method is known as disk shadowing, or simply shadowing. Before you can activate shadowing, you must first create a database shadow, as discussed in 'Shadowing' and 'Creating a shadow' in the InterBase 6 Operations Guide.

**Note:** You should install InterBase 6 to use this feature.

### 5.13.1.5.3.2 AlterJournal Method

Alters a pre-existing journal system.

**Class**

TIBCConfigService

**Syntax**

```plaintext
procedure AlterJournal;
```

**Remarks**

Call the AlterJournal method to alter a pre-existing journal system.

**Note:** You should install InterBase 6 to use this feature.

### 5.13.1.5.3.3 BringDatabaseOnline Method

Brings a database online.

**Class**

TIBCConfigService

**Syntax**

```plaintext
procedure BringDatabaseOnline(Options: TIBCShutdownOptions = soNormal);
```

**Parameters**

Options
Remarks

Call the BringDatabaseOnline method to bring a database online.

**Note:** You should install InterBase 6 to use this feature.

5.13.1.5.3.4 CreateJournal Method

Creates a journal.

Class

**TIBCConfigService**

Syntax

```pascal
procedure CreateJournal;
```

Remarks

Call the CreateJournal method to create a journal based on JournalInformation.

**Note:** You should install InterBase 6 to use this feature.

5.13.1.5.3.5 CreateJournalArchive Method

Creates an archive.

Class

**TIBCConfigService**

Syntax

```pascal
procedure CreateJournalArchive(const Directory: string);
```

Parameters

*Directory*

Holds the directory in which the archive is situated.
Remarks
Call the CreateJournalArchive method to create an archive.

Note: You should install InterBase 6 to use this feature.

Disables database flushing.

Class
TIBConfigService

Syntax

```plaintext
procedure DisableFlush;
```

Remarks
Call the DisableFlush method to disable database flushing.

Note: You should install InterBase 6 to use this feature.

Drops a journal system.

Class
TIBConfigService

Syntax

```plaintext
procedure DropJournal;
```

Remarks
Call the DropJournal method to drop a journal system.
5.13.1.5.3.8  DropJournalArchive Method

Drops an archive.

Class

TIBConfigService

Syntax

procedure DropJournalArchive;

Remarks

Call the DropJournalArchive to drop an archive.

Note: You should install InterBase 6 to use this feature.

5.13.1.5.3.9  FlushDatabase Method

Flushes a database.

Class

TIBConfigService

Syntax

procedure FlushDatabase;

Remarks

Call the FlushDatabase method to flush a database.

Note: You should install InterBase 6 to use this feature.
5.13.1.5.3.10 GetJournalInformation Method

Retrieves journaling information.

Class
TIBConfigService

Syntax

```delphi
procedure GetJournalInformation;
```

Remarks
Call the GetJournalInformation method to retrieve journaling information for this database and stores it in the `JournalInformation` property.

**Note:** You should install InterBase 6 to use this feature.

5.13.1.5.3.11 ReclaimMemory Method

Used to reclaim memory.

Class
TIBConfigService

Syntax

```delphi
procedure ReclaimMemory;
```

Remarks
Use the ReclaimMemory property to reclaim memory.

**Note:** You should install InterBase 6 to use this feature.
5.13.1.5.3.12 ServiceStart Method

Is not currently used by TIBCConfigService.

Class
TIBCConfigService

Syntax

procedure ServiceStart; override;

Remarks
Do not use ServiceStart. TIBCConfigService does not currently use this method.

Note: You should install InterBase 6 to use this feature.

© 1997-2021
Devart. All Rights Reserved.

5.13.1.5.3.13 SetAsyncMode Method

Changes the database write mode to asynchronous (buffered writes).

Class
TIBCConfigService

Syntax

procedure SetAsyncMode(Value: Boolean);

Parameters

Value
True, if the database write mode is changed to asynchronous.

Remarks
Set the SetAsyncMode method to True to change the database write mode to asynchronous (buffered writes).

InterBase 6 allows you to write to databases in both synchronous and asynchronous modes. In synchronous mode, the database writes are forced. In asynchronous mode, the database writes are buffered.

For more information, refer to 'Forced writes vs. buffered writes' in the InterBase 6 Operations
5.13.1.5.3.14  SetDBSqlDialect Method

Sets the SQL dialect for the database.

Class

TIBCConfigService

Syntax

```
procedure SetDBSqlDialect(Value: Integer);
```

Parameters

- **Value**
  - Holds the appropriate values of SQL dialect (valid values are 1, 2, and 3).

Remarks

Call the SetDBSqlDialect method to set the SQL dialect for the database. Valid integer values are 1, 2, and 3.

**Note:** You should install InterBase 6 to use this feature.

© 1997-2021
Devart. All Rights Reserved.  

5.13.1.5.3.15  SetFlushInterval Method

Sets the interval for database flushing

Class

TIBCConfigService

Syntax

```
procedure SetFlushInterval(Value: Integer);
```

Parameters

- **Value**
  - Holds the appropriate values of SQL dialect (valid values are 1, 2, and 3).

Remarks

© 1997-2021
Devart. All Rights Reserved.
**Value**
Holds the interval.

**Remarks**
Call the SetFlushInterval method to set the interval for database flushing.

**Note**: You should install InterBase 6 to use this feature.

---

### 5.13.1.5.3.17 SetLingerInterval Method

Sets the linger interval.

**Class**
**TIBCConfigService**

**Syntax**

```plaintext
procedure SetLingerInterval(Value: Integer);
```

**Parameters**

**Value**

**Remarks**
Call the SetLingerInterval method to set the linger interval.

**Note**: You should install InterBase 6 to use this feature.
5.13.1.5.3.18  SetPageBuffers Method

Sets the number of database page buffers.

Class

TIBCConfigService

Syntax

procedure SetPageBuffers(Value: Integer);

Parameters

Value
  Holds the number of database page buffers.

Remarks

Call the SetPageBuffers method to set the number of database page buffers.

When a program establishes a connection to a database, InterBase allocates system memory to use as a private buffer. The buffers are used to store accessed database pages to speed performance. The number of buffers assigned determines how many simultaneous database pages it can have access to in the memory pool. Buffers remain assigned until a program finishes with a database.

For more information on page buffers, refer to 'Setting database cache buffers' in the InterBase 6 Programmer's Guide.

Note: You should install InterBase 6 to use this feature.
Reserved.

5.13.1.5.3.19 SetReadOnly Method

Sets the database to read only.

Class

TIBCConfigService

Syntax

procedure SetReadOnly(Value: Boolean);

Parameters

Value
True, if the database is set to read only.

Remarks

Call the SetReadOnly method to set the database to read only.

Access mode specifies the type of access a transaction has for the table it uses. There are two possible modes: read-only and read-write.

Read-only specifies that a transaction can read data from a table, but cannot insert, update, or delete table data. Read-write specifies that a transaction can select, insert, update, and delete table data. This is the default setting if none is specified.

Tip: You should specify the transaction's access mode even if is read-write. It makes the application's source code easier to read and debug, because the program's intentions are clearly spelled out.

Note: You should install InterBase 6 to use this feature.

© 1997-2021
Devart. All Rights Reserved.

5.13.1.5.3.20 SetReclaimInterval Method

Sets the reclaim interval.

Class

TIBCConfigService
Syntax

```pascal
procedure SetReclaimInterval(Value: Integer);
```

**Parameters**

*Value*

Holds the reclaim interval.

Remarks

Call the SetReclaimInterval method to set the reclaim interval.

**Note:** You should install InterBase 6 to use this feature.

© 1997-2021

Devart. All Rights Reserved.

5.13.1.5.3.21 SetReserveSpace Method

Reserves space for versioning.

**Class**

*TIBConfigService*

Syntax

```pascal
procedure SetReserveSpace(Value: Boolean);
```

**Parameters**

*Value*

If True, space for versioning is reserved.

Remarks

Set SetReserveSpace to true to reserve space on the data page for versioning.

**Note:** You should install InterBase 6 to use this feature.

© 1997-2021

Devart. All Rights Reserved.
5.13.1.5.3.22 SetSweepInterval Method

Sets the sweep interval for the database.

Class

TIBCConfigService

Syntax

```pascal
procedure SetSweepInterval(Value: Integer);
```

Parameters

Value

Holds the sweep interval.

Remarks

Call SetSweepInterval set the sweep interval for the database. The sweep interval refers to the number of transactions between database sweeps.

To turn off database sweeps, set the sweep interval to 0.

Sweeping a database is a systematic way of removing outdated records from the database. Periodic sweeping keeps the database from getting too large. However, sweeping can also slow performance. For more information, refer to 'Setting the sweep interval' in the InterBase 6 Operations Guide.

**Note:** You should install InterBase 6 to use this feature.

5.13.1.5.3.23 ShutdownDatabase Method

Shuts down the database.

Class

TIBCConfigService

Syntax

```pascal
procedure ShutdownDatabase(Mode: TIBCShutdownMode; Wait: Integer; Options: TIBCShutdownOptions = soNormal);
```

Parameters
Mode

Wait
Holds the number of seconds to wait before shutting the database.

Options

Remarks
Call the ShutdownDatabase method to shut down the database after a specified number of seconds.

Note: You should install InterBase 6 to use this feature.

5.13.1.5.3.24  SweepDatabase Method

Performs a database sweep.

Class

TIBCConfigService

Syntax

procedure SweepDatabase;

Remarks
Call the SweepDatabase method to perform a database sweep.

Note: You should install InterBase 6 to use this feature.

5.13.1.6  TIBCControlAndQueryService Class

TIBCControlAndQueryService is the base class from which the log, statistical, validation, security, and backup and restore TIBCService components descend.

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

Unit

IBCAadmin
Syntax

```delphi
TIBCControlAndQueryService = class(TCustomIBCServicet);
```

Remarks

TIBCControlAndQueryService is the base class from which the log, statistical, validation, security, and backup and restore TIBCService components descend. The service is unstructured because the output is in an unformatted text file.

Inheritance Hierarchy

```
TCustomIBCServicet
  TIBCControlAndQueryServicet
```

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from <code>TCustomIBCServicet</code>)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BufferSize</strong></td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>Eof</strong></td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from <code>TCustomIBCServicet</code>)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from <code>TCustomIBCServicet</code>)</td>
<td>Used to dispaly a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from <code>TCustomIBCServicet</code>)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> (inherited from <code>TCustomIBCServicet</code>)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>Server</strong> (inherited from <code>TCustomIBCServicet</code>)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
</tbody>
</table>
### ServiceParamBySPB (inherited from TCustomIBCService)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>GetNextChunk</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td>GetNextLine</td>
<td>Returns the next line of data.</td>
</tr>
<tr>
<td>ServiceStart</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttach (inherited from TCustomIBCService)</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>

### Properties

Properties of the TIBControlAndQueryService class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eof</td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td>Handle (inherited from TCustomIBCService)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td>ServiceParamBySPB (inherited from TCustomIBCService)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>
Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active (inherited from TCustomIBCService)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>BufferSize</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td>LoginPrompt (inherited from TCustomIBCService)</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td>Params (inherited from TCustomIBCService)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td>Protocol (inherited from TCustomIBCService)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td>Server (inherited from TCustomIBCService)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
</tbody>
</table>

See Also
- TIBControlAndQueryService Class
- TIBControlAndQueryService Class Members

5.13.1.6.2.1 BufferSize Property

Used to set or return the buffer size.

Class
TIBControlAndQueryService

Syntax

```property``` BufferSize: Integer;

Remarks
Use the BufferSize property to set or return the buffer size.
5.13.1.6.2.2 Eof Property

Used to determine whether the end of the file has been reached.

Class

\texttt{TIBCControlAndQueryService}

Syntax

\begin{verbatim}
property Eof: Boolean;
\end{verbatim}

Remarks

Use the Eof property to determine whether the end of the file has been reached.

\textbf{Note:} You should install InterBase 6 to use this feature.

5.13.1.6.3 Methods

Methods of the \texttt{TIBCControlAndQueryService} class.

For a complete list of the \texttt{TIBCControlAndQueryService} class members, see the \texttt{TIBCControlAndQueryService Members} topic.

Public

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Name} & \textbf{Description} \\
\hline
\underline{Attach} (inherited from \texttt{TCustomIBCService}) & Attaches to the database. \\
\hline
\underline{Detach} (inherited from \texttt{TCustomIBCService}) & Detaches from the database. \\
\hline
\underline{GetNextChunk} & Returns the next chunk of data. \\
\hline
\underline{GetNextLine} & Returns the next line of data. \\
\hline
\underline{ServiceStart} (inherited from \texttt{TCustomIBCService}) & Starts the service. \\
\hline
\end{tabular}
\end{table}
5.13.1.6.3.1 GetNextChunk Method

Returns the next chunk of data.

Class

TIBCControlAndQueryService

Syntax

function GetNextChunk: string;

Remarks

Call the GetNextChunk method to get the next chunk of data.

Note: You should install InterBase 6 to use this feature.

5.13.1.6.3.2 GetNextLine Method

Returns the next line of data.

Class

TIBCControlAndQueryService

Syntax

function GetNextLine: string;

Remarks

Call the GetNextLine method to get the next line of data.

Note: You should install InterBase 6 to use this feature.
5.13.1.7  TIBCDatabaseInfo Class

Describes an InterBase database.

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

Unit
IBCAdmin

Syntax

TIBCDatabaseInfo = class(System.TObject);

Remarks

TIBCDatabaseInfo describes an InterBase database.

5.13.1.7.1  Members

TIBCDatabaseInfo class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbName</td>
<td>Holds the name of the database.</td>
</tr>
<tr>
<td>NoOfAttachments</td>
<td>Holds the number of attachments.</td>
</tr>
</tbody>
</table>

5.13.1.7.2  Properties

Properties of the TIBCDatabaseInfo class.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbName</td>
<td>Holds the name of the database.</td>
</tr>
<tr>
<td>NoOfAttachments</td>
<td>Holds the number of attachments.</td>
</tr>
</tbody>
</table>

See Also

- TIBCDatabaseInfo Class
- TIBCDatabaseInfo Class Members

5.13.1.7.2.1  DbName Property

Holds the name of the database.

Class

TIBCDatabaseInfo

Syntax

```property
DbName: TStringDynArray;
```

Remarks

The DbName property holds the name of the database.

5.13.1.7.2.2  NoOfAttachments Property

Holds the number of attachments.

Class

TIBCDatabaseInfo

Syntax
5.13.1.8 TIBCJournalInformation Class

A class used to access the TIBCConfigService component properties.

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

Unit
IBCAdmin

Syntax

TIBCJournalInformation = class(TComponent);

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckpointInterval</td>
<td>Used to determine the number of seconds between database checkpoints.</td>
</tr>
<tr>
<td>CheckpointLength</td>
<td>Used to determine the number of journal pages to be written before initiating a database checkpoint.</td>
</tr>
</tbody>
</table>
### Properties of the TIBCJournalInformation class.

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

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CheckpointInterval</strong></td>
<td>Used to determine the number of seconds between database checkpoints.</td>
</tr>
<tr>
<td><strong>CheckpointLength</strong></td>
<td>Used to determine the number of journal pages to be written before initiating a database checkpoint.</td>
</tr>
<tr>
<td><strong>Directory</strong></td>
<td>Holds the name of the directory in which journal is situated.</td>
</tr>
<tr>
<td><strong>HasArchive</strong></td>
<td>Turns archiving on.</td>
</tr>
<tr>
<td><strong>HasJournal</strong></td>
<td>Turns journaling on.</td>
</tr>
<tr>
<td><strong>PageSize</strong></td>
<td>Determines the size of a journal page in bytes.</td>
</tr>
<tr>
<td><strong>TimestampName</strong></td>
<td>Determines whether or not to append the file creation timestamp to the base journal file name.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PageCache</td>
<td>Determines the number of journal buffers that will be allocated.</td>
</tr>
<tr>
<td>PageLength</td>
<td>Used to retrieve the page length.</td>
</tr>
<tr>
<td>PageSize</td>
<td>Determines the size of a journal page in bytes.</td>
</tr>
<tr>
<td>TimestampName</td>
<td>Determines whether or not to append the file creation timestamp to the base journal file name.</td>
</tr>
</tbody>
</table>

### See Also
- [TIBCJournalInformation Class](#)
- [TIBCJournalInformation Class Members](#)

### 5.13.1.8.2.1 CheckpointInterval Property

Used to determine the number of seconds between database checkpoints.

#### Class

[TIBCJournalInformation](#)

#### Syntax

```property
CheckpointInterval: Integer default 0;
```

#### Remarks

Use the CheckpointInterval property to determine the number of seconds between database checkpoints.

**Note:** If both CHECKPOINT LENGTH and CHECKPOINT INTERVAL are specified, whichever event occurs first will initiate a database checkpoint.
5.13.1.8.2.2 CheckpointLength Property

Used to determine the number of journal pages to be written before initiating a database checkpoint.

Class

TIBCJournalInformation

Syntax

```property` CheckpointLength: Integer default 500;```

Remarks

Use the CheckpointInterval property to determine the number of journal pages to be written before initiating a database checkpoint.

**Note:** If both CHECKPOINT LENGTH and CHECKPOINT INTERVAL are specified, whichever event occurs first will initiate a database checkpoint.

--

5.13.1.8.2.3 Directory Property

Holds the name of the directory in which journal is situated.

Class

TIBCJournalInformation

Syntax

```property` Directory: string;```

Remarks

The Directory property holds the name of the directory in which journal is situated.
5.13.1.8.2.4 HasArchive Property

Turns archiving on.

Class
TIBCJournalInformation

Syntax

```property
HasArchive: Boolean;
```

Remarks

Set the HasArchive method to True to turn archiving on.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.8.2.5 HasJournal Property

Turns journaling on.

Class
TIBCJournalInformation

Syntax

```property
HasJournal: Boolean;
```

Remarks

Set the HasJournal method to True to turn journaling on.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.8.2.6 PageCache Property

Determines the number of journal buffers that will be allocated.

Class
TIBCJournalInformation
Syntax

```
property PageCache: Integer default 100;
```

Remarks

The PageCache property determines the number of journal buffers that will be allocated. The size of each buffer is the same as the journal page size.

---

5.13.1.8.2.7 PageLength Property

Used to retrieve the page length.

Class

`TIBCJournalInformation`

Syntax

```
property PageLength: Integer default 500;
```

Remarks

Use the PageLength property to retrieve the page length.

---

5.13.1.8.2.8 PageSize Property

Determines the size of a journal page in bytes.

Class

`TIBCJournalInformation`

Syntax

```
property PageSize: Integer default 0;
```

Remarks

The PageSize property determines the size of a journal page in bytes. A journal page size
must be at least twice the size of a database page size. If a journal page size of less is specified, it will be rounded up to twice the database page size and a warning will be returned.

The journal page size need not be a power of 2.

5.13.1.8.2.9 TimestampName Property

Determines whether or not to append the file creation timestamp to the base journal file name.

Class

TIBCJournalInformation

Syntax

property TimestampName: Boolean default True;

Remarks

The TimestampName property determines whether or not to append the file creation timestamp to the base journal file name.

If this option is on, the base journal file name will be appended with a timestamp of the form: `<YYYY>_<MM>_DD>T<hh>_<mm>_ss>Z.<sequence-number>.journal`

5.13.1.9 TIBCLicenseInfo Class

Stores information about licensed users.

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

Unit

IBCAadmin

Syntax

TIBCLicenseInfo = class(System TObject);
Remarks

TIBCLicenseInfo stores information about licensed users.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desc</td>
<td>Used to list the description of each licensed user.</td>
</tr>
<tr>
<td>Id</td>
<td>Used to list the user Id for each licensed user.</td>
</tr>
<tr>
<td>Key</td>
<td>Used to list the license key for each licensed user.</td>
</tr>
<tr>
<td>LicensedUsers</td>
<td>Used to determine the number of users.</td>
</tr>
</tbody>
</table>

Properties of the TIBCLicenseInfo class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desc</td>
<td>Used to list the description of each licensed user.</td>
</tr>
<tr>
<td>Id</td>
<td>Used to list the user Id for each licensed user.</td>
</tr>
<tr>
<td>Key</td>
<td>Used to list the license key for each licensed user.</td>
</tr>
<tr>
<td>LicensedUsers</td>
<td>Used to determine the</td>
</tr>
</tbody>
</table>
5.13.1.9.2.1  Desc Property

Used to list the description of each licensed user.

Class

TIBCLicenseInfo

Syntax

```pascal
property Desc: TStringDynArray;
```

Remarks

The Desc property lists the description of each licensed user.

5.13.1.9.2.2  Id Property

Used to list the user Id for each licensed user.

Class

TIBCLicenseInfo

Syntax

```pascal
property Id: TStringDynArray;
```

Remarks

The Id property lists the user Id for each licensed user.
5.13.1.9.2.3  Key Property

Used to list the license key for each licensed user.

Class

TIBCLicenseInfo

Syntax

```pascal
property Key: TStringDynArray;
```

Remarks

The Key property lists the license key for each licensed user.

5.13.1.9.2.4  LicensedUsers Property

Used to determine the number of users.

Class

TIBCLicenseInfo

Syntax

```pascal
property LicensedUsers: integer;
```

Remarks

The LicensedUsers property indicates the number of users (the number of entries in the lists).

5.13.1.10 TIBCLicenseMaskInfo Class

Indicates the software activation certificate options enabled on the server.

For a list of all members of this type, see TIBCLicenseMaskInfo members.
Unit

IBCAdmin

Syntax

TIBCLicenseMaskInfo = class(System.TObject);

Remarks

TIBCLicenseMaskInfo indicates the software activation certificate options enabled on the server.

© 1997-2021 Devart. All Rights Reserved.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapabilityMask</td>
<td>Used to determine a bitmask representing the capabilities currently enabled on the server.</td>
</tr>
<tr>
<td>LicenseMask</td>
<td>Used to determine a bitmask representing the software activation certificate options currently enabled on the server.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.13.1.10.2 Properties

Properties of the TIBCLicenseMaskInfo class.

For a complete list of the TIBCLicenseMaskInfo class members, see the TIBCLicenseMaskInfo Members topic.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapabilityMask</td>
<td>Used to determine a bitmask representing the capabilities currently enabled on the server.</td>
</tr>
<tr>
<td>LicenseMask</td>
<td>Used to determine a bitmask representing the software activation certificate options currently enabled on the server.</td>
</tr>
</tbody>
</table>

See Also
- TIBCLicenseMaskInfo Class
- TIBCLicenseMaskInfo Class Members

5.13.1.10.2.1 CapabilityMask Property

Used to determine a bitmask representing the capabilities currently enabled on the server.

Class

TIBCLicenseMaskInfo

Syntax

```property`` CapabilityMask: integer;```

Remarks

Use the CapabilityMask to determine a bitmask representing the capabilities currently enabled on the server.

5.13.1.10.2.2 LicenseMask Property

Used to determine a bitmask representing the software activation certificate options currently enabled on the server.

Class
**TIBCLicenseMaskInfo**

**Syntax**

```delphi
property LicenseMask: integer;
```

**Remarks**

Use the LicenseMask property to determine a bitmask representing the software activation certificate options currently enabled on the server.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.11 **TIBCLicensingService Class**

TIBCLicensingService configures the licensing parameters

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

**Unit**

`IBCAdmin`

**Syntax**

```delphi
TIBCLicensingService = class(TCustomIBCService);
```

**Remarks**

Use the TIBCLicensingService component to add or remove InterBase software activation certificates.

**Inheritance Hierarchy**

- `TCustomIBCService`
- **TIBCLicensingService**

© 1997-2021 Devart. All Rights Reserved.

5.13.1.11.1 **Members**

**TIBCLicensingService** class overview.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action</strong></td>
<td>Used to indicate what action the licensing service should take.</td>
</tr>
<tr>
<td><strong>Active</strong> (inherited from TCustomIBCService)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from TCustomIBCService)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>Used to set or return the license identification.</td>
</tr>
<tr>
<td><strong>Key</strong></td>
<td>Used to set or return the license key.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from TCustomIBCService)</td>
<td>Used to dispaly a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from TCustomIBCService)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> (inherited from TCustomIBCService)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>Server</strong> (inherited from TCustomIBCService)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td><strong>ServiceParamBySPB</strong> (inherited from TCustomIBCService)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddLicense</strong></td>
<td>Adds an InterBase software activation certificate.</td>
</tr>
<tr>
<td><strong>Attach</strong> (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td><strong>Detach</strong> (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td><strong>RemoveLicense</strong></td>
<td>Removes an InterBase software activation certificate.</td>
</tr>
<tr>
<td><strong>ServiceStart</strong>  (inherited from TCustomIBCService)</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>
Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttach</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>

Properties of the TIBCLicensingService class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td>ServiceParamBySPB</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Used to indicate what action the licensing service should take.</td>
</tr>
<tr>
<td>Active</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>ID</td>
<td>Used to set or return the license identification.</td>
</tr>
<tr>
<td>Key</td>
<td>Used to set or return the license key.</td>
</tr>
<tr>
<td>LoginPrompt</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> <em>(inherited from TCustomIBCService)</em></td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>Server</strong> <em>(inherited from TCustomIBCService)</em></td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
</tbody>
</table>

**See Also**
- TIBCLicensingService Class
- TIBCLicensingService Class Members

5.13.11.2.1 Action Property

Used to indicate what action the licensing service should take.

**Class**

TIBCLicensingService

**Syntax**

```property Action: TIBCLicensingAction default laAdd;```

**Remarks**

Use the Action property to indicate what action the licensing service should take. This causes the service to call the AddLicense or RemoveLicense method, depending on whether the software activation certificate is added or removed.

**Note:** You should install InterBase 6 to use this feature.

**See Also**
- AddLicense
- RemoveLicense
5.13.1.11.2.2 ID Property

Used to set or return the license identification.

Class
TIBCLicensingService

Syntax

property ID: string;

Remarks
Use the ID property to set or return the license identification.

Note: You should install InterBase 6 to use this feature.

5.13.1.11.2.3 Key Property

Used to set or return the license key.

Class
TIBCLicensingService

Syntax

property Key: string;

Remarks
Use the Key property to set or return the license key.

Note: You should install InterBase 6 to use this feature.

5.13.1.11.3 Methods

Methods of the TIBCLicensingService class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddLicense</td>
<td>Adds an InterBase software activation certificate.</td>
</tr>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>RemoveLicense</td>
<td>Removes an InterBase software activation certificate.</td>
</tr>
<tr>
<td>ServiceStart (inherited from TCustomIBCService)</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

See Also
- TIBCLicensingService Class
- TIBCLicensingService Class Members

5.13.1.11.3.1 AddLicense Method

Adds an InterBase software activation certificate.

Class
TIBCLicensingService

Syntax

```plaintext
procedure AddLicense;
```

Remarks

Call the AddLicense method along to add an InterBase software activation certificate. The Key and ID properties should be supplied.

Note: You should install InterBase 6 to use this feature.

See Also
5.13.1.11.3.2 RemoveLicense Method

Removes an InterBase software activation certificate.

Class

TIBCLicensingService

Syntax

procedure RemoveLicense;

Remarks

Call the RemoveLicense method to remove an InterBase software activation certificate.

Note: You should install InterBase 6 to use this feature.

5.13.1.12 TIBCLimboTransactionInfo Class

TIBCLimboTransactionInfo stores information about a limbo transaction.

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

Unit

IBCAadmin

Syntax

TIBCLimboTransactionInfo = class(System TObject);

Remarks

TIBCLimboTransactionInfo describes limbo transaction. Limbo transactions are usually caused by the failure of a two-phased commit. They can also exist due to system failure or when a single-database transaction is prepared.
5.13.1.12.1 Members

**TIBCLimboTransactionInfo** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Used to retrieve the action to take for ending the transaction.</td>
</tr>
<tr>
<td>Advise</td>
<td>Used to retrieve the suggested action for ending the transaction.</td>
</tr>
<tr>
<td>HostSite</td>
<td>Holds the host name for the server where the transaction was started.</td>
</tr>
<tr>
<td>ID</td>
<td>Holds the identifier for the limbo transaction.</td>
</tr>
<tr>
<td>MultiDatabase</td>
<td>Used to indicate whether the limbo transaction involves multiple databases.</td>
</tr>
<tr>
<td>RemoteDatabasePath</td>
<td>Holds the path to the remote server.</td>
</tr>
<tr>
<td>RemoteSite</td>
<td>Holds the host name for a remote server involved in the transaction.</td>
</tr>
<tr>
<td>State</td>
<td>Used to indicate the current state of the limbo transaction.</td>
</tr>
</tbody>
</table>

5.13.1.12.2 Properties

Properties of the **TIBCLimboTransactionInfo** class.

For a complete list of the **TIBCLimboTransactionInfo** class members, see the **TIBCLimboTransactionInfo Members** topic.
### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action</strong></td>
<td>Used to retrieve the action to take for ending the transaction.</td>
</tr>
<tr>
<td><strong>Advise</strong></td>
<td>Used to retrieve the suggested action for ending the transaction.</td>
</tr>
<tr>
<td><strong>HostSite</strong></td>
<td>Holds the host name for the server where the transaction was started.</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>Holds the identifier for the limbo transaction.</td>
</tr>
<tr>
<td><strong>MultiDatabase</strong></td>
<td>Used to indicate whether the limbo transaction involves multiple databases.</td>
</tr>
<tr>
<td><strong>RemoteDatabasePath</strong></td>
<td>Holds the path to the remote server.</td>
</tr>
<tr>
<td><strong>RemoteSite</strong></td>
<td>Holds the host name for a remote server involved in the transaction.</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>Used to indicate the current state of the limbo transaction.</td>
</tr>
</tbody>
</table>

**See Also**
- [TIBCLimboTransactionInfo Class](#)
- [TIBCLimboTransactionInfo Class Members](#)

® 1997-2021 Devart. All Rights Reserved.
**property** Action: `TIBCTransactionAction`;

**Remarks**

Use the Action property to show the action to take for ending the transaction.

---

5.13.1.12.2.2 Advise Property

Used to retrieve the suggested action for ending the transaction.

**Class**

`TIBCLimboTransactionInfo`

**Syntax**

```
property Advise: TIBCTransactionAdvise;
```

**Remarks**

Use the Advise property to show the suggested action for ending the transaction.

---

5.13.1.12.2.3 HostSite Property

Holds the host name for the server where the transaction was started.

**Class**

`TIBCLimboTransactionInfo`

**Syntax**

```
property HostSite: string;
```

**Remarks**

The HostSite property holds the host name for the server where the transaction was started.
5.13.1.12.2.4  ID Property

Holds the identifier for the limbo transaction.

Class

TIBCLimboTransactionInfo

Syntax

[property] ID: integer;

Remarks

The ID property holds the identifier for the limbo transaction.

5.13.1.12.2.5  MultiDatabase Property

Used to indicate whether the limbo transaction involves multiple databases.

Class

TIBCLimboTransactionInfo

Syntax

[property] MultiDatabase: boolean;

Remarks

Use the MultiDatabase property to indicate whether the limbo transaction involves multiple databases.

5.13.1.12.2.6  RemoteDatabasePath Property

Holds the path to the remote server.

Class

TIBCLimboTransactionInfo
5.13.1.12.2.7 RemoteSite Property

Holds the host name for a remote server involved in the transaction.

**Class**

*TIBCLimboTransactionInfo*

**Syntax**

```pascal
property RemoteSite: string;
```

**Remarks**

The RemoteSite property holds the host name for a remote server involved in the transaction.

5.13.1.12.2.8 State Property

Used to indicate the current state of the limbo transaction.

**Class**

*TIBCLimboTransactionInfo*

**Syntax**

```pascal
property State: TIBCTransactionState;
```

**Remarks**

Use the State property to indicate the current state of the limbo transaction.
5.13.1.13 TIBCLogService Class

Returns the contents of the interbase.log file from server.

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

Unit
IBCAdmin

Syntax

TIBCLogService = class(TIBCControlAndQueryService);

Remarks

Use a TIBCLogService object to return the contents of the interbase.log file from server.

Inheritance Hierarchy

TCustomIBCService
  TIBCControlAndQueryService
  TIBCLogService

© 1997-2021
Devart. All Rights Reserved.

5.13.1.13.1 Members

TIBCLogService class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong></td>
<td>(inherited from TCustomIBCService) Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BufferSize</strong></td>
<td>(inherited from TIBCControlAndQueryService) Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>Eof</strong></td>
<td>(inherited from TIBCControlAndQueryService) Used to determine whether the end of the file has been reached.</td>
</tr>
</tbody>
</table>
Handle (inherited from TCustomIBCService) | Used to return the database handle.
---|---
LoginPrompt (inherited from TCustomIBCService) | Used to display a login prompt before attaching to a database.
Params (inherited from TCustomIBCService) | Used to set or return database parameters.
Protocol (inherited from TCustomIBCService) | Used to select the network protocol.
Server (inherited from TCustomIBCService) | Used to set the name of the server on which the services are to be run.
ServiceParamBySPB (inherited from TCustomIBCService) | Used to return and sets SPB parameters.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>GetNextChunk (inherited from TIBCControlAndQueryService)</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td>GetNextLine (inherited from TIBCControlAndQueryService)</td>
<td>Returns the next line of data.</td>
</tr>
<tr>
<td>ServiceStart (inherited from TCustomIBCService)</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttach (inherited from TCustomIBCService)</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>
5.13.1.14 TIBCRestoreService Class

Used to restore a database.

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

Unit

IBCAdmin

Syntax

TIBCRestoreService = class(TIBCBackupRestoreService);

Remarks

Use a TIBCRestoreService object to restore a database.

Inheritance Hierarchy

TCustomIBCService
    TIBCControlAndQueryService
        TIBCBackupRestoreService
            TIBCRestoreService

© 1997-2021
Devart. All Rights Reserved.

5.13.1.14.1 Members

TIBCRestoreService class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>BackupFile</td>
<td>Used to set or return the backup file name.</td>
</tr>
<tr>
<td>BufferSize</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td>Database</td>
<td>Used to set or return the database name.</td>
</tr>
<tr>
<td>Eof</td>
<td>Used to determine whether</td>
</tr>
</tbody>
</table>
### InterBase Data Access Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Inherited From</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle (inherited from TCustomIBCService)</td>
<td>Used to return the database handle.</td>
<td></td>
</tr>
<tr>
<td>LoginPrompt (inherited from TCustomIBCService)</td>
<td>Used to display a login prompt before attaching to a database.</td>
<td></td>
</tr>
<tr>
<td>NBackupLevel (inherited from TIBCBackupRestoreService)</td>
<td>Used to set backup level for nBackup.</td>
<td></td>
</tr>
<tr>
<td>NBackupOptions (inherited from TIBCBackupRestoreService)</td>
<td>Used to set backup options for nBackup.</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>Used to build restore options.</td>
<td>TCustomIBCService</td>
</tr>
<tr>
<td>PageBuffers</td>
<td>Used to set or return the page buffer size.</td>
<td></td>
</tr>
<tr>
<td>PageSize</td>
<td>Used to set or return the page size.</td>
<td></td>
</tr>
<tr>
<td>Params (inherited from TCustomIBCService)</td>
<td>Used to set or return database parameters.</td>
<td></td>
</tr>
<tr>
<td>Protocol (inherited from TCustomIBCService)</td>
<td>Used to select the network protocol.</td>
<td></td>
</tr>
<tr>
<td>Server (inherited from TCustomIBCService)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
<td></td>
</tr>
<tr>
<td>ServiceParamBySPB (inherited from TCustomIBCService)</td>
<td>Used to return and sets SPB parameters.</td>
<td></td>
</tr>
<tr>
<td>UseNBackup (inherited from TIBCBackupRestoreService)</td>
<td>Used to enable or disable using nBackup service.</td>
<td></td>
</tr>
<tr>
<td>Verbose</td>
<td>Used to set or return the restore in the verbose mode.</td>
<td></td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Inherited From</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
<td></td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
<td></td>
</tr>
<tr>
<td>GetNextChunk (inherited from TIBCControlAndQueryService)</td>
<td>Returns the next chunk of data.</td>
<td></td>
</tr>
</tbody>
</table>
### GetNextLine (inherited from TIBCControlAndQueryService)
Returns the next line of data.

### ServiceStart (inherited from TCustomIBCService)
Starts the service.

#### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttach (inherited from TCustomIBCService)</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.13.1.14.2 Properties

Properties of the TIBCRestoreService class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eof (inherited from TIBCControlAndQueryService)</td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td>Handle (inherited from TCustomIBCService)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td>NBackupLevel (inherited from TIBCBackupRestoreService)</td>
<td>Used to set backup level for nBackup.</td>
</tr>
<tr>
<td>NBackupOptions (inherited from TIBCBackupRestoreService)</td>
<td>Used to set backup options for nBackup.</td>
</tr>
<tr>
<td>ServiceParamBySPB (inherited from TCustomIBCService)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
<tr>
<td>UseNBackup (inherited from TIBCBackupRestoreService)</td>
<td>Used to enable or disable using nBackup service.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Active</td>
<td>(inherited from TCustomIBCService) Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>BackupFile</td>
<td>Used to set or return the backup file name.</td>
</tr>
<tr>
<td>BufferSize</td>
<td>(inherited from TIBCControlAndQueryService) Used to set or return the buffer size.</td>
</tr>
<tr>
<td>Database</td>
<td>Used to set or return the database name.</td>
</tr>
<tr>
<td>LoginPrompt</td>
<td>(inherited from TCustomIBCService) Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to build restore options.</td>
</tr>
<tr>
<td>PageBuffers</td>
<td>Used to set or return the page buffer size.</td>
</tr>
<tr>
<td>PageSize</td>
<td>Used to set or return the page size.</td>
</tr>
<tr>
<td>Params</td>
<td>(inherited from TCustomIBCService) Used to set or return database parameters.</td>
</tr>
<tr>
<td>Protocol</td>
<td>(inherited from TCustomIBCService) Used to select the network protocol.</td>
</tr>
<tr>
<td>Server</td>
<td>(inherited from TCustomIBCService) Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td>Verbose</td>
<td>Used to set or return the restore in the verbose mode.</td>
</tr>
</tbody>
</table>

See Also
- TIBCRestoreService Class
- TIBCRestoreService Class Members

5.13.1.14.2.1 BackupFile Property

Used to set or return the backup file name.

Class

TIBCRestoreService
Syntax

**property** BackupFile: TStrings;

Remarks

Use the BackupFile property to set or return the backup file name.

**Note:** You should install InterBase 6 to use this feature.

Class

**TIBCRestoreService**

Syntax

**property** Database: TStrings;

Remarks

Use the Database property to set or return the database name to set properties on.

5.13.1.14.2.3 Options Property

Used to build restore options.

Class

**TIBCRestoreService**

Syntax

**property** Options: TIBCRestoreOptions default [roCreateNewDB];

Remarks
Use the Options property to build restore options of type TIBCRestoreOption into application.

5.13.1.14.2.4 PageBuffers Property

Used to set or return the page buffer size.

Class

TIBCRestoreService

Syntax

| property PageBuffers: Integer default 0; |

Remarks

Use the PageBuffers property to set or return the page buffer size in kilobytes.

Note: You should install InterBase 6 to use this feature.

5.13.1.14.2.5 PageSize Property

Used to set or return the page size.

Class

TIBCRestoreService

Syntax

| property PageSize: Integer default 4096; |

Remarks

Use the PageSize property to set or return the page size in kilobytes.

Note: You should install InterBase 6 to use this feature.
5.13.1.14.2.6 Verbose Property

Used to set or return the restore in the verbose mode.

Class

TIBCRestoreService

Syntax

```
property Verbose: Boolean;
```

Remarks

Use the Verbose property to set or return the restore in the verbose mode.

© 1997-2021
Devart. All Rights Reserved.

5.13.1.15 TIBCSecurityService Class

Used to manage user access to the InterBase server.

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

Unit

IBCAadmin

Syntax

```
TIBCSecurityService = class(TIBCControlAndQueryService);
```

Remarks

Use a TIBCSecurityService object to manage user access to the InterBase server.

Inheritance Hierarchy

```
TCustomIBCService
  TIBCControlAndQueryService
  TIBCSecurityService
```

© 1997-2021
Devart. All Rights Reserved.
### TIBCSecurityService class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong></td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BufferSize</strong></td>
<td>(inherited from <strong>TIBCControlAndQueryService</strong>) Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>Eof</strong></td>
<td>(inherited from <strong>TIBCControlAndQueryService</strong>) Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td><strong>Handle</strong></td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to return the database handle.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong></td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong></td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>SecurityAction</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td><strong>ServiceParamBySPB</strong></td>
<td>(inherited from <strong>TCustomIBCService</strong>) Used to return and sets SPB parameters.</td>
</tr>
<tr>
<td><strong>UserDatabase</strong></td>
<td></td>
</tr>
<tr>
<td><strong>UserInfo</strong></td>
<td></td>
</tr>
<tr>
<td><strong>UserInfos</strong></td>
<td>Used to get the user's information.</td>
</tr>
<tr>
<td><strong>UserInfosCount</strong></td>
<td>Used to get the number of returned records.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddUser</td>
<td>Adds a user to the user table.</td>
</tr>
<tr>
<td>Attach</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>DeleteUser</td>
<td>Used to delete a user from the user table.</td>
</tr>
<tr>
<td>Detach</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>DisplayUser</td>
<td>Fetches user information and dumps it into the TIBCUserInfo record</td>
</tr>
<tr>
<td>DisplayUsers</td>
<td>Fetches all valid user information.</td>
</tr>
<tr>
<td>EnableEUA</td>
<td>Enables user authentication.</td>
</tr>
<tr>
<td>GetNextChunk</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td>GetNextLine</td>
<td>Returns the next line of data.</td>
</tr>
<tr>
<td>ModifyUser</td>
<td>Modifies user's information.</td>
</tr>
<tr>
<td>ServiceStart</td>
<td>Starts the service.</td>
</tr>
<tr>
<td>SuspendEUA</td>
<td>Suspends embedded user authentication.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttach</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Eof</strong> (inherited from <strong>TIBCControlAndQueryService</strong>)</td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td><strong>ServiceParamBySPB</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
<tr>
<td><strong>UserInfos</strong></td>
<td>Used to get the user's information.</td>
</tr>
<tr>
<td><strong>UserInfosCount</strong></td>
<td>Used to get the number of returned records.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BufferSize</strong> (inherited from <strong>TIBCControlAndQueryService</strong>)</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to dispaly a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>SecurityAction</strong></td>
<td>Used to determine the type of the operation that will be performed by InterBase Security Service.</td>
</tr>
<tr>
<td><strong>Server</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td><strong>UserDatabase</strong></td>
<td>Used to add, modify, or delete a user in the</td>
</tr>
</tbody>
</table>
5.13.1.15.2.1  SecurityAction Property

Used to determine the type of the operation that will be performed by InterBase Security Service.

Class

TIBCSecurityService

Syntax

```pascal
property SecurityAction: TIBCSecurityAction default saAddUser;
```

Remarks

The SecurityAction property determines the type of the operation that will be performed by InterBase Security Service.

**Note:** You should install InterBase 6 to use this feature.

5.13.1.15.2.2  UserDatabase Property

Used to add, modify, or delete a user in the specified database.

Class

TIBCSecurityService

Syntax
**property** UserDatabase: `string`;

Remarks

Set the UserDatabase property to add, modify, or delete a user in the specified database.

**Note:** You should install InterBase 6 to use this feature.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.15.2.3 UserInfo Property

Used to return the user's information.

**Class**

`TIBCSecurityService`

**Syntax**

```property UserInfo: TIBCUserInfo;```

Remarks

Fill in the UserInfo property to add, delete, or modify a user.

**Note:** You should install InterBase 6 to use this feature.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.15.2.4 UserInfos Property(Indexer)

Used to get the user's information.

**Class**

`TIBCSecurityService`

**Syntax**

```property UserInfos[Index: Integer]: TIBCUserInfo;```

**Parameters**

`Index`
Holds the index of the user to get information on.

Remarks

Use the UserInfos property to return the user's information as a TIBCUserInfo record, which contains the user name, groupID, UserID, and the user's first, middle, and last names. Specify the index of the required user in the Index parameter.

Class

TIBCSecurityService

Syntax

property UserInfosCount: Integer;

Remarks

Use the UserInfosCount to get the number of returned TIBCUserInfo records.

Methods

Methods of the TIBCSecurityService class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddUser</td>
<td>Adds a user to the user table.</td>
</tr>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>DeleteUser</td>
<td>Used to delete a user from the user table.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Detach</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>DisplayUser</td>
<td>Fetches user information and dumps it into the TIBCUserInfo record</td>
</tr>
<tr>
<td>DisplayUsers</td>
<td>Fetches all valid user information.</td>
</tr>
<tr>
<td>EnableEUA</td>
<td>Enables user authentication.</td>
</tr>
<tr>
<td>GetNextChunk</td>
<td>(inherited from TIBCCcontrolAndQueryService)</td>
</tr>
<tr>
<td>GetNextLine</td>
<td>(inherited from TIBCCcontrolAndQueryService)</td>
</tr>
<tr>
<td>ModifyUser</td>
<td>Modifies user's information.</td>
</tr>
<tr>
<td>ServiceStart</td>
<td>(inherited from TCustomIBCService)</td>
</tr>
<tr>
<td>SuspendEUA</td>
<td>Suspends embedded user authentication.</td>
</tr>
</tbody>
</table>

**See Also**
- TIBCSecurityService Class
- TIBCSecurityService Class Members

© 1997-2021 Devart. All Rights Reserved.
5.13.1.15.3.2 DeleteUser Method

Used to delete a user from the user table.

Class

TIBCSecurityService

Syntax

procedure DeleteUser;

Remarks

Call the DeleteUser method to delete a user from the user table.

Note: You should install InterBase 6 to use this feature.

5.13.1.15.3.3 DisplayUser Method

Fetches user information and dumps it into the TIBCUserInfo record.

Class

TIBCSecurityService

Syntax

procedure DisplayUser(const UserName: string);

Parameters

UserName

Holds the user name.

Remarks

Call the DisplayUser method to fetch user information and dump it into the TIBCUserInfo record, which can then be returned by the UserInfos property.
5.13.1.15.3.4 DisplayUsers Method

Fetches all valid user information.

Class

TIBCSecurityService

Syntax

```pascal
procedure DisplayUsers;
```

Remarks

Call the DisplayUsers method to fetch all valid user information, which can then be returned by the UserInfo property.

Note: You should install InterBase 6 to use this feature.

5.13.1.15.3.5 EnableEUA Method

Enables user authentication.

Class

TIBCSecurityService

Syntax

```pascal
procedure EnableEUA(Value: Boolean);
```

Parameters

- `Value`  
  True if EUA is enabled.

Remarks

Call the EnableEUA method to enable embedded user authentication.
Note: You should install InterBase 6 to use this feature.

5.13.1.15.3.6  ModifyUser Method

Modifies user's information.

Class

TIBCSecurityService

Syntax

procedure ModifyUser;

Remarks

Call ModifyUser to modify user's information.

Note: You should install InterBase 6 to use this feature.

5.13.1.15.3.7  SuspendEUA Method

Suspends embedded user authentication.

Class

TIBCSecurityService

Syntax

procedure SuspendEUA(Value: Boolean);

Parameters

Value

True, if embedded user authentication is suspended.

Remarks

Call the SuspendEUA method to suspend embedded user authentication.
5.13.1.16 TIBCServerProperties Class

A class that returns database server information.

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

Unit

IBCAadmin

Syntax

TIBCServerProperties = class(TCustomIBCService);

Remarks

The TIBCServerProperties class returns database server information, including configuration parameters, and also version and license information.

Note: You must install InterBase 6 to use this feature.

Inheritance Hierarchy

TCustomIBCService

TIBCServerProperties

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>(inherited from TCustomIBCService) Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>AliasCount</td>
<td>Used to get the alias count.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>ConfigParams</td>
<td>Used to return server configuration parameters.</td>
</tr>
<tr>
<td>DatabaseInfo</td>
<td>Used to get database information.</td>
</tr>
<tr>
<td>Handle (inherited from TCustomIBCService)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td>LicenseInfo</td>
<td>Used to get licensing information.</td>
</tr>
<tr>
<td>LicenseMaskInfo</td>
<td>Used to return licensing information as a TIBCLicenseMaskInfo record, which includes the license mask and capability mask.</td>
</tr>
<tr>
<td>LoginPrompt (inherited from TCustomIBCService)</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td>Params (inherited from TCustomIBCService)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td>Protocol (inherited from TCustomIBCService)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td>Server (inherited from TCustomIBCService)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td>ServiceParamBySPB (inherited from TCustomIBCService)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
<tr>
<td>VersionInfo</td>
<td>Used to get the server configuration parameters.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>AddAlias</td>
<td>Adds database alias.</td>
</tr>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>DeleteAlias</td>
<td>Deletes database alias.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>Fetch</td>
<td>Gets information from the server.</td>
</tr>
<tr>
<td>FetchAliasInfo</td>
<td>Returns database alias information.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>FetchConfigParams</td>
<td>Returns database configuration parameters.</td>
</tr>
<tr>
<td>FetchDatabaseInfo</td>
<td>Returns database information.</td>
</tr>
<tr>
<td>FetchLicenseInfo</td>
<td>Returns database license information.</td>
</tr>
<tr>
<td>FetchLicenseMaskInfo</td>
<td>Returns license mask information.</td>
</tr>
<tr>
<td>FetchVersionInfo</td>
<td>Returns the database version information.</td>
</tr>
<tr>
<td>ServiceStart (inherited from TCustomIBCService)</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttach (inherited from TCustomIBCService)</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>

Properties of the TIBCServerProperties class.

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

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AliasCount</td>
<td>Used to get the alias count.</td>
</tr>
<tr>
<td>ConfigParams</td>
<td>Used to return server configuration parameters.</td>
</tr>
<tr>
<td>DatabaseInfo</td>
<td>Used to get database information.</td>
</tr>
<tr>
<td>Handle (inherited from TCustomIBCService)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td>LicenseInfo</td>
<td>Used to get licensing information.</td>
</tr>
<tr>
<td>LicenseMaskInfo</td>
<td>Used to return licensing information as a</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>ServiceParamBySPB (inherited from TCustomIBCService)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
<tr>
<td>VersionInfo</td>
<td>Used to get the server configuration parameters.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active (inherited from TCustomIBCService)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>LoginPrompt (inherited from TCustomIBCService)</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td>Params (inherited from TCustomIBCService)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td>Protocol (inherited from TCustomIBCService)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td>Server (inherited from TCustomIBCService)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
</tbody>
</table>

See Also

- TIBCServerProperties Class
- TIBCServerProperties Class Members

© 1997-2021 Devart. All Rights Reserved.

5.13.1.16.2.1 AliasCount Property

Used to get the alias count.

Class

TIBCServerProperties

Syntax
**property** AliasCount: Integer;

Remarks

Use the AliasCount property to obtain the alias count.

**Note:** You should install InterBase 6 to use this feature.

---

5.13.1.16.2.2 ConfigParams Property

Used to return server configuration parameters.

Class

**TIBCServerProperties**

Syntax

```plaintext
property ConfigParams: TIBCConfigParams;
```

Remarks

Use the ConfigParams property to return the server configuration parameters as a TIBCConfigParams record, which includes the configuration file data, base location, lock file location, message file location, and the security database location.

**Note:** You should install InterBase 6 to use this feature.

---

5.13.1.16.2.3 DatabaseInfo Property

Used to get database information.

Class

**TIBCServerProperties**

Syntax

```plaintext
property DatabaseInfo: TIBCDatabaseInfo;
```
Remarks

Use the DatabaseInfo property to obtain database information, which includes the database name, number of attachments, and number of databases.

Note: You should install InterBase 6 to use this feature.

Class

TIBCServerProperties

Syntax

property LicenseInfo: TIBCLicenseInfo;

Remarks

Use the LicenseInfo property to return licensing information as a TIBCLicenseInfo record, which includes the key, ID, and number of licensed users.

Note: You should install InterBase 6 to use this feature.

Class

TIBCServerProperties

Syntax

property LicenseMaskInfo: TIBCLicenseMaskInfo;

Remarks

Used to return licensing information as a TIBCLicenseMaskInfo record, which includes the license mask and capability mask.

Class

TIBCServerProperties
Remarks

Use the LicenseMaskInfo property to return licensing information as a TIBCLicenseMaskInfo record, which includes the license mask and capability mask. A license mask is a bitmask representing the software activation certificate options currently enabled on the server. A capability mask is a bitmask representing the capabilities currently enabled on the server.

Note: You should install InterBase 6 to use this feature.

5.13.1.16.2.6 VersionInfo Property

Used to get the server configuration parameters.

Class

TIBCServerProperties

Syntax

property VersionInfo: TIBCVersionInfo;

Remarks

Use the VersionInfo property to obtain the server configuration parameters as a TIBCVersionInfo type, which includes the service version, and server version and implementation.

Note: You should install InterBase 6 to use this feature.

5.13.1.16.3 Methods

Methods of the TIBCServerProperties class.

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

Public
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddAlias</td>
<td>Adds database alias.</td>
</tr>
<tr>
<td>Attach (inherited from TCustomIBCSERVICE)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>DeleteAlias</td>
<td>Deletes database alias.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCSERVICE)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>Fetch</td>
<td>Gets information from the server.</td>
</tr>
<tr>
<td>FetchAliasInfo</td>
<td>Returns database alias information.</td>
</tr>
<tr>
<td>FetchConfigParams</td>
<td>Returns database configuration parameters.</td>
</tr>
<tr>
<td>FetchDatabaseInfo</td>
<td>Returns database information.</td>
</tr>
<tr>
<td>FetchLicenseInfo</td>
<td>Returns database license information.</td>
</tr>
<tr>
<td>FetchLicenseMaskInfo</td>
<td>Returns license mask information.</td>
</tr>
<tr>
<td>FetchVersionInfo</td>
<td>Returns the database version information.</td>
</tr>
<tr>
<td>ServiceStart (inherited from TCustomIBCSERVICE)</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

See Also
- TIBCServerProperties Class
- TIBCServerProperties Class Members

© 1997-2021 Devart. All Rights Reserved.

5.13.1.16.3.1 AddAlias Method

Adds database alias.

Class

TIBCServerProperties

Syntax

```
procedure AddAlias(const Alias: string; const DBPath: string);
```

Parameters
### Alias

Holds the database alias.

### DBPath

**Remarks**

Call the AddAlias method to add database alias.

**Note:** You should install InterBase 6 to use this feature.

### DeleteAlias Method

Deletes database alias.

**Class**

**TIBCServerProperties**

**Syntax**

```
procedure DeleteAlias(const Alias: string);
```

**Parameters**

- **Alias**
  
  Holds database alias.

**Remarks**

Call the DeleteAlias method to delete database alias.

**Note:** You should install InterBase 6 to use this feature.

### Fetch Method

Gets information from the server.

**Class**

**TIBCServerProperties**
5.13.1.16.3.4 FetchAliasInfo Method

Returns database alias information.

Class

TIBCServerProperties

Syntax

```
procedure FetchAliasInfo;
```

Remarks

Call the FetchAliasInfo method to return database alias information.

Note: You should install InterBase 6 to use this feature.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.16.3.5 FetchConfigParams Method

Returns database configuration parameters.

Class

TIBCServerProperties

Syntax

```
procedure FetchConfigParams;
```
Remarks
Call the FetchConfigParams method to return database configuration parameters.

Note: You should install InterBase 6 to use this feature.

5.13.1.16.3.6 FetchDatabaseInfo Method

Returns database information.

Class
TIBCServerProperties

Syntax

procedure FetchDatabaseInfo;

Remarks
Call the FetchDatabaseInfo method to return database information.

Note: You should install InterBase 6 to use this feature.

5.13.1.16.3.7 FetchLicenseInfo Method

Returns database license information.

Class
TIBCServerProperties

Syntax

procedure FetchLicenseInfo;

Remarks
Call the FetchLicenseInfo method to return database license information.
5.13.1.16.3.8 FetchLicenseMaskInfo Method

Returns license mask information.

Class

TIBCServerProperties

Syntax

procedure FetchLicenseMaskInfo;

Remarks

Call the FetchLicenseMaskInfo method to return database license mask information.

Note: You should install InterBase 6 to use this feature.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.16.3.9 FetchVersionInfo Method

Returns the database version information.

Class

TIBCServerProperties

Syntax

procedure FetchVersionInfo;

Remarks

Call the FetchVersionInfo method to return the database version information.

Note: You should install InterBase 6 to use this feature.

© 1997-2021 Devart. All Rights Reserved.
5.13.1.17 TIBCStatisticalService Class

TIBCStatisticalService is used to view database statistics.

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

Unit
IBCAdmin

Syntax

TIBCStatisticalService = class(TIBControlAndQueryService);

Remarks

Use a TIBCStatisticalService object to view database statistics.

TIBCStatisticalService contains many properties and methods to allow you to build a statistical component into your application. Only the SYSDBA user or owner of the database will be able to run this service.

Inheritance Hierarchy

TCustomIBCService
  TIBControlAndQueryService
  TIBCStatisticalService

© 1997-2021
Devart. All Rights Reserved.

5.13.1.17.1 Members

TIBCStatisticalService class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>(inherited from TCustomIBCService)</td>
</tr>
<tr>
<td>BufferSize</td>
<td>(inherited from TIBControlAndQueryService)</td>
</tr>
<tr>
<td>Database</td>
<td>Used to set or return the database name.</td>
</tr>
</tbody>
</table>
Eof (inherited from TIBCControlAndQueryService) | Used to determine whether the end of the file has been reached.
Handle (inherited from TCustomIBCService) | Used to return the database handle.
LoginPrompt (inherited from TCustomIBCService) | Used to display a login prompt before attaching to a database.
Options | Used to specify the behaviour of a TIBCStatisticalService object.
Params (inherited from TCustomIBCService) | Used to set or return database parameters.
Protocol (inherited from TCustomIBCService) | Used to select the network protocol.
Server (inherited from TCustomIBCService) | Used to set the name of the server on which the services are to be run.
ServiceParamBySPB (inherited from TCustomIBCService) | Used to return and sets SPB parameters.
TableNames | Used to specify the name of the table to request statistics for.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>GetNextChunk (inherited from TIBCControlAndQueryService)</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td>GetNextLine (inherited from TIBCControlAndQueryService)</td>
<td>Returns the next line of data.</td>
</tr>
<tr>
<td>ServiceStart (inherited from TCustomIBCService)</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>

Events
### Properties

Properties of the `TIBCStatisticalService` class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Eof</code> (inherited from <code>TIBCControlAndQueryService</code>)</td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td><code>Handle</code> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td><code>ServiceParamBySPB</code> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Active</code> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><code>BufferSize</code> (inherited from <code>TIBCControlAndQueryService</code>)</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td><code>Database</code></td>
<td>Used to set or return the database name.</td>
</tr>
<tr>
<td><code>LoginPrompt</code> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to dispaly a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><code>Options</code></td>
<td>Used to specify the behaviour of a <code>TIBCStatisticalService</code> object.</td>
</tr>
</tbody>
</table>
**Params** (inherited from **TCustomIBCService**)  
Used to set or return database parameters.

**Protocol** (inherited from **TCustomIBCService**)  
Used to select the network protocol.

**Server** (inherited from **TCustomIBCService**)  
Used to set the name of the server on which the services are to be run.

**TableNames**  
Used to specify the name of the table to request statistics for.

### See Also
- **TIBCStatisticalService Class**
- **TIBCStatisticalService Class Members**

#### 5.13.1.17.2.1 Database Property

Used to set or return the database name.

**Class**  
**TIBCStatisticalService**

**Syntax**

```
property Database: string;
```

**Remarks**

Use the Database property to set or return the database name to set properties on.

© 1997-2021  
Devart. All Rights Reserved.
Syntax

```delphi
property Options: TIBCStatOptions;
```

Remarks

Use the Options property of TIBCStatisticalService to request database statistics.

These options are incremental; that is, setting DbLog to True also returns HeaderPages statistics, setting IndexPages to True returns also returns DbLog and HeaderPages statistics, and so forth.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.17.2.3 TableNames Property

Used to specify the name of the table to request statistics for.

Class

TIBCStatisticalService

Syntax

```delphi
property TableNames: string;
```

Remarks

Use the TableName property to specify the name of the table for which statistics should be requested.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.18 TIBCTraceService Class

This component is used for working with trace service added in Firebird 2.5.

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

Unit

IBCAdmin
Syntax

TIBCTraceService = class(TIBCControlAndQueryService);

Remarks

The TIBCTraceService component is used to work with trace service added in Firebird 2.5.

Use the TIBCTraceService component to work with trace service. Trace enables various events performed inside the engine, such as statement execution, connections, disconnections, etc., to be logged and collated for real-time analysis of the corresponding performance characteristics.

Trace takes place in the context of a trace session. Each trace session has its own configuration, state and output. When a user application starts a trace session, it sets SessionName (optional) and the session configuration (Config) (mandatory).

Inheritance Hierarchy

TCustomIBCService
  TIBCControlAndQueryService
    TIBCTraceService

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

5.13.1.18.1 Members

TIBCTraceService class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>(inherited from TCustomIBCService) Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td>BufferSize</td>
<td>(inherited from TIBCControlAndQueryService) Used to set or return the buffer size.</td>
</tr>
<tr>
<td>Config</td>
<td>Used to set the configuration of a trace session.</td>
</tr>
<tr>
<td>Eof</td>
<td>(inherited from TIBCControlAndQueryService) Used to determine whether the end of the file has been reached.</td>
</tr>
</tbody>
</table>
Handle (inherited from TCustomIBCService)  |  Used to return the database handle.
LoginPrompt (inherited from TCustomIBCService)  |  Used to display a login prompt before attaching to a database.
Params (inherited from TCustomIBCService)  |  Used to set or return database parameters.
Protocol (inherited from TCustomIBCService)  |  Used to select the network protocol.
Server (inherited from TCustomIBCService)  |  Used to set the name of the server on which the services are to be run.
ServiceParamBySPB (inherited from TCustomIBCService)  |  Used to return and sets SPB parameters.
SessionName  |  Used to set session name for a trace session.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>GetNextChunk (inherited from TIBControlAndQueryService)</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td>GetNextLine (inherited from TIBControlAndQueryService)</td>
<td>Returns the next line of data.</td>
</tr>
<tr>
<td>ListTraceSessions</td>
<td>Returns information about all trace sessions.</td>
</tr>
<tr>
<td>ResumeTrace</td>
<td>Resumes a trace session.</td>
</tr>
<tr>
<td>ServiceStart (inherited from TCustomIBCService)</td>
<td>Starts the service.</td>
</tr>
<tr>
<td>StartTrace</td>
<td>Starts the trace session.</td>
</tr>
<tr>
<td>StopTrace</td>
<td>Stops a trace session.</td>
</tr>
<tr>
<td>SuspendTrace</td>
<td>Suspends a trace session.</td>
</tr>
</tbody>
</table>

Events
### Properties of the `TIBCTraceService` class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Eof</strong> (inherited from <code>TIBCControlAndQueryService</code>)</td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td><strong>ServiceParamBySPB</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BufferSize</strong> (inherited from <code>TIBCControlAndQueryService</code>)</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>Config</strong></td>
<td>Used to set the configuration of a trace session.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to display a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>Server</strong> <em>(inherited from <strong>TCustomIBCService</strong>)</em></td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SessionName</strong></td>
<td>Used to set session name for a trace session.</td>
</tr>
</tbody>
</table>

See Also

- **TIBCTraceService Class**
- **TIBCTraceService Class Members**

```plaintext
5.13.1.18.2.1  Config Property
```

Used to set the configuration of a trace session.

**Class**

**TIBCTraceService**

**Syntax**

```plaintext
property Config: TStrings;
```

**Remarks**

Use the Config property to set the configuration of a trace session.

The session configuration is a text conforming to the rules and syntax modelled in the fbtrace.conf template that is in Firebird's root directory, apart from the lines relating to placement of the output.

For example, the following configuration is used to trace prepare, free and execution of all statements in the mydatabase.fdb database:

```xml
<database mydatabase.fdb>
  enabled true
  log_statement_prepare true
  log_statement_free true
  log_statement_start true
  log_statement_finish true
  time_threshold 0
</database>
```

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

© 2021 Devart
5.13.1.18.2.2 SessionName Property

Used to set session name for a trace session.

Class

TIBCTraceService

Syntax

property SessionName: string;

Remarks

Set the SessionName property to distinguish your session in the list of trace sessions.

5.13.1.18.3 Methods

Methods of the TIBCTraceService class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>getNextChunk (inherited from TIBCControlAndQueryService)</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td>getNextLine (inherited from TIBCControlAndQueryService)</td>
<td>Returns the next line of data.</td>
</tr>
<tr>
<td>ListTraceSessions</td>
<td>Returns information about all trace sessions.</td>
</tr>
<tr>
<td>ResumeTrace</td>
<td>Resumes a trace session.</td>
</tr>
<tr>
<td>ServiceStart (inherited from TCustomIBCService)</td>
<td>Starts the service.</td>
</tr>
</tbody>
</table>
StartTrace       Starts the trace session.
StopTrace       Stops a trace session.
SuspendTrace    Suspends a trace session.

See Also
• TIBCTraceService Class
• TIBCTraceService Class Members

5.13.1.18.3.1 ListTraceSessions Method

Returns information about all trace sessions.

Class
TIBCTraceService

Syntax

procedure ListTraceSessions;

Remarks
Call the ListTraceSessions method to get information about all trace sessions. You can read this information from the service output using the GetNextChunk or GetNextLine methods. For each session the service returns a text message in the following format:
• Session ID: <number>
• name: <string>. Prints the trace session name if it is not empty
• user: <string>. Prints the user name of the user that created the trace session
• date: YYYY-MM-DD HH:NN:SS, start date and time of the user session
• flags: <string>
5.13.1.18.3.2 ResumeTrace Method

Resumes a trace session.

Class
TIBCTraceService

Syntax

```plaintext
procedure ResumeTrace(SessionID: integer);
```

Parameters

SessionID
Holds a session ID.

Remarks

Call the ResumeTrace method to resume the trace session with the specified ID. You can find the ID of the current trace session in the first lines of its output. Use ListTraceSessions to get ID for all trace sessions.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.18.3.3 StartTrace Method

Starts the trace session.

Class
TIBCTraceService

Syntax

```plaintext
procedure StartTrace;
```

Remarks

Call the StartTrace method to start the trace session. After the session is started you can read its output using the GetNextChunk or GetNextLine methods.

© 1997-2021 Devart. All Rights Reserved.
5.13.1.18.3.4  StopTrace Method

Stops a trace session.

Class
TIBCTraceService

Syntax

procedure StopTrace(SessionID: integer);

Parameters

SessionID
Holds a session ID.

Remarks

Call the StopTrace method to stop the trace session with the specified ID. You can find the ID of the current trace session in the first lines of its output. Use ListTraceSessions to get ID for all trace sessions.

5.13.1.18.3.5  SuspendTrace Method

Suspends a trace session.

Class
TIBCTraceService

Syntax

procedure SuspendTrace(SessionID: integer);

Parameters

SessionID
Holds a session ID.

Remarks

Call the SuspendTrace method to suspend the trace session with the specified ID. You can find ID of the current trace session in the first lines of its output. Use ListTraceSessions to get ID for all trace sessions.
5.13.1.19 TIBCUserInfo Class

TIBCUserInfo stores information about an InterBase user for the security service.

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

Unit
IBCAdmin

Syntax

TIBCUserInfo = class(TPersistent);

Remarks

TIBCUserInfo is the user descriptor that the InterBase security service uses to describe a single user.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveUser</td>
<td>Used to determine if the user is active.</td>
</tr>
<tr>
<td>DefaultRole</td>
<td>Holds the user default role.</td>
</tr>
<tr>
<td>Description</td>
<td>Holds the description.</td>
</tr>
<tr>
<td>FirstName</td>
<td>Holds the user's first name.</td>
</tr>
<tr>
<td>GroupID</td>
<td>Holds the user's group ID.</td>
</tr>
<tr>
<td>GroupName</td>
<td>Holds the name of the group.</td>
</tr>
<tr>
<td>LastName</td>
<td>Holds the user's Last name.</td>
</tr>
<tr>
<td>MiddleName</td>
<td>Holds the user's middle name.</td>
</tr>
</tbody>
</table>
### Properties

Properties of the **TIBCUserInfo** class.

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

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ActiveUser</strong></td>
<td>Used to determine if the user is active.</td>
</tr>
<tr>
<td><strong>DefaultRole</strong></td>
<td>Holds the user default role.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Holds the description.</td>
</tr>
<tr>
<td><strong>FirstName</strong></td>
<td>Holds the user's first name.</td>
</tr>
<tr>
<td><strong>GroupID</strong></td>
<td>Holds the user's group ID.</td>
</tr>
<tr>
<td><strong>GroupName</strong></td>
<td>Holds the name of the group.</td>
</tr>
<tr>
<td><strong>LastName</strong></td>
<td>Holds the user's Last name.</td>
</tr>
<tr>
<td><strong>MiddleName</strong></td>
<td>Holds the user's middle name.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Holds the user's password.</td>
</tr>
<tr>
<td><strong>SQLRole</strong></td>
<td>Holds an optional role to use when attaching to the security database.</td>
</tr>
<tr>
<td><strong>SystemUserName</strong></td>
<td>Holds the system user name.</td>
</tr>
</tbody>
</table>
### UserID

**Holds the user's individual ID.**

### UserName

**Holds the user's login name.**

#### See Also
- **TIBCUserInfo Class**
- **TIBCUserInfo Class Members**

---

5.13.1.19.2.1 **ActiveUser Property**

Used to determine if the user is active.

**Class**

**TIBCUserInfo**

**Syntax**

```pascal
property ActiveUser: Boolean default False;
```

**Remarks**

Use the `ActiveUser` property to find out if the user is active.

---

5.13.1.19.2.2 **DefaultRole Property**

Holds the user default role.

**Class**

**TIBCUserInfo**

**Syntax**

```pascal
property DefaultRole: string;
```

**Remarks**
The DefaultRole property holds the user default role.

Class

TIBCUserInfo

Syntax

```property Description: string;```

Remarks

The Description property holds the description.

5.13.1.19.2.4 FirstName Property

Holds the user's first name.

Class

TIBCUserInfo

Syntax

```property FirstName: string;```

Remarks

The FirstName property holds the user's first name.
5.13.1.19.2.5  GroupID Property

Holds the user's group ID.

Class

TIBCUserInfo

Syntax

```
property GroupID: Integer default 0;
```

Remarks

The GroupID property holds the user's group ID.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.19.2.6  GroupName Property

Holds the name of the group.

Class

TIBCUserInfo

Syntax

```
property GroupName: string;
```

Remarks

The GroupName property holds the name of the group.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.19.2.7  LastName Property

Holds the user's Last name.

Class

TIBCUserInfo
Syntax

```plaintext
property LastName: string;
```

Remarks

The `LastName` property holds the user's Last name.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.19.2.8 `MiddleName` Property

Holds the user's middle name.

Class

`TIBCUserInfo`

Syntax

```plaintext
property MiddleName: string;
```

Remarks

The `MiddleName` property holds the user's middle name.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.19.2.9 `Password` Property

Holds the user's password.

Class

`TIBCUserInfo`

Syntax

```plaintext
property Password: string;
```

Remarks

The `Password` property holds the user's password. The password can be maximum 31 characters, only first 8 characters are significant.
5.13.1.19.2.10  SQLRole Property

Holds an optional role to use when attaching to the security database.

Class
TIBCUserInfo

Syntax

```
property SQLRole: string;
```

Remarks
The SQLRole property is an optional role to use when attaching to the security database.

5.13.1.19.2.11  SystemUserName Property

Holds the system user name.

Class
TIBCUserInfo

Syntax

```
property SystemUserName: string;
```

Remarks
The SystemUserName property holds the system user name.

5.13.1.19.2.12  UserID Property

Holds the user's individual ID.

Class
TIBCUserInfo

Syntax

```property UserID: Integer default 0;```

Remarks

The UserID property holds the user's individual ID.

5.13.19.2.13 UserName Property

Holds the user's login name.

Class

TIBCUserInfo

Syntax

```property UserName: string;```

Remarks

The UserName property holds the user's login name.

5.13.20 TIBCValidationService Class

Used to validate a database and reconcile database transactions.

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

Unit

IBCAadmin

Syntax

```TIBCValidationService = class(TIBCControlAndQueryService);```
Remarks
Use a TIBCValidationService object to validate a database and reconcile database transactions.

Inheritance Hierarchy

TCustomIBCService
  TIBCControlAndQueryService
  TIBCValidationService

© 1997-2021 Devart. All Rights Reserved.

5.13.1.20.1 Members

**TIBCValidationService** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from TCustomIBCService)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BufferSize</strong> (inherited from TIBCControlAndQueryService)</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Used to set or return the database name.</td>
</tr>
<tr>
<td><strong>Eof</strong> (inherited from TIBCControlAndQueryService)</td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td><strong>GlobalAction</strong></td>
<td>Used to set or return the global transaction action.</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from TCustomIBCService)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td><strong>LimboTransactionInfo</strong></td>
<td>Used to return the limbo transaction information.</td>
</tr>
<tr>
<td><strong>LimboTransactionInfoCount</strong></td>
<td>Used to set or return the number of elements in the TIBCLimboTransactionInfo array.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from TCustomIBCService)</td>
<td>Used to dispaly a login prompt before attaching to a database.</td>
</tr>
</tbody>
</table>
### Options
- **Params** (inherited from `TCustomIBCService`)
  - Used to set or return database parameters.
- **Protocol** (inherited from `TCustomIBCService`)
  - Used to select the network protocol.
- **RecoverTwoPhaseGlobal**
  - Used to restore two-phase transactions.
- **Server** (inherited from `TCustomIBCService`)
  - Used to set the name of the server on which the services are to be run.
- **ServiceParamBySPB** (inherited from `TCustomIBCService`)
  - Used to return and sets SPB parameters.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attach</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td><strong>Detach</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td><strong>FetchLimboTransactionInfo</strong></td>
<td>Gets limbo transaction information.</td>
</tr>
<tr>
<td><strong>FixLimboTransactionErrors</strong></td>
<td>Fixes limbo transaction errors.</td>
</tr>
<tr>
<td><strong>GetNextChunk</strong> (inherited from <code>TIBCControlAndQueryService</code>)</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td><strong>GetNextLine</strong> (inherited from <code>TIBCControlAndQueryService</code>)</td>
<td>Returns the next line of data.</td>
</tr>
<tr>
<td><strong>ServiceStart</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Starts the service.</td>
</tr>
<tr>
<td><strong>SweepDatabase</strong></td>
<td>Performs a database sweep.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnAttach</strong> (inherited from <code>TCustomIBCService</code>)</td>
<td>Occurs when the database is attached.</td>
</tr>
</tbody>
</table>
Reserved.

5.13.1.20.2 Properties

Properties of the **TIBCValidationService** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Eof</strong> (inherited from <a href="#">TIBCControlAndQueryService</a>)</td>
<td>Used to determine whether the end of the file has been reached.</td>
</tr>
<tr>
<td><strong>Handle</strong> (inherited from <a href="#">TCustomIBCService</a>)</td>
<td>Used to return the database handle.</td>
</tr>
<tr>
<td><strong>LimboTransactionInfo</strong></td>
<td>Used to return the limbo transaction information.</td>
</tr>
<tr>
<td><strong>LimboTransactionInfoCount</strong></td>
<td>Used to set or return the number of elements in the TIBCLimboTransactionInfo array.</td>
</tr>
<tr>
<td><strong>ServiceParamBySPB</strong> (inherited from <a href="#">TCustomIBCService</a>)</td>
<td>Used to return and sets SPB parameters.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from <a href="#">TCustomIBCService</a>)</td>
<td>Used to set the service to active or inactive (default).</td>
</tr>
<tr>
<td><strong>BufferSize</strong> (inherited from <a href="#">TIBCControlAndQueryService</a>)</td>
<td>Used to set or return the buffer size.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Used to set or return the database name.</td>
</tr>
<tr>
<td><strong>GlobalAction</strong></td>
<td>Used to set or return the global transaction action.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from <a href="#">TCustomIBCService</a>)</td>
<td>Used to dispaly a login prompt before attaching to a database.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to invoke database validation.</td>
</tr>
</tbody>
</table>
### InterBase Data Access Components

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Params</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to set or return database parameters.</td>
</tr>
<tr>
<td><strong>Protocol</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to select the network protocol.</td>
</tr>
<tr>
<td><strong>RecoverTwoPhaseGlobal</strong></td>
<td>Used to restore two-phase transactions.</td>
</tr>
<tr>
<td><strong>Server</strong> (inherited from <strong>TCustomIBCService</strong>)</td>
<td>Used to set the name of the server on which the services are to be run.</td>
</tr>
</tbody>
</table>

#### See Also
- **TIBCValidationService Class**
- **TIBCValidationService Class Members**

© 1997-2021 Devart. All Rights Reserved.

---

#### 5.13.1.20.2.1 Database Property

Used to set or return the database name.

**Class**

**TIBCValidationService**

**Syntax**

`property Database: string;`

**Remarks**

Use the Database property to set or return the database name to set properties on.

© 1997-2021 Devart. All Rights Reserved.

---

#### 5.13.1.20.2.2 GlobalAction Property

Used to set or return the global transaction action.

**Class**

**TIBCValidationService**
Syntax

**property** GlobalAction: `TIBCTransactionGlobalAction` **default**

tgNoGlobalAction;

Remarks

Use the GlobalAction property to set or return the global transaction action. This value indicates what action to take with the transactions that follow.

**Note:** You should install InterBase 6 to use this feature.

5.13.1.20.2.3  LimboTransactionInfo Property (Indexer)

Used to return the limbo transaction information.

Class

`TIBCValidationService`

Syntax

**property** LimboTransactionInfo[Index: Integer]: `TIBCLimboTransactionInfo`;

Parameters

`Index`

Remarks

Use the LimboTransactionInfo property to return the limbo transaction information.

**Note:** You should install InterBase 6 to use this feature.

5.13.1.20.2.4  LimboTransactionInfoCount Property

Used to set or return the number of elements in the TIBCLimboTransactionInfo array.

Class
**TIBCVValidationService**

**Syntax**

```nim
property LimboTransactionInfoCount: Integer;
```

**Remarks**

Use the LimboTransactionInfoCount property to set or return the number of elements in the TIBCLimboTransactionInfo array.

**Note:** You should install InterBase 6 to use this feature.

---

**5.13.1.20.2.5 Options Property**

Used to invoke database validation.

**Class**

**TIBCVValidationService**

**Syntax**

```nim
property Options: TIBCVValidateOptions default [];`n```

**Remarks**

Use the Options property of the TIBCVValidationService component to invoke database validation.

Set any of the following options of type TValidateOption to True to perform the appropriate validation:

**Note:** Not all combinations of validation options work together. For example, you cannot simultaneously mend and validate the database at the same time. Conversely, some options are intended to be used with other options, such as ValidateDB, or ValidateFull with ValidateDB.
5.13.1.20.2.6 RecoverTwoPhaseGlobal Property

Used to restore two-phase transactions.

Class

TIBCValidationService

Syntax

property RecoverTwoPhaseGlobal: Boolean;

Remarks

Use the RecoverTwoPhaseGlobal property to restore two-phase transactions.

RecoverTwoPhaseGlobal performs automated two-phase recovery, either for a limbo transaction specified by ID or for all limbo transactions.

© 1997-2021 Devart. All Rights Reserved.

5.13.1.20.3 Methods

Methods of the TIBCValidationService class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach (inherited from TCustomIBCService)</td>
<td>Attaches to the database.</td>
</tr>
<tr>
<td>Detach (inherited from TCustomIBCService)</td>
<td>Detaches from the database.</td>
</tr>
<tr>
<td>FetchLimboTransactionInfo</td>
<td>Gets limbo transaction information.</td>
</tr>
<tr>
<td>FixLimboTransactionErrors</td>
<td>Fixes limbo transaction errors.</td>
</tr>
<tr>
<td>GetNextChunk (inherited from TIBCControlAndQueryService)</td>
<td>Returns the next chunk of data.</td>
</tr>
<tr>
<td>GetNextLine (inherited from TIBCControlAndQueryService)</td>
<td>Returns the next line of data.</td>
</tr>
</tbody>
</table>
ServiceStart (inherited from TCustomIBCService)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceStart</td>
<td>Starts the service.</td>
</tr>
<tr>
<td>SweepDatabase</td>
<td>Performs a database sweep.</td>
</tr>
</tbody>
</table>

See Also

- TIBCValidationService Class
- TIBCValidationService Class Members

Get limbo transaction information.

Class

TIBCValidationService

Syntax

```plaintext
procedure FetchLimboTransactionInfo;
```

Remarks

Call the FetchLimboTransactionInfo method to get limbo transaction information from the LimboTransactionInfo property.

Note: You should install InterBase 6 to use this feature.

Fixes limbo transaction errors.

Class

TIBCValidationService

Syntax

```plaintext
procedure FixLimboTransactionErrors;
```
Remarks

Call the FixLimboTransactionErrors method to repair limbo transaction errors.

**Note**: You should install InterBase 6 to use this feature.

5.13.1.20.3.3 SweepDatabase Method

Performs a database sweep.

**Class**

*TIBCValidationService*

**Syntax**

```delphi
procedure SweepDatabase;
```

**Remarks**

Call the SweepDatabase method to perform a database sweep.

5.13.1.21 TIBCVersionInfo Class

Represents the version information about an InterBase server.

For a list of all members of this type, see *TIBCVersionInfo* members.

**Unit**

*IBCAadmin*

**Syntax**

```delphi
TIBCVersionInfo = class(System TObject);
```

**Remarks**

TIBCVersionInfo represents the version information about an InterBase server.
5.13.1.21.1 Members

**TIBCVersionInfo** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerImplementation</td>
<td>Used to retrieve the implementation string.</td>
</tr>
<tr>
<td>ServerVersion</td>
<td>Used to retrieve the version of the InterBase server.</td>
</tr>
<tr>
<td>ServiceVersion</td>
<td>Used to retrieve the version number for the service that fetches the version information.</td>
</tr>
</tbody>
</table>

5.13.1.21.2 Properties

Properties of the **TIBCVersionInfo** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerImplementation</td>
<td>Used to retrieve the implementation string.</td>
</tr>
<tr>
<td>ServerVersion</td>
<td>Used to retrieve the version of the InterBase server.</td>
</tr>
<tr>
<td>ServiceVersion</td>
<td>Used to retrieve the version number for the service that fetches the version information.</td>
</tr>
</tbody>
</table>

See Also

- **TIBCVersionInfo Class**
5.13.1.21.2.1 ServerImplementation Property

Used to retrieve the implementation string.

Class

**TIBCVersionInfo**

Syntax

```property ServerImplementation: string;```

Remarks

Use the ServerImplementation property to retrieve the implementation string.

5.13.1.21.2.2 ServerVersion Property

Used to retrieve the version of the InterBase server.

Class

**TIBCVersionInfo**

Syntax

```property ServerVersion: string;```

Remarks

Use the ServerVersion property to retrieve the version of the InterBase server.
5.13.1.21.2.3 ServiceVersion Property

Used to retrieve the version number for the service that fetches the version information.

Class

_TIBCVersionInfo_

Syntax

```property serviceVersion: integer;```

Remarks

Use the ServiceVersion property to retrieve the version number for the service that fetches the version information.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>TIBCBackupOptions</em></td>
<td>Represents the set of <em>TIBCBackupOption</em>.</td>
</tr>
<tr>
<td><em>TIBCNBackupOptions</em></td>
<td>Represents the set of <em>TIBCNBackupOption</em>.</td>
</tr>
<tr>
<td><em>TIBCRestoreOptions</em></td>
<td>Represents the set of <em>TIBCRestoreOption</em>.</td>
</tr>
<tr>
<td><em>TIBCStatOptions</em></td>
<td>Represents the set of <em>TIBCStatOption</em>.</td>
</tr>
<tr>
<td><em>TIBCValidateOptions</em></td>
<td>Represents the set of <em>TIBCValidateOption</em>.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.

Request Support       DAC Forum       Provide Feedback
5.13.2.1 TIBCBackupOptions Set

Represents the set of TIBCBackupOption.

Unit
IBCAdmin

Syntax

TIBCBackupOptions = set of TIBCBackupOption;

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

5.13.2.2 TIBCNBackupOptions Set

Represents the set of TIBCNBackupOption.

Unit
IBCAdmin

Syntax

TIBCNBackupOptions = set of TIBCNBackupOption;

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

5.13.2.3 TIBCRestoreOptions Set

Represents the set of TIBCRestoreOption.

Unit
IBCAdmin

Syntax

TIBCRestoreOptions = set of TIBCRestoreOption;

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

Represents the set of TIBCStatOption.

Unit

IBCAdmin

Syntax

TIBCStatOptions = set of TIBCStatOption;

5.13.2.5 TIBCValidateOptions Set

Represents the set of TIBCValidateOption.

Unit

IBCAdmin

Syntax

TIBCValidateOptions = set of TIBCValidateOption;

5.13.3 Enumerations

Enumerations in the IBCAdmin unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCBackupOption</td>
<td>Allows you to build backup options into your application.</td>
</tr>
<tr>
<td>TIBCLicensingAction</td>
<td>Allows to add or remove an InterBase software activation certificate.</td>
</tr>
<tr>
<td>TIBCNBackupOption</td>
<td>Options used in TIBCBackupRestoreService for nBackup.</td>
</tr>
</tbody>
</table>
### TIBCBackupOption Enumeration

Allows you to build backup options into your application.

**Unit**

`IBCAadmin`

**Syntax**

```plaintext
TIBCBackupOption = (boIgnoreChecksums, boIgnoreLimbo,
boMetadataOnly, boNoGarbageCollection, boOldMetadataDesc,
boNonTransportable, boConvertExtTables);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>boConvertExtTables</td>
<td>Convert external table data to internal tables.</td>
</tr>
<tr>
<td>boIgnoreChecksums</td>
<td>Ignore checksums during backup.</td>
</tr>
</tbody>
</table>
bolgnoreLimbo | Ignored limbo transactions during backup.
boMetadataOnly | Output backup file for metadata only with empty tables.
boNoGarbageCollection | Suppress normal garbage collection during backup; improves performance on some databases.
boNonTransportable | Output backup file with non-XDR data format; improves space and performance by a negligible amount.
boOldMetadataDesc | Output metadata in pre-4.0 format.

5.13.3.2 TIBCLicensingAction Enumeration

Allows to add or remove an InterBase software activation certificate.

Unit
IBCAdmin

Syntax
TIBCLicensingAction = (laAdd, laRemove);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>laAdd</td>
<td>Adds an InterBase software activation certificate.</td>
</tr>
<tr>
<td>laRemove</td>
<td>Removes an InterBase software activation certificate.</td>
</tr>
</tbody>
</table>

5.13.3.3 TIBCNBackupOption Enumeration

Options used in TIBCBackupRestoreService for nBackup.

Unit
IBCAdmin

Syntax
TIBCNBackupOption = (nboNoTriggers);
5.13.3.4 TIBCRestoreOption Enumeration

Specifies the data restore parameters.

Unit
IBCAdmin

Syntax

TIBCRestoreOption = (roDeactivateIndexes, roNoShadow, roNoValidityCheck, roOneRelationAtATime, roReplace, roCreateNewDB, roUseAllSpace, roValidationCheck);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>roCreateNewDB</td>
<td>Restore but do not overwrite an existing database (restores the database from the backup copy into a new file).</td>
</tr>
<tr>
<td>roDeactivateIndexes</td>
<td>Do not build indexes during restore. InterBase rebuilds indexes while restoring a database in the normal mode of operation. If there are duplicates of the records that are unique in the database, the restore process will be terminated. Duplicate records can be added to the database if the unique index is temporary turned off.</td>
</tr>
<tr>
<td>roNoShadow</td>
<td>Do not recreate shadow files during restore. In the normal mode of operation InterBase recreates shadow files during restore. By setting this value such option will be blocked.</td>
</tr>
<tr>
<td>roNoValidityCheck</td>
<td>Do not enforce validity conditions (for example, NOT NULL) during restore. This value should be used when restoring a database that includes records in a wrong format. If you set this value, the correspondence of data to the existing limitations would not be checked.</td>
</tr>
<tr>
<td>roOneRelationAtATime</td>
<td>Commit after completing a restore of each table.</td>
</tr>
<tr>
<td>roReplace</td>
<td>Replace database if one exists. If you do not use this value, the...</td>
</tr>
</tbody>
</table>
5.13.3.5 TIBCSecurityAction Enumeration

Specify the type of the operation for InterBase Security Service to perform.

Unit
IBCAdmin

Syntax
TIBCSecurityAction = (saAddUser, saDeleteUser, saModifyUser, saDisplayUser);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>saAddUser</td>
<td>New user account will be added.</td>
</tr>
<tr>
<td>saDeleteUser</td>
<td>Existing user account will be deleted.</td>
</tr>
<tr>
<td>saDisplayUser</td>
<td>Available information on the user account will be displayed.</td>
</tr>
<tr>
<td>saModifyUser</td>
<td>The user account will be modified.</td>
</tr>
</tbody>
</table>

5.13.3.6 TIBCStatOption Enumeration

Allows to specify the way the database statistics would be requested.

Unit
IBCAdmin

Syntax
TIBCStatOption = (soDataPages, soDbLog, soHeaderPages, soIndexPages, soSystemRelations, soRecordVersions, soStatTables);
Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>soDataPages</td>
<td>Request statistics for data tables in the database.</td>
</tr>
<tr>
<td>soDbLog</td>
<td>Stop reporting statistics after reporting information on the log pages.</td>
</tr>
<tr>
<td>soHeaderPages</td>
<td>Stop reporting statistics after reporting the information on the header page.</td>
</tr>
<tr>
<td>soIndexPages</td>
<td>Request statistics for the user indexes in the database.</td>
</tr>
<tr>
<td>soRecordVersions</td>
<td>Request statistics for the record versions.</td>
</tr>
<tr>
<td>soStatTables</td>
<td>Request statistics for tables specified in the TableNames property.</td>
</tr>
<tr>
<td>soSystemRelations</td>
<td>Request statistics for system tables and indexes in addition to user tables and indexes.</td>
</tr>
</tbody>
</table>

5.13.3.7 TIBCTransactionAdvise Enumeration

Allows to specify the suggested action for ending the transaction.

Unit

IBCAdmin

Syntax

TIBCTransactionAdvise = (taCommitAdvise, taRollbackAdvise, taUnknownAdvise);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>taCommitAdvise</td>
<td>Commit the transaction.</td>
</tr>
<tr>
<td>taRollbackAdvise</td>
<td>Rollback the transaction.</td>
</tr>
<tr>
<td>taUnknownAdvise</td>
<td>No suggested action is specified for ending the transaction.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.  

Request Support  DAC Forum  Provide Feedback
5.13.3.8 TIBCTransactionGlobalAction Enumeration

Determines which action to take concerning limbo transactions.

Unit
IBCAdmin

Syntax

TIBCTransactionGlobalAction = (tgNoGlobalAction, tgCommitGlobal, tgRollbackGlobal);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tgCommitGlobal</td>
<td>Commits the limbo transaction specified by ID or commits all limbo transactions.</td>
</tr>
<tr>
<td>tgNoGlobalAction</td>
<td>Takes no action.</td>
</tr>
<tr>
<td>tgRollbackGlobal</td>
<td>Rolls back the limbo transaction specified by ID or rolls back all limbo transactions.</td>
</tr>
</tbody>
</table>

5.13.3.9 TIBCTransactionState Enumeration

Allows to specify the current state of the limbo transaction.

Unit
IBCAdmin

Syntax

TIBCTransactionState = (tsLimboState, tsCommitState, tsRollbackState, tsUnknownState);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsCommitState</td>
<td>The transaction is in commit state.</td>
</tr>
<tr>
<td>tsLimboState</td>
<td>The limbo transaction is in limbo state.</td>
</tr>
<tr>
<td>tsRollbackState</td>
<td>The limbo transaction is in rollback state.</td>
</tr>
</tbody>
</table>
5.13.3.10 TIBCValidateOption Enumeration

Sets the options of validation.

Unit

IBCAdmin

Syntax

TIBCValidateOption = (voCheckDB, voIgnoreChecksum, voKillShadows, voMendDB, voValidateDB, voValidateFull);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>voCheckDB</td>
<td>Request a read-only validation of the database without correcting any problems.</td>
</tr>
<tr>
<td>voIgnoreChecksum</td>
<td>Ignore all checksum errors when validating or sweeping.</td>
</tr>
<tr>
<td>voKillShadows</td>
<td>Remove references to unavailable shadow files.</td>
</tr>
<tr>
<td>voMendDB</td>
<td>Mark corrupted records as unavailable so that subsequent operations skip them.</td>
</tr>
<tr>
<td>voValidateDB</td>
<td>Locate and release pages that are allocated but unassigned to any data structures.</td>
</tr>
<tr>
<td>voValidateFull</td>
<td>Check record and page structures, releasing unassigned record fragments; use with ValidateDB.</td>
</tr>
</tbody>
</table>

5.14 IBCAlerter

This unit contains implementation of the TIBCAlerter component.

Classes
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCAlerter</td>
<td>A component for transferring messages between connections.</td>
</tr>
</tbody>
</table>

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCAlerterEvent</td>
<td>This type is used for the TIBCAlerter.OnEvent event.</td>
</tr>
</tbody>
</table>

5.14.1 Classes

Classes in the `IBCAlerter` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCAlerter</td>
<td>A component for transferring messages between connections.</td>
</tr>
</tbody>
</table>

5.14.1.1 TIBCAlerter Class

A component for transferring messages between connections.

For a list of all members of this type, see `TIBCAlerter` members.

Unit

`IBCAlerter`

Syntax

```
TIBCAlerter = class(TDAAlerter);
```

Remarks
The TIBCAlerter component allows you to register interest in and asynchronously handle events posted by an InterBase server. Use TIBCAlerter 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 **TIBCAlerter.Events** property to required events and call `M:Devart.IbDac.TIBCAlerter.Start()` method. When one of the registered events occurs **TIBCAlerter.OnEvent** handler is called.

**Note:** Events are transaction-based. This means that the waiting connection does not get event until the transaction posting the event commits.

### Inheritance Hierarchy

- **TDAAlerter**
- **TIBCAlerter**

---

**TIBCAlerter** class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong></td>
<td>(inherited from <strong>TDAAlerter</strong>) Used to determine if TDAAlerter waits for messages.</td>
</tr>
<tr>
<td><strong>AutoCommit</strong></td>
<td>Used to automatically commit transaction after calling the SendEvent method.</td>
</tr>
<tr>
<td><strong>AutoRegister</strong></td>
<td>(inherited from <strong>TDAAlerter</strong>) Used to automatically register events whenever connection opens.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify the connection for TIBCAlerter.</td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td>Used to specify the names of events to wait.</td>
</tr>
<tr>
<td><strong>Transaction</strong></td>
<td>Used to set the transaction to be used by the SendEvent method.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendEvent</td>
<td>Sends an event with Name.</td>
</tr>
<tr>
<td>Start (inherited from TDAAlerter)</td>
<td>Starts waiting process.</td>
</tr>
<tr>
<td>Stop (inherited from TDAAlerter)</td>
<td>Stops waiting process.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnError (inherited from TDAAlerter)</td>
<td>Occurs if an exception occurs in waiting process</td>
</tr>
<tr>
<td>OnEvent</td>
<td>Occurs when waiting process receives an event from the InterBase server.</td>
</tr>
</tbody>
</table>

Properties of the TIBCAlerter class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active (inherited from TDAAlerter)</td>
<td>Used to determine if TDAAlerter waits for messages.</td>
</tr>
<tr>
<td>AutoRegister (inherited from TDAAlerter)</td>
<td>Used to automatically register events whenever connection opens.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCommit</td>
<td>Used to automatically commit transaction after calling the SendEvent</td>
</tr>
</tbody>
</table>
5.14.1.1.2.1 AutoCommit Property

Used to automatically commit transaction after calling the SendEvent method.

Class
TIBCAlerter

Syntax
property AutoCommit: boolean;

Remarks
Use the AutoCommit property to automatically commit transaction after calling the SendEvent method. Events are transaction-based. This means that the waiting connection does not get event until the transaction posting the event commits.

5.14.1.1.2.2 Connection Property

Used to specify the connection for TIBCAlerter.

Class
TIBCAlerter
Syntaex

```
property Connection: TIBCConnection;
```

Remarks

Use the Connection property to specify the connection for TIBCAlerter.

See Also

- TIBCConnection

5.14.1.1.2.3 Events Property

Used to specify the names of events to wait.

Class

TIBCAlerter

Syntaex

```
property Events: TStrings;
```

Remarks

Use the Events property to set the names of events to wait.

5.14.1.1.2.4 Transaction Property

Used to set the transaction to be used by the SendEvent method.

Class

TIBCAlerter

Syntaex

```
property Transaction: TIBCTransaction stored IsTransactionStored;
```
Remarks

Use the Transaction property to set the transaction to be used by the SendEvent method. Events are transaction-based. This means that the waiting connection does not get event until the transaction posting the event commits.

5.14.1.3 Methods

Methods of the TIBCAlerter class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendEvent</td>
<td>Sends an event with Name.</td>
</tr>
<tr>
<td>Start (inherited from TDAAlerter)</td>
<td>Starts waiting process.</td>
</tr>
<tr>
<td>Stop (inherited from TDAAlerter)</td>
<td>Stops waiting process.</td>
</tr>
</tbody>
</table>

See Also

- TIBCAlerter Class
- TIBCAlerter Class Members

5.14.1.3.1 SendEvent Method

Sends an event with Name.

Class

TIBCAlerter

Syntax

```plaintext
procedure SendEvent(const Name: string);
```

Parameters
Name
Holds the name of the event to send.

Remarks
Call the SendEvent procedure to send an event with Name.

This procedure is supported only for Firebird 2.0 and higher.

Events of the TIBCAlerter class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnError</td>
<td>(inherited from TDAAlerter) Occurs if an exception occurs in waiting process</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnEvent</td>
<td>Occurs when waiting process receives an event from the InterBase server.</td>
</tr>
</tbody>
</table>

See Also
- TIBCAlerter Class
- TIBCAlerter Class Members

Class
TIBCAlerter

Syntax

```property
OnEvent: TIBCAlertEvent;
```

Remarks

The OnEvent event occurs when waiting process receives event from the InterBase server. The EventName parameter contains the name of the event and the EventCount parameter contains the number of events that were raised in the transaction. Waiting connection does not get event until the transaction signaling the alert commits and there can be several event raises in one transaction.

5.14.2 Types

Types in the IBCAlerter unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCAlertEvent</td>
<td>This type is used for the TIBCAlerter.OnEvent event.</td>
</tr>
</tbody>
</table>

5.14.2.1 TIBCAlertEvent Procedure Reference

This type is used for the TIBCAlerter.OnEvent event.

Unit

IBCAlerter

Syntax

```TIBCAlertEvent = procedure (Sender: TObject; EventName: string; EventCount: Integer) of object;```

Parameters
Sender
An object that raised the event.

EventName
Holds the name of the event.

EventCount
Holds the number of events that were raised in the transaction.

5.15 IBCArray

This unit contains the TIBCArray class for representing the value of the InterBase array data type.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomIBCArray</td>
<td>A base class representing the value of the InterBase array data type.</td>
</tr>
</tbody>
</table>

5.15.1 Classes

Classes in the IBCArray unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomIBCArray</td>
<td>A base class representing the value of the InterBase array data type.</td>
</tr>
</tbody>
</table>
5.15.1.1 TCustomIBCArray Class

A base class representing the value of the InterBase array data type.

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

Unit
IBCArray

Syntax

TCustomIBCArray = class(TDBObject);

Inheritance Hierarchy

TSharedObject
TDBObject
TCustomIBCArray

© 1997-2021
Devart. All Rights Reserved.

5.15.1.1.1 Members

TCustomIBCArray class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayDimensions</td>
<td>Contains the array dimensions count.</td>
</tr>
<tr>
<td>ArrayHighBound</td>
<td>Used to get or set the upper boundary of the defined dimension subscript.</td>
</tr>
<tr>
<td>ArrayID</td>
<td>Contains the array ID.</td>
</tr>
<tr>
<td>ArrayLowBound</td>
<td>Used to get or set the lower boundary of the defined dimension subscript.</td>
</tr>
<tr>
<td>ArraySize</td>
<td>Used to determine the size of the whole array data in bytes.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to return array as string.</td>
</tr>
<tr>
<td><strong>Cached</strong></td>
<td>Indicates whether to cache array data on the client side.</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td><strong>CachedDimensions</strong></td>
<td>Contains cached array dimensions count.</td>
</tr>
<tr>
<td><strong>CachedHighBound</strong></td>
<td>Used to get the upper boundary of defined dimension subscript of cached array elements.</td>
</tr>
<tr>
<td><strong>CachedLowBound</strong></td>
<td>Used to get the lower boundary of the defined dimension subscript of cached array elements.</td>
</tr>
<tr>
<td><strong>CachedSize</strong></td>
<td>Used to get the cached array data size in bytes.</td>
</tr>
<tr>
<td><strong>ColumnName</strong></td>
<td>Used to get or set the name of the table column that has array type.</td>
</tr>
<tr>
<td><strong>DbHandle</strong></td>
<td>Contains the handle of a database where the array is stored.</td>
</tr>
<tr>
<td><strong>isNull</strong></td>
<td>Used to define whether the array field in the database is null.</td>
</tr>
<tr>
<td><strong>Items</strong></td>
<td>Used to get or set array items.</td>
</tr>
<tr>
<td><strong>ItemScale</strong></td>
<td>Used to get or set the scale for array items for the NUMERIC and DECIMAL datatypes.</td>
</tr>
<tr>
<td><strong>ItemSize</strong></td>
<td>Contains the size of an array item.</td>
</tr>
<tr>
<td><strong>Modified</strong></td>
<td>Used to indicate if the modifications done in cache were saved to the database.</td>
</tr>
<tr>
<td><strong>RefCount</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>** TableName**</td>
<td>Used to set the name of the table containing an array field.</td>
</tr>
<tr>
<td><strong>TrHandle</strong></td>
<td>Contains the handle of the transaction in which the array is read or written.</td>
</tr>
</tbody>
</table>
# Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Copies the Source object content to the current one.</td>
</tr>
<tr>
<td>ClearArray</td>
<td>Clears all array values on the server if Cached is set to False.</td>
</tr>
<tr>
<td>CreateTemporaryArray</td>
<td>Creates a temporary array on the InterBase server.</td>
</tr>
<tr>
<td>GetArrayInfo</td>
<td>Used to get array descriptor.</td>
</tr>
<tr>
<td>GetItemAsDateTime</td>
<td>Reads the value of an array item into an object or variable of the TDateTime type.</td>
</tr>
<tr>
<td>GetItemAsFloat</td>
<td>Reads the value of an array item into a floating-point number.</td>
</tr>
<tr>
<td>GetItemAsInteger</td>
<td>Reads the value of an array item into an integer.</td>
</tr>
<tr>
<td>GetItemAsSmallInt</td>
<td>Reads the value of an array item into a short integer.</td>
</tr>
<tr>
<td>GetItemAsString</td>
<td>Reads the value of an array item into a string.</td>
</tr>
<tr>
<td>GetItemAsWideString</td>
<td>Reads the value of an array item into a WideString.</td>
</tr>
<tr>
<td>GetItemsSlice</td>
<td>Returns the array slice items' values.</td>
</tr>
<tr>
<td>GetItemValue</td>
<td>Returns the array item value.</td>
</tr>
<tr>
<td>ReadArray</td>
<td>Reads an array from the database to memory.</td>
</tr>
<tr>
<td>ReadArrayItem</td>
<td>Reads the array item specified by indices from the database to memory.</td>
</tr>
<tr>
<td>ReadArraySlice</td>
<td>Reads array slice from the database to memory.</td>
</tr>
<tr>
<td>Release (inherited from <strong>TSharedObject</strong>)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>
### Properties

Properties of the `TCustomIBCArray` class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArrayDimensions</code></td>
<td>Contains the array dimensions count.</td>
</tr>
<tr>
<td><code>ArrayHighBound</code></td>
<td>Used to get or set the upper boundary of the defined dimension subscript.</td>
</tr>
<tr>
<td><code>ArrayID</code></td>
<td>Contains the array ID.</td>
</tr>
<tr>
<td><code>ArrayLowBound</code></td>
<td>Used to get or set the lower boundary of the defined</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ArraySize</strong></td>
<td>Used to determine the size of the whole array data in bytes.</td>
</tr>
<tr>
<td><strong>AsString</strong></td>
<td>Used to return array as string.</td>
</tr>
<tr>
<td><strong>Cached</strong></td>
<td>Indicates whether to cache array data on the client side.</td>
</tr>
<tr>
<td><strong>CachedDimensions</strong></td>
<td>Contains cached array dimensions count.</td>
</tr>
<tr>
<td><strong>CachedHighBound</strong></td>
<td>Used to get the upper boundary of defined dimension subscript of cached array elements.</td>
</tr>
<tr>
<td><strong>CachedLowBound</strong></td>
<td>Used to get the lower boundary of the defined dimension subscript of cached array elements.</td>
</tr>
<tr>
<td><strong>CachedSize</strong></td>
<td>Used to get the cached array data size in bytes.</td>
</tr>
<tr>
<td><strong>ColumnName</strong></td>
<td>Used to get or set the name of the table column that has array type.</td>
</tr>
<tr>
<td><strong>DbHandle</strong></td>
<td>Contains the handle of a database where the array is stored.</td>
</tr>
<tr>
<td><strong>IsNull</strong></td>
<td>Used to define whether the array field in the database is null.</td>
</tr>
<tr>
<td><strong>Items</strong></td>
<td>Used to get or set array items.</td>
</tr>
<tr>
<td><strong>ItemScale</strong></td>
<td>Used to get or set the scale for array items for the NUMERIC and DECIMAL datatypes.</td>
</tr>
<tr>
<td><strong>ItemSize</strong></td>
<td>Contains the size of an array item.</td>
</tr>
<tr>
<td><strong>Modified</strong></td>
<td>Used to indicate if the modifications done in cache were saved to the database.</td>
</tr>
<tr>
<td><strong>RefCount</strong></td>
<td>(inherited from <strong>TSharedObject</strong>) Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td><strong>TableName</strong></td>
<td>Used to set the name of the</td>
</tr>
</tbody>
</table>
### ArrayDimensions Property

Contains the array dimensions count.

**Class**

`TCustomIBCArray`

**Syntax**

```pascal
property ArrayDimensions: integer;
```

**Remarks**

The `ArrayDimensions` property is used to hold the array dimensions count. InterBase supports multi-dimensional arrays with 1 to 16 dimensions.

---

### ArrayHighBound Property (Indexer)

Used to get or set the upper boundary of the defined dimension subscript.

**Class**

`TCustomIBCArray`

**Syntax**

```pascal
property ArrayHighBound[Dimension: integer]: integer;
```
**Parameters**

*Dimension*

Holds the dimension subscript.

**Remarks**

Use the ArrayHighBound property to get or set the upper boundary of the defined dimension subscript.

**See Also**

- [ArrayLowBound](#)
- [ArrayDimensions](#)

---

**5.15.1.1.2.3 ArrayID Property**

Contains the array ID.

**Class**

*TCustomIBCArray*

**Syntax**

```
property ArrayID: TISC_QUAD;
```

**Remarks**

The ArrayID property is used to hold the array ID that is actually stored in the array field of the table record. It is a unique numeric value that references the array data.

---

**5.15.1.1.2.4 ArrayLowBound Property(Indexer)**

Used to get or set the lower boundary of the defined dimension subscript.

**Class**

*TCustomIBCArray*
**Syntax**

```pascal
property ArrayLowBound[Dimension: integer]: integer;
```

**Parameters**

*Dimension*

Holds the dimension subscript.

**Remarks**

Use the `ArrayLowBound` property to get or set the lower boundary of the defined dimension subscript.

**See Also**

- `ArrayHighBound`
- `ArrayDimensions`

---

**5.15.1.1.2.5 ArraySize Property**

Used to determine the size of the whole array data in bytes.

**Class**

`TCustomIBCArray`

**Syntax**

```pascal
property ArraySize: integer;
```

**Remarks**

Use the `ArraySize` property to find out the size of the whole array data in bytes.

---

**5.15.1.1.2.6 AsString Property**

Used to return array as string.

**Class**
**TCustomIBCArr**

**Syntax**

```property AsString: string;```

**Remarks**

Use the AsString property to return array as string. For example, AsString property for two-dimensional array of integer with 2 rows, 3 elements in width can have the following value:

'((1; 2; 3), (4; 5; 6))'. Array values can be set using this property.

© 1997-2021 Devart. All Rights Reserved.

5.15.1.1.2.7 Cached Property

Indicates whether to cache array data on the client side.

**Class**

**TCustomIBCArr**

**Syntax**

```property cached: boolean;```

**Remarks**

The Cached property is used to define whether to cache array data on the client side.

© 1997-2021 Devart. All Rights Reserved.

5.15.1.1.2.8 CachedDimensions Property

Contains cached array dimensions count.

**Class**

**TCustomIBCArr**

**Syntax**

```property CachedDimensions: integer;```
Remarks

The CachedDimensions property is used to hold the cached array dimensions count.

Class

TCustomIBCArray

Syntax

property CachedHighBound[Dimension: integer]: integer;

Parameters

Dimension

Holds the dimension subscript.

Remarks

Use the CachedHighBound property to get the upper boundary of defined dimension subscript of cached array elements.

See Also

- ArrayHighBound
- CachedLowBound

Remarks

Used to get the lower boundary of the defined dimension subscript of cached array elements.

Class

TCustomIBCArray

Syntax
**property** CachedLowBound[Dimension: integer]: integer;

**Parameters**

*Dimension*

Holds the dimension subscript.

**Remarks**

Use `CachedLowBound` property to get the lower boundary of the defined dimension subscript of cached array elements.

**See Also**

- ArrayLowBound
- CachedHighBound

**CachedSize Property**

Used to get the cached array data size in bytes.

**Class**

TCustomIBCArray

**Syntax**

**property** CachedSize: integer;

**Remarks**

Use the `CachedSize` property to get the cached array data size in bytes.

**ColumnName Property**

Used to get or set the name of the table column that has array type.

**Class**

TCustomIBCArray
**Syntax**

```
property ColumnName: string;
```

**Remarks**

Use the ColumnName property to get or set the name of the table column that has array type.

**See Also**

- `GetArrayInfo`
- `TableName`

---

5.15.1.2.13 DbHandle Property

Contains the handle of a database where the array is stored.

**Class**

`TCustomIBCArray`

**Syntax**

```
property DbHandle: TISC_DB_HANDLE;
```

**Remarks**

The DbHandle property is used to hold the handle of a database where the array is stored.

**See Also**

- `TIBCConnection.Handle`

---

5.15.1.2.14 IsNull Property

Used to define whether the array field in the database is null.

**Class**

`TCustomIBCArray`
Syntax

```pascal
property IsNull: boolean;
```

Remarks

Use the IsNull property to define whether the array field in the database is null.

5.15.1.1.2.15  Items Property

Used to get or set array items.

Class

`TCustomIBCArray`

Syntax

```pascal
property Items: variant;
```

Remarks

Returns `varArray`, containing array items. Use the `Items` property to get or set array items.

5.15.1.1.2.16  ItemScale Property

Used to get or set the scale for array items for the NUMERIC and DECIMAL datatypes.

Class

`TCustomIBCArray`

Syntax

```pascal
property ItemScale: integer;
```

Remarks

Use the `ItemScale` property to get or set the scale for array items for the NUMERIC and DECIMAL datatypes.
5.15.1.2.17  ItemSize Property

Contains the size of an array item.

Class

TCustomIBCArray

Syntax

```plaintext
property ItemSize: integer;
```

Remarks

The ItemSize property is used to hold the size of an array item in bytes.

5.15.1.2.18  Modified Property

Used to indicate if the modifications done in cache were saved to the database.

Class

TCustomIBCArray

Syntax

```plaintext
property Modified: boolean;
```

Remarks

The Modified property is True when the array was modified in cache and these changes were not saved to the database.
5.15.1.1.2.19 TableName Property

Used to set the name of the table containing an array field.

Class

TCustomIBCArray

Syntax

property TableName: string;

Remarks

Use the TableName property to get or set the name of the table containing an array field.

See Also

- GetArrayInfo
- ColumnName

5.15.1.1.2.20 TrHandle Property

Contains the handle of the transaction in which the array is read or written.

Class

TCustomIBCArray

Syntax

property TrHandle: TISC_TR_HANDLE;

Remarks

The TrHandle property is used to hold the handle of the transaction in which the array is read or written.

See Also

- TIBCTransaction.Handle
5.15.1.3 Methods

Methods of the TCustomIBCArray class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Copies the Source object content to the current one.</td>
</tr>
<tr>
<td>ClearArray</td>
<td>Clears all array values on the server if Cached is set to False.</td>
</tr>
<tr>
<td>CreateTemporaryArray</td>
<td>Creates a temporary array on the InterBase server.</td>
</tr>
<tr>
<td>GetArrayInfo</td>
<td>Used to get array descriptor.</td>
</tr>
<tr>
<td>GetItemAsDateTime</td>
<td>Reads the value of an array item into an object or variable of the TDateTime type.</td>
</tr>
<tr>
<td>GetItemAsFloat</td>
<td>Reads the value of an array item into a floating-point number.</td>
</tr>
<tr>
<td>GetItemAsInteger</td>
<td>Reads the value of an array item into an integer.</td>
</tr>
<tr>
<td>GetItemAsSmallInt</td>
<td>Reads the value of an array item into a short integer.</td>
</tr>
<tr>
<td>GetItemAsString</td>
<td>Reads the value of an array item into a string.</td>
</tr>
<tr>
<td>GetItemAsWideString</td>
<td>Reads the value of an array item into a WideString.</td>
</tr>
<tr>
<td>GetItemsSlice</td>
<td>Returns the array slice items' values.</td>
</tr>
<tr>
<td>GetItemValue</td>
<td>Returns the array item value.</td>
</tr>
<tr>
<td>ReadArray</td>
<td>Reads an array from the database to memory.</td>
</tr>
<tr>
<td>ReadArrayItem</td>
<td>Reads the array item specified by indices from the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ReadArraySlice</td>
<td>Reads array slice from the database to memory.</td>
</tr>
<tr>
<td>Release</td>
<td>Increments the reference count.</td>
</tr>
<tr>
<td>SetItemAsDateTime</td>
<td>Assigns the TDateTime value to the contents of an array item.</td>
</tr>
<tr>
<td>SetItemAsFloat</td>
<td>Assigns floating-point value to the contents of an array item.</td>
</tr>
<tr>
<td>SetItemAsInteger</td>
<td>Assigns integer value to the contents of an array item.</td>
</tr>
<tr>
<td>SetItemAsSmallInt</td>
<td>Assigns short integer value to the contents of an array item.</td>
</tr>
<tr>
<td>SetItemAsString</td>
<td>Assigns string value to the contents of an array item.</td>
</tr>
<tr>
<td>SetItemAsWideString</td>
<td>Assigns WideString value to the contents of an array item.</td>
</tr>
<tr>
<td>SetItemsSlice</td>
<td>Sets the array slice values.</td>
</tr>
<tr>
<td>SetItemValue</td>
<td>Sets the array item value.</td>
</tr>
<tr>
<td>WriteArray</td>
<td>Writes all cached array values to the database.</td>
</tr>
<tr>
<td>WriteArraySlice</td>
<td>Writes cached array slice.</td>
</tr>
</tbody>
</table>

See Also
- TCustomIBCArray Class
- TCustomIBCArray Class Members

5.15.1.3.1 Assign Method

Copies the Source object content to the current one.

Class
TCustomIBCArray

Syntax
procedure Assign(Source: TCustomIBCArray);

Parameters

Source
Holds the Source object content.

Remarks

Use the Assign method to copy the Source object content to the current one.

Class

TCustomIBCArray

Syntax

procedure ClearArray;

Remarks

Call the ClearArray method to clear all array values on the server if Cached is set to False. If Cached property is set to True, this method clears all cached array values.

Class

TCustomIBCArray

Syntax

procedure CreateTemporaryArray;

Remarks

Creates a temporary array on the InterBase server.
Call the CreateTemporaryArray method to create a temporary array on the InterBase server. To use this method, the ArrayDimensions, ItemType, ItemSize, ItemScale properties must be set.

See Also
- GetArrayInfo

Syntax

```pascal
procedure GetArrayInfo;
```

Remarks

Call the GetArrayInfo method to get array descriptor, which contains information about array dimensions, high and low subscript boundaries, array item datatype, size and scale. This method is useful when using TIBCAArray as IN parameter. It can be also useful for creating temporary array with the same array dimensions and item type as array field in a table. For doing this set TableName and ColumnName properties for TIBCAArray object, call GetArrayInfo method and then call CreateTemporaryArray method.

See Also
- CreateTemporaryArray
- TableName
- ColumnName
5.15.1.3.5 GetItemAsDateTime Method

Reads the value of an array item into an object or variable of the TDateTime type.

Class
TCustomIBCArray

Syntax
function GetItemAsDateTime(Indices: array of integer): TDateTime;

Parameters
Indices
Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

Return Value
the value of an array item as DateTime.

Remarks
Call the GetItemAsDateTime method to read the value of an array item into an object or variable of the TDateTime type.

See Also
- GetItemValue
- TIBCArray.ItemType

© 1997-2021 Devart. All Rights Reserved.

5.15.1.3.6 GetItemAsFloat Method

Reads the value of an array item into a floating-point number.

Class
TCustomIBCArray

Syntax
function GetItemAsFloat(Indices: array of integer): double;

Parameters
Indices

© 2021 Devart
Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

**Return Value**
the value of an array item as a floating-point number.

**Remarks**
Call the GetItemAsFloat method to read the value of an array item into a floating-point number.

**See Also**
- `GetItemValue`
- `TIBCAArray.ItemType`

© 1997-2021 Devart. All Rights Reserved.

5.15.1.3.7 GetItemAsInteger Method

Reads the value of an array item into an integer.

**Class**
`TCustomIBCAArray`

**Syntax**
```pascal
function GetItemAsInteger(Indices: array of integer): integer;
```

**Parameters**
- `Indices`
  Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

**Return Value**
the value of an array item as integer.

**Remarks**
Call the GetItemAsInteger method to read the value of an array item into an integer.

**See Also**
- `GetItemValue`
- `TIBCAArray.ItemType`
5.15.1.1.3.8 GetItemAsSmallInt Method

Reads the value of an array item into a short integer.

Class

TCustomIBCArray

Syntax

```function GetItemAsSmallInt(Indices: array of integer): SmallInt;```

Parameters

Indices

- Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

Return Value

- the value of an array item as short integer.

Remarks

Call the GetItemAsSmallInt method to read the value of an array item into a short integer.

See Also

- GetItemValue
- TIBCArray.ItemType

5.15.1.1.3.9 GetItemAsString Method

Reads the value of an array item into a string.

Class

TCustomIBCArray

Syntax

```function GetItemAsString(Indices: array of integer): string;```

Parameters

Indices

- the value of an array item as string.
holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

**Return Value**
the value of an array item as string.

Remarks
Call the GetItemAsString method to read the value of an array item into a string.

See Also
- [GetItemValue](#)
- [TIBCArray.ItemType](#)

© 1997-2021
Devart. All Rights Reserved.

### 5.15.1.1.3.10 GetItemAsString Method

Reads the value of an array item into a WideString.

**Class**

[TCustomIBCArray](#)

**Syntax**

```
function GetItemAsString(Indices: array of integer): string;
```

**Parameters**

**Indices**
Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

**Return Value**
the value of an array item as WideString.

**Remarks**
Call the GetItemAsString method to read the value of an array item into a WideString.

See Also
- [GetItemValue](#)
- [TIBCArray.ItemType](#)

© 1997-2021
Devart. All Rights Reserved.
5.15.1.1.3.11 GetItemsSlice Method

Returns the array slice items’ values.

Class
TCustomIBCArrary

Syntax

function GetItemsSlice(Bounds: array of integer): variant;

Parameters

Bounds
Holds the upper and lower boundaries of the slice.

Return Value
the array slice items’ values.

Remarks
Call the GetItemsSlice method to get the array slice items’ values.

5.15.1.1.3.12 GetItemValue Method

Returns the array item value.

Class
TCustomIBCArrary

Syntax

function GetItemValue(Indices: array of integer): variant;

Parameters

Indices
Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

Return Value
the value of an array item.
Remarks
Call the GetItemValue method to get the array item value.

See Also
- GetItemAsDateTime
- GetItemAsFloat
- GetItemAsInteger
- GetItemAsSmallInt
- GetItemAsString
- GetItemAsWideString

© 1997-2021
Devart. All Rights Reserved.

5.15.1.1.3.13 ReadArray Method

Reads an array from the database to memory.

Class
TCustomIBCArray

Syntax

procedure ReadArray;

Remarks
Call the ReadArray method to read an array from database to memory.

See Also
- WriteArray
- ReadArraySlice
- ReadArrayItem

© 1997-2021
Devart. All Rights Reserved.
5.15.1.3.14  ReadArrayItem Method

Reads the array item specified by indices from the database to memory.

Class

TCustomIBCArray

Syntax

procedure ReadArrayItem(Indices: array of integer);

Parameters

Indices

Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

Remarks

Call the ReadArrayItem method to read array item specified by indices from the database to the memory.

See Also

• ReadArray
• ReadArraySlice

© 1997-2021 Devart. All Rights Reserved.

5.15.1.3.15  ReadArraySlice Method

Reads array slice from the database to memory.

Class

TCustomIBCArray

Syntax

procedure ReadArraySlice(Bounds: array of integer);

Parameters

Bounds

Holds the upper and lower boundaries of the array slice.
Remarks

Call the ReadArraySlice method to read array slice from database to memory.

See Also

- WriteArraySlice
- ReadArray
- ReadArrayItem

© 1997-2021
Devart. All Rights Reserved.

5.15.1.1.3.16 SetItemAsDateTime Method

Assigns the TDateTime value to the contents of an array item.

Class

TCustomIBCArray

Syntax

```pascal
procedure SetItemAsDateTime(Indices: array of integer; Value: TDateTime);
```

Parameters

- **Indices**
  - Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).
- **Value**
  - Holds the array item value as TDateTime.

Remarks

Call the SetItemAsDateTime method to assign the TDateTime value to the contents of an array item.

See Also

- SetItemValue
- TIBCArray.ItemType

© 1997-2021
Devart. All Rights Reserved.
5.15.1.1.3.17 SetItemAsFloat Method

Assigns floating-point value to the contents of an array item.

Class

TCustomIBCArray

Syntax

procedure SetItemAsFloat(Indices: array of integer; Value: double);

Parameters

Indices
Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

Value
Holds the array item value as floating-point.

Remarks

Call the SetItemAsFloat method to assign floating-point value to the contents of an array item.

See Also

• SetItemValue
• TIBCArray.ItemType

© 1997-2021 Devart. All Rights Reserved.

5.15.1.1.3.18 SetItemAsInteger Method

Assigns integer value to the contents of an array item.

Class

TCustomIBCArray

Syntax

procedure SetItemAsInteger(Indices: array of integer; Value: integer);

Parameters
**Indices**

Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

**Value**

Holds the array item value as integer.

**Remarks**

Call the `SetItemAsInteger` method to assign integer value to the contents of an array item.

**See Also**

- `SetItemValue`
- `TIBCArray.ItemType`

---

### 5.15.1.1.3.19 SetItemAsSmallInt Method

Assigns short integer value to the contents of an array item.

**Class**

`TCustomIBCArray`

**Syntax**

```pascal
procedure SetItemAsSmallInt(Indices: array of integer; Value: SmallInt);
```

**Parameters**

**Indices**

Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

**Value**

Holds the array item value as short integer.

**Remarks**

Call the `SetItemAsSmallInt` method to assign short integer value to the contents of an array item.

**See Also**

- `SetItemValue`
5.15.1.1.3.20  SetItemAsString Method

Assigns string value to the contents of an array item.

Class

TCustomIBCArray

Syntax

```
procedure SetItemAsString(Indices: array of integer; Value: string);
```

Parameters

- **Indices**
  - Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).
- **Value**
  - Holds the array item value as string.

Remarks

Call the `SetItemAsString` method to assign string value to the contents of an array item.

See Also

- `SetItemValue`
- `TIBCArray.ItemType`

5.15.1.1.3.21  SetItemAsWideString Method

Assigns WideString value to the contents of an array item.

Class

TCustomIBCArray

Syntax
**procedure** SetItemAsWideString(Indices: array of integer; Value: string);

**Parameters**

*Indices*
- Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

*Value*
- Holds the array item value as WideString.

**Remarks**

Call the SetItemAsWideString method to assign WideString value to the contents of an array item.

**See Also**

- SetItemValue
- TIBCArry.ItemType

Sets the array slice values.

**Class**

TCustomIBCArray

**Syntax**

**procedure** SetItemsSlice(const Values: variant);

**Parameters**

*Values*
- Holds the array slice values.

**Remarks**

Call the SetItemsSlice method to set array slice values.
5.15.1.3.23 SetItemValue Method

Sets the array item value.

Class
TCustomIBCArray

Syntax

```delphi
procedure SetItemValue(Indices: array of integer; Value: variant);
```

Parameters

- **Indices**
  - Holds an array of element indexes in the array (if the array is one-dimensional, there is only one index).

- **Value**
  - Holds the array item value.

Remarks

Call the SetItemValue method to set the array item value.

See Also

- SetItemAsDateTime
- SetItemAsFloat
- SetItemAsInteger
- SetItemAsSmallInt
- SetItemAsString
- SetItemAsWideString

© 1997-2021 Devart. All Rights Reserved.

5.15.1.3.24 WriteArray Method

Writes all cached array values to the database.

Class

TCustomIBCArray

Syntax

```delphi
```
procedure WriteArray;

Remarks

Call the WriteArray method to write all cached array values to the database. This method does nothing if the Cached property is set to False.

See Also

- WriteArraySlice
- ReadArray

5.15.1.1.3.25 WriteArraySlice Method

Writes cached array slice.

Class

TCustomIBCArray

Syntax

procedure writeArraySlice(Bounds: array of integer);

Parameters

Bounds
  Holds the upper and lower boundaries of the array slice.

Remarks

Call the WriteArraySlice method to write cached array slice. This method does nothing if Cached property is set to False.

See Also

- WriteArray
- ReadArraySlice
5.16 IBCClasses

IBCClasses unit defines the following data type constants: dtDbKey dtFixedChar dtFixedWideChar

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGDSDatabaseInfo</td>
<td>A class providing information about an InterBase database.</td>
</tr>
<tr>
<td>TIBCBlob</td>
<td>A class holding value of the BLOB fields and parameters.</td>
</tr>
</tbody>
</table>

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCIsolationLevel</td>
<td>Specifies the transaction isolation level and access mode.</td>
</tr>
</tbody>
</table>

Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reverse2</td>
<td>Switches places of bytes in the word argument.</td>
</tr>
<tr>
<td>Reverse4</td>
<td>Switches places of bytes in the cardinal argument.</td>
</tr>
<tr>
<td>XSQLDA_LENGTH</td>
<td>Analogue of InterBase XSQLDA_LENGTH macro. Calculates the number of bytes that must be allocated for an input or output XSQLDA.</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntegerPrecision</td>
<td>Set this constant to define the type of NUMERIC and DECIMAL fields with</td>
</tr>
</tbody>
</table>
5.16.1 Classes

Classes in the IBCClasses unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGDSDatabaseInfo</td>
<td>A class providing information about an InterBase database.</td>
</tr>
<tr>
<td>TIBCBlob</td>
<td>A class holding value of the BLOB fields and parameters.</td>
</tr>
</tbody>
</table>

5.16.1.1 TGDSDatabaseInfo Class

A class providing information about an InterBase database.

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

Unit

IBCClasses

Syntax

TGDSDatabaseInfo = class(System.TObject);

Remarks

Use the TGDSDatabaseInfo class to get information about InterBase database.

See Also
### TIBConnection.DatabaseInfo

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.16.1.1.1 Members

**TGDSDatabaseInfo** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation</td>
<td>Indicates the number of database pages allocated.</td>
</tr>
<tr>
<td>AttachmentID</td>
<td>Indicates a unique connection identifier.</td>
</tr>
<tr>
<td>BackoutCount</td>
<td>Contains the number of removals of a record version.</td>
</tr>
<tr>
<td>BaseLevel</td>
<td>Indicates the database version number.</td>
</tr>
<tr>
<td>CurLogFileName</td>
<td>Contains the name of the log-file.</td>
</tr>
<tr>
<td>CurLogPartitionOffset</td>
<td>Indicates the value of isc_info_cur_log_part_offset.</td>
</tr>
<tr>
<td>CurrentMemory</td>
<td>Indicates the amount of the server memory currently in use.</td>
</tr>
<tr>
<td>DBFileName</td>
<td>Contains the database filename.</td>
</tr>
<tr>
<td>DBImplementationClass</td>
<td>Indicates the database implementation class number.</td>
</tr>
<tr>
<td>DBImplementationNo</td>
<td>Indicates the database implementation number.</td>
</tr>
<tr>
<td>DBSiteName</td>
<td>Contains the database site name.</td>
</tr>
<tr>
<td>DeleteCount</td>
<td>Holds the number of database deletes since the database was last attached.</td>
</tr>
<tr>
<td>ExpungeCount</td>
<td>Indicates the number of removals of a record and all of its ancestors.</td>
</tr>
<tr>
<td><strong>Fetches</strong></td>
<td>Indicates the number of reads from the memory buffer cache.</td>
</tr>
<tr>
<td><strong>ForcedWrites</strong></td>
<td>Used to indicate the mode in which database writes are performed.</td>
</tr>
<tr>
<td><strong>InsertCount</strong></td>
<td>Holds the number of insertions into the database since the database was last attached.</td>
</tr>
<tr>
<td><strong>IsRemoteConnect</strong></td>
<td>Used to indicate whether the connection with the database is remote.</td>
</tr>
<tr>
<td><strong>LogFile</strong></td>
<td>Used to indicate the value of isc_info_log_file.</td>
</tr>
<tr>
<td><strong>Marks</strong></td>
<td>Used to indicate the number of writes to the memory buffer cache.</td>
</tr>
<tr>
<td><strong>MaxMemory</strong></td>
<td>Indicates the maximum amount of memory used at one time since the first process attached to the database.</td>
</tr>
<tr>
<td><strong>NoReserve</strong></td>
<td>Specifies if the space for holding backup versions of modified records is reserved on each database page.</td>
</tr>
<tr>
<td><strong>NumBuffers</strong></td>
<td>Indicates the number of currently allocated memory buffers.</td>
</tr>
<tr>
<td><strong>NumWALBuffers</strong></td>
<td>Indicates the value of isc_info_num_walBuffers.</td>
</tr>
<tr>
<td><strong>ODSMajorVersion</strong></td>
<td>Indicates the on disk structure (ODS) major version number.</td>
</tr>
<tr>
<td><strong>ODSMinorVersion</strong></td>
<td>Indicates the on disk structure (ODS) minor version number.</td>
</tr>
<tr>
<td><strong>PageSize</strong></td>
<td>Shows the number of bytes per page for the database.</td>
</tr>
<tr>
<td><strong>PurgeCount</strong></td>
<td>Holds the number of removals of records committed from the</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ReadIdxCount</td>
<td>Holds the number of reads done via an index since the database was last attached.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Indicates whether the database is read only.</td>
</tr>
<tr>
<td>Reads</td>
<td>Indicates the number of page reads from the database since the current database was first attached.</td>
</tr>
<tr>
<td>ReadSeqCount</td>
<td>Contains the number of sequential database reads done on each table since the database was last attached.</td>
</tr>
<tr>
<td>SweepInterval</td>
<td>Indicates the number of transactions that are committed between &quot;sweeps&quot;.</td>
</tr>
<tr>
<td>UpdateCount</td>
<td>Contains the number of database updates since the database was last attached.</td>
</tr>
<tr>
<td>UserNames</td>
<td>Contains the names of all users currently attached to the database.</td>
</tr>
<tr>
<td>Version</td>
<td>Indicates the version of the database implementation.</td>
</tr>
<tr>
<td>WALAverageGroupCommitSize</td>
<td>Indicates the value of isc_info_wal_avg_grpc_size.</td>
</tr>
<tr>
<td>WALAverageIOSize</td>
<td>Indicates the value of isc_info_wal_avg_io_size.</td>
</tr>
<tr>
<td>WALBufferSize</td>
<td>Indicates the value of isc_info_wal_buffer_size.</td>
</tr>
<tr>
<td>WALCheckpointLength</td>
<td>Indicates the value of isc_info_wal_ckpt_length.</td>
</tr>
<tr>
<td>WALCurCheckpointInterval</td>
<td>Indicates the value of isc_info_wal_cur_ckpt_interval.</td>
</tr>
<tr>
<td>WALGroupCommitWaitUSecs</td>
<td>Indicates the value of isc_info_wal_grpc_wait_usecs.</td>
</tr>
<tr>
<td>WALNumCommits</td>
<td>Indicates the value of isc_info_wal_num_commits.</td>
</tr>
</tbody>
</table>
### 5.16.1.1.2 Properties

Properties of the **TGDSDatabaseInfo** class.

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

<table>
<thead>
<tr>
<th>Public</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Allocation</strong></td>
<td>Indicates the number of database pages allocated.</td>
</tr>
<tr>
<td><strong>AttachmentID</strong></td>
<td>Indicates a unique connection identifier.</td>
</tr>
<tr>
<td><strong>BackoutCount</strong></td>
<td>Contains the number of removals of a record version.</td>
</tr>
<tr>
<td><strong>BaseLevel</strong></td>
<td>Indicates the database version number.</td>
</tr>
<tr>
<td><strong>CurLogFileName</strong></td>
<td>Contains the name of the log-file.</td>
</tr>
<tr>
<td><strong>CurLogPartitionOffset</strong></td>
<td>Indicates the value of <strong>isc_info_cur_log_part_offset</strong>.</td>
</tr>
<tr>
<td><strong>CurrentMemory</strong></td>
<td>Indicates the amount of server memory currently in use.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DBFileName</td>
<td>Contains the database filename.</td>
</tr>
<tr>
<td>DBImplementationClass</td>
<td>Indicates the database implementation class number.</td>
</tr>
<tr>
<td>DBImplementationNo</td>
<td>Indicates the database implementation number.</td>
</tr>
<tr>
<td>DBSiteName</td>
<td>Contains the database site name.</td>
</tr>
<tr>
<td>DeleteCount</td>
<td>Holds the number of database deletes since the database was last attached.</td>
</tr>
<tr>
<td>ExpungeCount</td>
<td>Indicates the number of removals of a record and all of its ancestors.</td>
</tr>
<tr>
<td>Fetches</td>
<td>Indicates the number of reads from the memory buffer cache.</td>
</tr>
<tr>
<td>ForcedWrites</td>
<td>Used to indicate the mode in which database writes are performed.</td>
</tr>
<tr>
<td>InsertCount</td>
<td>Holds the number of insertions into the database since the database was last attached.</td>
</tr>
<tr>
<td>IsRemoteConnect</td>
<td>Used to indicate whether the connection with the database is remote.</td>
</tr>
<tr>
<td>LogFile</td>
<td>Used to indicate the value of isc_info_log_file.</td>
</tr>
<tr>
<td>Marks</td>
<td>Used to indicate the number of writes to the memory buffer cache.</td>
</tr>
<tr>
<td>MaxMemory</td>
<td>Indicates the maximum amount of memory used at one time since the first process attached to the database.</td>
</tr>
<tr>
<td>NoReserve</td>
<td>Specifies if the space for holding backup versions of modified records is reserved on each database page.</td>
</tr>
<tr>
<td>NumBuffers</td>
<td>Indicates the number of</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NumWALBuffers</td>
<td>Indicates the value of isc_info_num_wal_buffers.</td>
</tr>
<tr>
<td>ODSMajorVersion</td>
<td>Indicates the on disk structure (ODS) major version number.</td>
</tr>
<tr>
<td>ODSMinorVersion</td>
<td>Indicates the on disk structure (ODS) minor version number.</td>
</tr>
<tr>
<td>PageSize</td>
<td>Shows the number of bytes per page for the database.</td>
</tr>
<tr>
<td>PurgeCount</td>
<td>Holds the number of removals of records committed from the database, resulting in older versions.</td>
</tr>
<tr>
<td>ReadIdxCount</td>
<td>Holds the number of reads done via an index since the database was last attached.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Indicates whether the database is read only.</td>
</tr>
<tr>
<td>Reads</td>
<td>Indicates the number of page reads from the database since the current database was first attached.</td>
</tr>
<tr>
<td>ReadSeqCount</td>
<td>Contains the number of sequential database reads done on each table since the database was last attached.</td>
</tr>
<tr>
<td>SweepInterval</td>
<td>Indicates the number of transactions that are committed between &quot;sweeps&quot;.</td>
</tr>
<tr>
<td>UpdateCount</td>
<td>Contains the number of database updates since the database was last attached.</td>
</tr>
<tr>
<td>UserNames</td>
<td>Contains the names of all users currently attached to the database.</td>
</tr>
<tr>
<td>Version</td>
<td>Indicates the version of the database implementation.</td>
</tr>
<tr>
<td>WALAverageGroupCommitSize</td>
<td>Indicates the value of isc_info_wal_avg_grpc_size.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WALAverageIOSize</td>
<td>Indicates the value of isc_info_wal_avg_io_size.</td>
</tr>
<tr>
<td>WALSErrorSize</td>
<td>Indicates the value of isc_info_wal_buffer_size.</td>
</tr>
<tr>
<td>WALCheckpointLength</td>
<td>Indicates the value of isc_info_wal_ckpt_length.</td>
</tr>
<tr>
<td>WALCurCheckpointInterval</td>
<td>Indicates the value of isc_info_wal_cur_ckpt_interval.</td>
</tr>
<tr>
<td>WALGroupCommitWaitUSecs</td>
<td>Indicates the value of isc_info_wal_grpc_wait_usecs.</td>
</tr>
<tr>
<td>WALNumCommits</td>
<td>Indicates the value of isc_info_wal_num_commits.</td>
</tr>
<tr>
<td>WALNumIO</td>
<td>Indicates the value of isc_info_wal_num_id.</td>
</tr>
<tr>
<td>WALPrvCheckpointFilename</td>
<td>Indicates the value of isc_info_wal_prv_ckpt_filename.</td>
</tr>
<tr>
<td>WALPrvCheckpointPartOffset</td>
<td>Indicates the value of isc_info_wal_prv_ckpt_part_offset.</td>
</tr>
<tr>
<td>Writes</td>
<td>Indicates the number of page writes to the current database since it was first attached by any process.</td>
</tr>
</tbody>
</table>

See Also
- TGDSDatabaseInfo Class
- TGDSDatabaseInfo Class Members

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

5.16.1.1.2.1 Allocation Property

Indicates the number of database pages allocated.

Class
TGDSDatabaseInfo
### Syntax

```property
Allocation: integer;
```

### Remarks

Use the Allocation property to determine the number of database pages allocated.

### 5.16.1.1.2.2 AttachmentID Property

Indicates a unique connection identifier.

### Class

`TGDSDatabaseInfo`

### Syntax

```property
AttachmentID: integer;
```

### Remarks

Use the AttachmentID property to indicate a unique connection identifier.

### 5.16.1.1.2.3 BackoutCount Property

Contains the number of removals of a record version.

### Class

`TGDSDatabaseInfo`

### Syntax

```property
BackoutCount: TStringList;
```

### Remarks

Use the BackoutCount property to indicate the number of removals of a record version.
5.16.1.1.4 BaseLevel Property

Indicates the database version number.

Class
TGDSDatabaseInfo

Syntax

property BaseLevel: integer;

Remarks
Use the BaseLevel property to determine the database version number.

5.16.1.1.5 CurLogFileName Property

Contains the name of the log-file.

Class
TGDSDatabaseInfo

Syntax

property CurLogFileName: string;

Remarks
Contains the name of the log-file. The TIBConnection component must be active, otherwise an exception is raised.
5.16.1.1.2.6 CurLogPartitionOffset Property

Indicates the value of isc_info_cur_log_part_offset.

Class
TGDSDatabaseInfo

Syntax

property CurLogPartitionOffset: integer;

Remarks
Use the CurLogPartitionOffset property to determine the value of isc_info_cur_log_part_offset.

5.16.1.1.2.7 CurrentMemory Property

Indicates the amount of the server memory currently in use.

Class
TGDSDatabaseInfo

Syntax

property CurrentMemory: integer;

Remarks
Use the CurrentMemory property to determine the amount of server memory (in bytes) currently in use.

5.16.1.1.2.8 DBFileName Property

Contains the database filename.

Class
**TGDSDatabaseInfo**

**Syntax**

```property```

string DBFileName;
```

**Remarks**

Contains the database filename.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.1.2.9 DBImplementationClass Property

Indicates the database implementation class number.

**Class**

**TGDSDatabaseInfo**

**Syntax**

```property```

integer DBImplementationClass;
```

**Remarks**

Use the DBImplementationClass property to determine the database implementation class number.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.1.2.10 DBImplementationNo Property

Indicates the database implementation number.

**Class**

**TGDSDatabaseInfo**

**Syntax**

```property```

integer DBImplementationNo;
```

**Remarks**
Use the DBImplementationNo to determine the database implementation number.

5.16.1.1.2.11 DBSiteName Property

Contains the database site name.

Class

**TGDSDatabaseInfo**

Syntax

```property DBSiteName: string;
```

Remarks

Contains the database site name.

5.16.1.1.2.12 DeleteCount Property

Holds the number of database deletes since the database was last attached.

Class

**TGDSDatabaseInfo**

Syntax

```property DeleteCount: TStringList;
```

Remarks

Contains the number of database deletes since the database was last attached.
5.16.1.1.2.13 ExpungeCount Property

Indicates the number of removals of a record and all of its ancestors.

Class
TGDSDatabaseInfo

Syntax

```pascal
property ExpungeCount: TStringList;
```

Remarks
Use the ExpungeCount property to determine the number of removals of a record and all of its ancestors.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.1.2.14 Fetches Property

Indicates the number of reads from the memory buffer cache.

Class
TGDSDatabaseInfo

Syntax

```pascal
property Fetches: integer;
```

Remarks
Use the Fetches property to determine the number of reads from the memory buffer cache.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.1.2.15 ForcedWrites Property

Used to indicate the mode in which database writes are performed.

Class
TGDSDatabaseInfo

© 2021 Devart
Syntax

```pascal
property ForcedWrites: integer;
```

Remarks

Use the ForcedWrites property to indicate the mode in which database writes are performed. It is 0 for asynchronous mode, 1 for synchronous mode.

Class

`TGDSDatabaseInfo`

Syntax

```pascal
property InsertCount: TStringList;
```

Remarks

Contains the number of insertions into the database since the database was last attached.

Class

`TGDSDatabaseInfo`

Syntax

```pascal
property IsRemoteConnect: boolean;
```

Remarks

Use the IsRemoteConnect property to determine whether the connection with the database is remote.
5.16.1.2.18 LogFile Property

Used to indicate the value of isc_info_log_file.

Class

TGDSDatabaseInfo

Syntax

```property` LogFile: integer;`  
```

Remarks

Use the LogFile property to indicate the value of isc_info_log_file.

5.16.1.2.19 Marks Property

Used to indicate the number of writes to the memory buffer cache.

Class

TGDSDatabaseInfo

Syntax

```property` Marks: integer;`  
```

Remarks

Use the Marks property to determine the number of writes to the memory buffer cache.
5.16.1.2.20 MaxMemory Property

Indicates the maximum amount of memory used at one time since the first process attached to the database.

Class
TGDSDatabaseInfo

Syntax

```pascal
property MaxMemory: integer;
```

Remarks
Use the MaxMemory property to indicate the maximum amount of memory used at one time since the first process attached to the database.

5.16.1.2.21 NoReserve Property

Specifies if the space for holding backup versions of modified records is reserved on each database page.

Class
TGDSDatabaseInfo

Syntax

```pascal
property NoReserve: integer;
```

Remarks
If 0, the space is reserved on each database page for holding backup versions of modified records.

If 1, no space is reserved for such records.
5.16.1.1.2.22 NumBuffers Property

Indicates the number of currently allocated memory buffers.

Class

TGDSDatabaseInfo

Syntax

property NumBuffers: integer;

Remarks

Use the NumBuffers property to indicate the number of currently allocated memory buffers.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.1.2.23 NumWALBuffers Property

Indicates the value of isc_info_num_wal_buffers.

Class

TGDSDatabaseInfo

Syntax

property NumWALBuffers: integer;

Remarks

Use the NumWALBuffers property to indicate the value of isc_info_num_wal_buffers.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.1.2.24 ODSMajorVersion Property

Indicates the on disk structure (ODS) major version number.

Class

TGDSDatabaseInfo
Syntax

```
property ODSMajorVersion: integer;
```

Remarks

Use the ODSMajorVersion to determine the on disk structure (ODS) major version number.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.1.2.25 ODSMinorVersion Property

Indicates the on disk structure (ODS) minor version number.

Class

`TGDSDatabaseInfo`

Syntax

```
property ODSMinorVersion: integer;
```

Remarks

Use the ODSMinorVersion property to determine the on disk structure (ODS) minor version number.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.1.2.26 PageSize Property

Shows the number of bytes per page for the database.

Class

`TGDSDatabaseInfo`

Syntax

```
property PageSize: integer;
```

Remarks

Shows the number of bytes per page for the database.
5.16.1.2.27 PurgeCount Property

Holds the number of removals of records committed from the database, resulting in older versions.

Class

TGDSDatabaseInfo

Syntax

```property
PurgeCount: TStringList;
```

Remarks

Contains the number of removals of records committed from the database, resulting in older versions.

5.16.1.2.28 ReadIdxCount Property

Holds the number of reads done via an index since the database was last attached.

Class

TGDSDatabaseInfo

Syntax

```property
ReadIdxCount: TStringList;
```

Remarks

The ReadIdxCount property is used to contain the number of reads done via an index since the database was last attached.
5.16.1.1.2.29  ReadOnly Property

Indicates whether the database is read only.

**Class**

*TGDSDatabaseInfo*

**Syntax**

```property
ReadOnly: Boolean;
```

**Remarks**

The `ReadOnly` property is used to indicate whether the database is read only.

© 1997-2021
Devar. All Rights Reserved.

5.16.1.1.2.30  Reads Property

Indicates the number of page reads from the database since the current database was first attached.

**Class**

*TGDSDatabaseInfo*

**Syntax**

```property
Reads: integer;
```

**Remarks**

The `Reads` property is used to indicate the number of page reads from the database since the current database was first attached.

© 1997-2021
Devar. All Rights Reserved.

5.16.1.1.2.31  ReadSeqCount Property

Contains the number of sequential database reads done on each table since the database was last attached.

**Class**
**TGDSDatabaseInfo**

**Syntax**

```
property ReadSeqCount: TStringList;
```

**Remarks**

The ReadSeqCount property is used to contain the number of sequential database reads done on each table since the database was last attached.

---

5.16.1.1.2.32  **SweepInterval Property**

Indicates the number of transactions that are committed between "sweeps".

**Class**

**TGDSDatabaseInfo**

**Syntax**

```
property SweepInterval: integer;
```

**Remarks**

Use the SweepInterval property to indicate the number of transactions that are committed between "sweeps".

---

5.16.1.1.2.33  **UpdateCount Property**

Contains the number of database updates since the database was last attached.

**Class**

**TGDSDatabaseInfo**

**Syntax**

```
property UpdateCount: TStringList;
```
Remarks
The UpdateCount property is used to hold the number of database updates since the
database was last attached.

5.16.1.2.34 UserNames Property

Contains the names of all users currently attached to the database.

Class
TGDSDatabaseInfo

Syntax

```property UserNames: TStringList;```

Remarks
The UserNames property is used to hold the names of all users currently attached to the
database.

5.16.1.2.35 Version Property

Indicates the version of the database implementation.

Class
TGDSDatabaseInfo

Syntax

```property version: string;```

Remarks
The Version property is used to indicate the version of the database implementation.
Reserved.

5.16.1.1.2.36  WALAverageGroupCommitSize Property

Indicates the value of isc_info_wal_avg_grpc_size.

Class

TGDSDatabaseInfo

Syntax

```property
WALAverageGroupCommitSize: integer;
```

Remarks

The WALAverageGroupCommitSize property is used to indicate the value of isc_info_wal_avg_grpc_size.

© 1997-2021
Devart. All Rights Reserved.

5.16.1.1.2.37  WALAverageIOSize Property

Indicates the value of isc_info_wal_avg_io_size.

Class

TGDSDatabaseInfo

Syntax

```property
WALAverageIOSize: integer;
```

Remarks

The WALAverageIOSize property is used to indicate the value of isc_info_wal_avg_io_size.

© 1997-2021
Devart. All Rights Reserved.

5.16.1.1.2.38  WALBufferSize Property

Indicates the value of isc_info_wal_buffer_size.

Class
**TGDSDatabaseInfo**

**Syntax**

```property```
WALBufferSize: integer;
```property```

**Remarks**
The WALBufferSize property is used to indicate the value of isc_info_wal_buffer_size.

---

**5.16.1.1.2.39  WALCheckpointLength Property**

**Remarks**
The WALCheckpointLength is used to indicate the value of isc_info_wal_ckpt_length.

**Class**

**TGDSDatabaseInfo**

**Syntax**

```property```
WALCheckpointLength: integer;
```property```

**Remarks**
The WALCheckpointLength is used to indicate the value of isc_info_wal_ckpt_length.

---

**5.16.1.1.2.40  WALCurCheckpointInterval Property**

**Remarks**
Indicates the value of isc_info_wal_cur_ckpt_interval.

**Class**

**TGDSDatabaseInfo**

**Syntax**

```property```
WALCurCheckpointInterval: integer;
```property```

**Remarks**
The WALCurCheckpointInterval property is used to indicate the value of
isc_info_wal_cur_ckpt_interval.

Class

TGDSDatabaseInfo

Syntax

property WALGroupCommitWaitUSecs: integer;

Remarks

The WALGroupCommitWaitUSec property is used to indicate the value of
isc_info_wal_grpc_wait_usecs.

Indicates the value of isc_info_wal_num_commits.

Class

TGDSDatabaseInfo

Syntax

property WALNumCommits: integer;

Remarks

The WALNumCommits property is used to indicate the value of isc_info_wal_num_commits.
5.16.1.1.2.43 WALNumIO Property

Indicates the value of isc_info_wal_num_id.

Class

TGDSDatabaseInfo

Syntax

```property``` WALNumIO: integer;

Remarks

The WALNumIO property is used to indicate the value of isc_info_wal_num_id.

5.16.1.1.2.44 WALPrvCheckpointFilename Property

Indicates the value of isc_info_wal_prv_ckpt_fname.

Class

TGDSDatabaseInfo

Syntax

```property``` WALPrvCheckpointFilename: string;

Remarks

The WALPrvCheckpointFilename property is used to indicate the value of isc_info_wal_prv_ckpt_fname.

5.16.1.1.2.45 WALPrvCheckpointPartOffset Property

Indicates the value of isc_info_wal_prv_ckpt_poffset.

Class

TGDSDatabaseInfo
## WALPrvCheckpointPartOffset Property

**Syntax**

```plaintext
property WALPrvCheckpointPartOffset: integer;
```

**Remarks**

The `WALPrvCheckpointPartOffset` property is used to indicate the value of `isc_info_wal_prv_ckpt_poffset`.

© 1997-2021 Devart. All Rights Reserved.

### 5.16.1.1.2.46 Writes Property

Indicates the number of page writes to the current database since it was first attached by any process.

**Class**

`TGDSDatabaseInfo`

**Syntax**

```plaintext
property Writes: integer;
```

**Remarks**

The `Writes` property is used to indicate the number of page writes to the current database since it was first attached by any process.

© 1997-2021 Devart. All Rights Reserved.

### 5.16.1.2 TIBCBlob Class

A class holding value of the BLOB fields and parameters.

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

**Unit**

`IBCClasses`

**Syntax**
TIBCBlob = class(TCompressedBlob);

Remarks

TIBCBlob is a descendant of TCompressedBlob class. It holds value of the BLOB fields and parameters.

Inheritance Hierarchy

TSharedObject
  TBlob
    TCompressedBlob
    TIBCBlob

See Also

- TCompressedBlob
- TCustomIBCDataSet.GetBlob

© 1997-2021 Devart. All Rights Reserved.

5.16.1.2.1 Members

**TIBCBlob** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString (inherited from TBlob)</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString (inherited from TBlob)</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>Cached</td>
<td>Indicates whether the BLOB data are cached on the client or they are accessed remotely on the server.</td>
</tr>
<tr>
<td>CharSetID</td>
<td>Source charset for BLOB subtype conversion using BLOB filters.</td>
</tr>
<tr>
<td>Compressed (inherited from TCompressedBlob)</td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td>CompressedSize (inherited from TCompressedBlob)</td>
<td>Used to indicate compressed size of the Blob</td>
</tr>
</tbody>
</table>
### InterBase Data Access Components

#### Data

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConversionCharsetID</td>
<td>Target charset for BLOB subtype conversion using Blob filters.</td>
</tr>
<tr>
<td>ConversionSubType</td>
<td>Target subtype for the BLOB subtype conversion using BLOB filters.</td>
</tr>
<tr>
<td>DbHandle</td>
<td>Contains the handle of the database where the BLOB is stored.</td>
</tr>
<tr>
<td>Handle</td>
<td>Contains the BLOB handle.</td>
</tr>
<tr>
<td>ID</td>
<td>Contains BLOB ID.</td>
</tr>
<tr>
<td>IsUnicode (inherited from <strong>TBlob</strong>)</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td>MaxSegmentSize</td>
<td>Indicates the largest BLOB segment size.</td>
</tr>
<tr>
<td>NumSegments</td>
<td>Indicates the number of the BLOB segments in the database.</td>
</tr>
<tr>
<td>RefCount (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size (inherited from <strong>TBlob</strong>)</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
<tr>
<td>Streamed</td>
<td>Indicates whether the BLOB is stream or segmented.</td>
</tr>
<tr>
<td>SubType</td>
<td>Source subtype for the BLOB subtype conversion using BLOB filters.</td>
</tr>
<tr>
<td>TrHandle</td>
<td>Contains the handle of the transaction in which the BLOB is read or written.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddRef</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AllocBlob</td>
<td>Allocates and initializes the Blob handle.</td>
</tr>
<tr>
<td>Assign</td>
<td>Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td>CloseBlob</td>
<td>Frees the Blob handle.</td>
</tr>
<tr>
<td>FreeBlob</td>
<td>Clears BLOB ID.</td>
</tr>
<tr>
<td>IsInit</td>
<td>Verifies BLOB ID initialization.</td>
</tr>
<tr>
<td>LengthBlob</td>
<td>Determines the number of bytes contained in the Blob object.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Copies the contents of a stream into the TBlob object.</td>
</tr>
<tr>
<td>Read</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlob.</td>
</tr>
<tr>
<td>ReadBlob</td>
<td>Reads BLOB from the database.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrementsthe reference count.</td>
</tr>
<tr>
<td>SaveToFile</td>
<td>Saves the contents of the TBlob object to a file.</td>
</tr>
<tr>
<td>SaveToStream</td>
<td>Copies the contents of a TBlob object to a stream.</td>
</tr>
<tr>
<td>Truncate</td>
<td>Sets new TBlob size and discards all data over it.</td>
</tr>
<tr>
<td>Write</td>
<td>Stores a raw sequence of bytes into a TBlob object.</td>
</tr>
<tr>
<td>WriteBlob</td>
<td>Writes BLOB to the database.</td>
</tr>
</tbody>
</table>

5.16.1.2.2 Properties

Properties of the TIBCBlob class.

For a complete list of the TIBCBlob class members, see the TIBCBlob Members topic.
## Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsString</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td><strong>AsWideString</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td><strong>Cached</strong></td>
<td>Indicates whether the BLOB data are cached on the client or they are accessed remotely on the server.</td>
</tr>
<tr>
<td><strong>CharsetID</strong></td>
<td>Source charset for BLOB subtype conversion using BLOB filters.</td>
</tr>
<tr>
<td><strong>Compressed</strong> (inherited from <strong>TCompressedBlob</strong>)</td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td><strong>CompressedSize</strong> (inherited from <strong>TCompressedBlob</strong>)</td>
<td>Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td><strong>ConversionCharsetID</strong></td>
<td>Target charset for BLOB subtype conversion using Blob filters.</td>
</tr>
<tr>
<td><strong>ConversionSubType</strong></td>
<td>Target subtype for the BLOB subtype conversion using BLOB filters.</td>
</tr>
<tr>
<td><strong>DbHandle</strong></td>
<td>Contains the handle of the database where the BLOB is stored.</td>
</tr>
<tr>
<td><strong>Handle</strong></td>
<td>Contains the BLOB handle.</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>Contains BLOB ID.</td>
</tr>
<tr>
<td><strong>IsUnicode</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td><strong>MaxSegmentSize</strong></td>
<td>Indicates the largest BLOB segment size.</td>
</tr>
<tr>
<td><strong>NumSegments</strong></td>
<td>Indicates the number of the BLOB segments in the database.</td>
</tr>
<tr>
<td><strong>RefCount</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>
### Cached Property

Indicates whether the BLOB data are cached on the client or they are accessed remotely on the server.

**Class**

**TIBCBlob**

**Syntax**

```property```
cached: boolean;
```endproperty```

**Remarks**

The Cached property is used to indicate whether the BLOB data are cached on the client or they are accessed remotely on the server. In most cases you don't need to set the value of this property directly. To enable or disable BLOB caching, use the `TCustomIBCDataset.Options` property.

**See Also**

- `TCustomIBCDataset.Options`
5.16.1.2.2.2  CharsetID Property

Source charset for BLOB subtype conversion using BLOB filters.

Class

TIBCBlob

Syntax

```pascal
property CharsetID: integer;
```

Remarks

Source charset for BLOB subtype conversion using BLOB filters. It corresponds to
isc_bpb_source_interp parameter in BLOB Parameter Buffer (BPB) that is used for filtration.
For more information on BLOB filters refer to InterBase API Guide.

See Also

- ConversionCharsetID
- SubType

5.16.1.2.2.3  ConversionCharsetID Property

Target charset for BLOB subtype conversion using Blob filters.

Class

TIBCBlob

Syntax

```pascal
property ConversionCharsetID: integer;
```

Remarks

Target charset for BLOB subtype conversion using Blob filters. It corresponds to
isc_bpb_target_interp parameter in BLOB Parameter Buffer (BPB) that is used for filtration.
For more information on BLOB filters refer to InterBase API Guide.

Set ConversionCharset property to the charset that you want to get when reading the BLOB
from database.
Set ConversionCharset property to the charset that you want BLOB to store to database in when writing BLOB to database.

See Also

- CharsetID
- ConversionSubType

5.16.1.2.2.4 ConversionSubType Property

Target subtype for the BLOB subtype conversion using BLOB filters.

Class

TIBCBlob

Syntax

```pascal
property ConversionSubType: integer;
```

Remarks

Target subtype for the BLOB subtype conversion using BLOB filters. It corresponds to the `isc_bpb_target_type` parameter in BLOB Parameter Buffer (BPB) that is used for filtration. For more information about Blob subtypes and filters refer to InterBase API Guide.

Set the ConversionSubType property to the Blob subtype that you want to get when reading the Blob from database.

Set the ConversionSubType property to the Blob subtype that you want to store to database when writing Blob to database.

See Also

- SubType
- ConversionCharsetID
5.16.1.2.2.5 DbHandle Property

Contains the handle of the database where the BLOB is stored.

Class

TIBCBlob

Syntax

[property] DbHandle: TISC_DB_HANDLE;

Remarks

The DbHandle property is used to contain the handle of the database where the BLOB is stored.

See Also

• TIBCConnection.Handle

5.16.1.2.2.6 Handle Property

Contains the BLOB handle.

Class

TIBCBlob

Syntax

[property] Handle: TISC_BLOB_HANDLE;

Remarks

Contains the BLOB handle. Use the Handle property for direct calls to InterBase BLOB API.

See Also

• AllocBlob
5.16.1.2.2.7 ID Property

Contains BLOB ID.

Class

TIBCBlob

Syntax

```property
ID: TISC_QUAD;
```

Remarks

Contains BLOB ID that is actually stored in the BLOB field of the table record. It is a unique numeric value that references the BLOB data.

5.16.1.2.2.8 MaxSegmentSize Property

Indicates the largest BLOB segment size.

Class

TIBCBlob

Syntax

```property
MaxSegmentSize: Word;
```

Remarks

The MaxSegmentSize property is used to indicate the size in bytes of the largest segment of the BLOB.

5.16.1.2.2.9 NumSegments Property

Indicates the number of the BLOB segments in the database.
TIBCBlob

Syntax

property NumSegments: Cardinal;

Remarks

Use the NumSegments property to indicate the number of the BLOB segments in the database.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.2.2.10 Streamed Property

Indicates whether the BLOB is stream or segmented.

Class

TIBCBlob

Syntax

property Streamed: boolean;

Remarks

Use the Streamed property to determine whether the BLOB is stream or segmented. Segmented BLOBs are usual InterBase BLOBs and are stored in chunks. Stream BLOBs are stored as a continuous array of data bytes.

© 1997-2021 Devart. All Rights Reserved.

5.16.1.2.2.11 SubType Property

Source subtype for the BLOB subtype conversion using BLOB filters.

Class

TIBCBlob

Syntax

property SubType: integer;
Remarks
Source subtype for the BLOB subtype conversion using BLOB filters. It corresponds to
isc_bpbd_source_type parameter in BLOB Parameter Buffer (BPB) that is used for filtration.
For more information on BLOB subtypes and filters refer to InterBase API Guide.

See Also
- ConversionSubType
- CharSetID

© 1997-2021
Devart. All Rights Reserved.

5.16.1.2.2.12 TrHandle Property

Contains the handle of the transaction in which the BLOB is read or written.

Class
TIBCBlob

Syntax

| property | TrHandle: TISC_TR_HANDLE; |

Remarks
Use the TrHandle property to hold the handle of the transaction in which the BLOB is read or written.

See Also
- TIBCTransaction.Handle

© 1997-2021
Devart. All Rights Reserved.

5.16.1.2.3 Methods

Methods of the TIBCBlob class.

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

Public
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddRef</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td><strong>AllocBlob</strong></td>
<td>Allocates and initializes the Blob handle.</td>
</tr>
<tr>
<td><strong>Assign</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td><strong>Clear</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td><strong>CloseBlob</strong></td>
<td>Frees the Blob handle.</td>
</tr>
<tr>
<td><strong>FreeBlob</strong></td>
<td>Clears BLOB ID.</td>
</tr>
<tr>
<td><strong>IsInit</strong></td>
<td>Verifies BLOB ID initialization.</td>
</tr>
<tr>
<td><strong>LengthBlob</strong></td>
<td>Determines the number of bytes contained in the Blob object.</td>
</tr>
<tr>
<td><strong>LoadFromFile</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td><strong>LoadFromStream</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Copies the contents of a stream into the TBlob object.</td>
</tr>
<tr>
<td><strong>Read</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlob.</td>
</tr>
<tr>
<td><strong>ReadBlob</strong></td>
<td>Reads BLOB from the database.</td>
</tr>
<tr>
<td><strong>Release</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Decrements the reference count.</td>
</tr>
<tr>
<td><strong>SaveToFile</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Saves the contents of the TBlob object to a file.</td>
</tr>
<tr>
<td><strong>SaveToStream</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Copies the contents of a TBlob object to a stream.</td>
</tr>
<tr>
<td><strong>Truncate</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Sets new TBlob size and discards all data over it.</td>
</tr>
<tr>
<td><strong>Write</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Stores a raw sequence of bytes into a TBlob object.</td>
</tr>
<tr>
<td><strong>WriteBlob</strong></td>
<td>Writes BLOB to the database.</td>
</tr>
</tbody>
</table>

See Also

- TIBCBlob Class
5.16.1.2.3.1 AllocBlob Method

Allocates and initializes the Blob handle.

Class
TIBCBlob

Syntax

```pascal
procedure AllocBlob(Read: boolean = True);
```

Parameters

Read
True, if BLOB is opened for reading or writing.

Remarks

Call the AllocBLOB method to allocate and initialize the Blob handle. The Read parameter indicates whether the BLOB is opened for reading or writing. If the BLOB is opened for writing, new BLOB ID is generated.
5.16.1.2.3.3  FreeBlob Method

Clears BLOB ID.

Class
TIBCBlob

Syntax

procedure FreeBlob; override;

Remarks

Call the FreeBLOB method to clear BLOB ID.

5.16.1.2.3.4  IsInit Method

Verifies BLOB ID initialization.

Class
TIBCBlob

Syntax

function IsInit: boolean;

Return Value

True, if BLOB ID is initialized. False otherwise.

Remarks

The IsInit method verifies that BLOB ID is initialized.
5.16.1.2.3.5 LengthBlob Method

Determined the number of bytes contained in the Blob object.

Class

TIBCBlob

Syntax

```delphi
function LengthBlob: integer;
```

Return Value

The number of bytes contained in the Blob object.

Remarks

The LengthBLOB method returns the number of bytes contained in the Blob object.

© 1997-2021
Devart. All Rights Reserved.

5.16.1.2.3.6 ReadBlob Method

Reads BLOB from the database.

Class

TIBCBlob

Syntax

```delphi
procedure ReadBlob;
```

Remarks

Call the ReadBLOB method to read BLOB from the database.

© 1997-2021
Devart. All Rights Reserved.

5.16.1.2.3.7 WriteBlob Method

Writes BLOB to the database.

Class

© 2021 Devart
TIBCBlob

Syntax

```plaintext
procedure WriteBlob;
```

Remarks

Call the WriteBLOB method to write BLOB to the database.

5.16.2 Enumerations

Enumerations in the IBCClasses unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCIsolationLevel</td>
<td>Specifies the transaction isolation level and access mode.</td>
</tr>
</tbody>
</table>

5.16.2.1 TIBCIsolationLevel Enumeration

Specifies the transaction isolation level and access mode.

Unit

IBCClasses

Syntax

```plaintext
TIBCIsolationLevel = (iblSnapshot, iblReadCommitted,
iblReadOnlyReadCommitted, iblTableStability,
iblReadOnlyTableStability, iblCustom);
```

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
</table>
### iblCustom

The parameters of the transaction are set manually in the `Params` property.

### iblReadCommitted

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.

### iblReadOnlyCommitted

Enables the transaction to see all committed data in the database with read-only access mode.

### iblReadOnlyTableStability

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 cannot insert, update, and delete rows from these tables.

### iblSnapshot

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.

### iblTableStability

Provides a transaction sole insert, update, and delete access to the tables it uses. Other simultaneous transactions may still be able to select rows from these tables.

© 1997-2021 Devart. All Rights Reserved.

### 5.16.3 Routines

Routines in the `IBClasses` unit.

#### Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reverse2</td>
<td>Switches places of bytes in the word argument.</td>
</tr>
<tr>
<td>Reverse4</td>
<td>Switches places of bytes in the cardinal argument.</td>
</tr>
<tr>
<td>XSQLDA_LENGTH</td>
<td>Analogue of InterBase XSQLDA_LENGTH macro. Calculates the number of bytes that must be allocated for an input or output XSQLDA.</td>
</tr>
</tbody>
</table>
5.16.3.1 Reverse2 Function

Switches places of bytes in the word argument.

Unit
IBCClasses

Syntax

```pascal
function Reverse2(Value: word): Word;
```

Parameters

`Value`
Holds the input value.

Return Value
the output value.

5.16.3.2 Reverse4 Function

Switches places of bytes in the cardinal argument.

Unit
IBCClasses

Syntax

```pascal
function Reverse4(Value: cardinal): cardinal;
```

Parameters

`Value`
Holds the input value.

Return Value
the output value.
5.16.3.3 XSQLDA_LENGTH Function

Analogue of InterBase XSQLDA_LENGTH macro. Calculates the number of bytes that must be allocated for an input or output XSQLDA.

Unit
IBCClasses

Syntax

\[
\text{function } \text{XSQLDA_LENGTH}(n: \text{Long}; \text{XSQLVARType}: \text{TXSQLVARType}): \text{Long};
\]

Parameters

\(n\)
Holds the count of XSQLVAR.

\(XSQLVARType\)
Holds the type of XSQLVAR.

Return Value

the number of bytes that must be allocated for an input or output XSQLDA.

5.16.4 Variables

Variables in the IBCClasses unit.

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntegerPrecision</td>
<td>Set this constant to define the type of NUMERIC and DECIMAL fields with precision less or equal than IntegerPrecision as dtInteger. Otherwise, they are defined as dtFloat.</td>
</tr>
</tbody>
</table>
5.16.4.1 IntegerPrecision Variable

Set this constant to define the type of NUMERIC and DECIMAL fields with precision less or equal than IntegerPrecision as dtInteger. Otherwise, they are defined as dtFloat.

Unit

IBCClasses

Syntax

IntegerPrecision: integer = 10;

5.17 IBCConnectionPool

This unit contains the TIBCConnectionPoolManager class for managing connection pool.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCConnectionPoolManager</td>
<td>A class of methods that are used for managing IBDAC connection pool.</td>
</tr>
</tbody>
</table>

5.17.1 Classes

Classes in the IBCConnectionPool unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCConnectionPoolManager</td>
<td>A class of methods that are used for managing IBDAC connection pool.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
5.17.1.1  **TIBCConnectionPoolManager Class**

A class of methods that are used for managing IBDAC connection pool.

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

**Unit**

[IBCConnectionPool](#)

**Syntax**

```{vbnet}
TIBCConnectionPoolManager = class(TCRConnectionPoolManager);
```

**Remarks**

Use the TIBCConnectionPoolManager methods to manage IBDAC connection pool.

**Inheritance Hierarchy**

TCRConnectionPoolManager

  TIBCConnectionPoolManager

See Also

- [Connection Pooling](#)

© 1997-2021 Devart. All Rights Reserved.

5.17.1.1.1  **Members**

[TIBCConnectionPoolManager](#) class overview.

© 1997-2021 Devart. All Rights Reserved.

5.18  **IBCDataTypeMap**

5.18.1  **Constants**

Constants in the [IBCDataTypeMap](#) unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 2021 Devart
This unit contains the TIBCAArray class for representing the value of the InterBase array data type.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomIBCAArray</td>
<td>A base class representing the value of the InterBase array data type.</td>
</tr>
</tbody>
</table>

**5.19 IBCError**

IBCError unit implements the class.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIBCErr</td>
<td>A base class for exceptions raised when a component detects an InterBase error.</td>
</tr>
</tbody>
</table>

**5.19.1 Classes**

Classes in the IBCError unit.
EIBCError Class

A base class for exceptions raised when a component detects an InterBase error.

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

Unit

IBCError

Syntax

EIBCError = class (EDAError);

Remarks

EIBCError is raised when a component detects an InterBase error. Use EIBCError in an exception handling block.

Inheritance Hierarchy

EDAError

EIBCError

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Contains the component that caused the error.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Contains the component that caused the error.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>Determines the error code returned by the server.</td>
</tr>
<tr>
<td>ErrorNumber</td>
<td>Used to determine the error number returned by InterBase.</td>
</tr>
<tr>
<td>Sender</td>
<td>Holds the reference to the sender if exception is raised by the TComponent instance.</td>
</tr>
<tr>
<td>SQLErrorMsg</td>
<td>Holds the error message describing the part of the SQL code that caused the error.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.19.1.1.2.1 ErrorNumber Property

Used to determine the error number returned by InterBase.

Class

**EIBCError**

Syntax

```
property ErrorNumber: integer;
```

Remarks

Use the ErrorNumber property to determine the error number returned by InterBase.

See Also

- EDAError.ErrorCode

5.19.1.1.2.2 Sender Property

Holds the reference to the sender if exception is raised by the TComponent instance.

Class

**EIBCError**

Syntax

```
property Sender: TComponent;
```

Remarks

The Sender property holds the reference to the sender if exception is raised by the TComponent instance.
5.19.1.2.3  SQLErrorMsg Property

Holds the error message describing the part of the SQL code that caused the error.

Class

EIBCError

Syntax

property SQLErrorMsg: string;

Remarks

The error message describing the part of the SQL code that caused the error.

5.20  IBCLoader

This unit contains implementation of the TIBCLoader component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCLoader</td>
<td>This component serves for loading external data into the database table.</td>
</tr>
<tr>
<td>TIBCLoaderOptions</td>
<td>This class allows setting up the behaviour of the TIBCLoader class.</td>
</tr>
</tbody>
</table>

5.20.1  Classes

Classes in the IBCLoader unit.

Classes
### Name | Description
---|---
**TIBCLoader** | This component serves for loading external data into the database table.

**TIBCLoaderOptions** | This class allows setting up the behaviour of the TIBCLoader class.

#### 5.20.1.1 TIBCLoader Class

This component serves for loading external data into the database table.

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

**Unit**

**IBCLoader**

**Syntax**

```
TIBCLoader = class(TDALoader);
```

**Remarks**

The TIBCLoader component allows to load external data into the database table.

TIBCLoader serves for fast loading data to the database. To specify the name of the loading table set the **TIBCLoader.TableName** property. Use the **TIBCLoader.Columns** property to access individual columns. Write the **TIBCLoader.OnGetColumnData** or **TIBCLoader.OnPutData** event handlers to read external data and pass it to the database. Call the **Load** method to start loading data.

TIBCLoader loads data by executing INSERT statements. For Firebird 2.0 and higher several INSERT statements are combined in one EXECUTE BLOCK statement to speed up loading (the number of records that are sent to the server at once is controlled by the **TIBCLoaderOptions.RowsPerBatch** property).

The **TIBCLoaderOptions.InsertMode** property controls whether the INSERT INTO or UPDATE OR INSERT INTO statement will be used for loading data. Using of UPDATE OR INSERT INTO statements is available in Firebird 2.1 and higher.
Inheritance Hierarchy

**TDALoader**

**TIBCLoader**

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

5.20.1.1.1 Members

**TIBCLoader** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Used to access individual columns.</td>
</tr>
<tr>
<td>Connection</td>
<td>(inherited from <strong>TDALoader</strong>) property. Used to specify TCustomDACConnection in which TDALoader will be executed.</td>
</tr>
<tr>
<td>Options</td>
<td>This class allows setting up the behaviour of the TIBCLoader class.</td>
</tr>
<tr>
<td>TableName</td>
<td>Used to specify the name of the loading table set.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateColumns</td>
<td>(inherited from <strong>TDALoader</strong>) Creates <strong>TDAColumn</strong> objects for all fields of the table with the same name as <strong>TDALoader.TableName</strong>.</td>
</tr>
<tr>
<td>Load</td>
<td>(inherited from <strong>TDALoader</strong>) Starts loading data.</td>
</tr>
<tr>
<td>LoadFromDataSet</td>
<td>(inherited from <strong>TDALoader</strong>) Loads data from the specified dataset.</td>
</tr>
<tr>
<td>PutColumnData</td>
<td>(inherited from <strong>TDALoader</strong>) Overloaded. Puts the value of individual columns.</td>
</tr>
</tbody>
</table>

### Events
## Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection (inherited from <strong>TDALoader</strong>)</td>
<td>property. Used to specify TCustomDACConnection in which TDALoader will be executed.</td>
</tr>
</tbody>
</table>

## Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Used to access individual columns.</td>
</tr>
<tr>
<td>Options</td>
<td>This class allows setting up the behaviour of the TIBCLoader class.</td>
</tr>
<tr>
<td>TableName</td>
<td>Used to specify the name of the loading table set.</td>
</tr>
</tbody>
</table>

### See Also
- **TIBCLoader Class**
- **TIBCLoader Class Members**
5.20.1.2.1 Columns Property

Used to access individual columns.

**Class**

**TIBClassLoader**

**Syntax**

```property** Columns: TDAColumns;```

**Remarks**

Use the Columns property to access individual columns.

© 1997-2021 Devart. All Rights Reserved.

5.20.1.2.2 Options Property

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

**Class**

**TIBClassLoader**

**Syntax**

```property** Options: TIBClassLoaderOptions;```

© 1997-2021 Devart. All Rights Reserved.

5.20.1.2.3 TableName Property

Used to specify the name of the loading table set.

**Class**

**TIBClassLoader**

**Syntax**

```property** TableName: string;```
Remarks

Use the TableName property to specify the name of the loading table set.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnProgress</td>
<td>Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnGetColumnData</td>
<td>Used to read external data.</td>
</tr>
<tr>
<td>OnPutData</td>
<td>Used to pass external data to the database.</td>
</tr>
</tbody>
</table>

See Also

- TIBCLoader Class
- TIBCLoader Class Members

5.20.1.1.3 Events

Events of the TIBCLoader class.

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

5.20.1.1.3.1 OnGetColumnData Event

Used to read external data.

Class

TIBCLoader
5.20.1.1.3.2 OnGetColumnData Event

Used to pass external data to the database.

Class
TIBCLoader

Syntax

```pascal
property OnGetColumnData: TGetColumnDataEvent;
```

Remarks
Write the OnGetColumnData event handler to read external data.

5.20.1.2 TIBCLoaderOptions Class

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

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

Unit
TIBCLoader

Syntax

```pascal
TIBCLoaderOptions = class(TDALoaderOptions);
```

Inheritance Hierarchy
**TDALoaderOptions**

**TIBCLoaderOptions**

© 1997-2021 Devart. All Rights Reserved.

5.20.1.2.1 Members

**TIBCLoaderOptions** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InsertMode</td>
<td>Used to control the type of statement used for loading data.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used to quote columns names in the INSERT statement used for loading data.</td>
</tr>
<tr>
<td>RowsPerBatch</td>
<td>Used to control the number of records that are sent to the server at once.</td>
</tr>
<tr>
<td>UseBlankValues</td>
<td>(inherited from <strong>TDALoaderOptions</strong>) Forces IBDAC to fill the buffer with null values after loading a row to the database.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.20.1.2.2 Properties

Properties of the **TIBCLoaderOptions** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseBlankValues</td>
<td>(inherited from <strong>TDALoaderOptions</strong>) Forces IBDAC to fill the buffer with null values after</td>
</tr>
</tbody>
</table>
### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InsertMode</td>
<td>Used to control the type of statement used for loading data.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used to quote columns names in the INSERT statement used for loading data.</td>
</tr>
<tr>
<td>RowsPerBatch</td>
<td>Used to control the number of records that are sent to the server at once.</td>
</tr>
</tbody>
</table>

#### See Also
- [TIBCLoaderOptions Class](#)
- [TIBCLoaderOptions Class Members](#)

---

**5.20.1.2.2.1 InsertMode Property**

Used to control the type of statement used for loading data.

**Class**

[TIBCLoaderOptions](#)

**Syntax**

```property
property InsertMode: TIBCInsertMode default imInsert;
```

**Remarks**

Use the InsertMode property to specify the type of statement used for loading data using the [TIBCLoader](#) component.

When the property value is `imInsert`, then the INSERT INTO statement is used. When the property value is `imUpdateOrInsert`, then the UPDATE OR INSERT INTO statement is used.
Using of UPDATE OR INSERT INTO statements is available in Firebird 2.1 and higher.

The default value is `imInsert`.

5.20.1.2.2.2 QuoteNames Property

Used to quote columns names in the INSERT statement used for loading data.

Class

`TIBCLoaderOptions`

Syntax

```
property QuoteNames: boolean;
```

Remarks

Use the QuoteNames property to specify whether the column names in the INSERT statement will be quoted.

When the property value is `True`, then the column names will be quoted. When the property value is `False`, then the column names will not be quoted.

The default value is `False`.

5.20.1.2.3 RowsPerBatch Property

Used to control the number of records that are sent to the server at once.

Class

`TIBCLoaderOptions`

Syntax

```
property RowsPerBatch: integer default 50;
```

Remarks
Use the RowsPerBatch property to specify the number of records that are sent to the server at once when loading data using the TIBCLoader component.

TIBCLoader loads data by executing INSERT statements. For Firebird 2.0 and higher several INSERT statements (depending on the RowsPerBatch property value) are combined in one EXECUTE BLOCK statement to speed up loading.

The default value is 50.

### 5.21 IBCScript

This unit contains implementation of the TIBCScript component.

#### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCScript</td>
<td>A component for executing several SQL statements one by one.</td>
</tr>
</tbody>
</table>

#### 5.21.1 Classes

Classes in the IBCScript unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCScript</td>
<td>A component for executing several SQL statements one by one.</td>
</tr>
</tbody>
</table>

5.21.1.1 TIBCScript Class

A component for executing several SQL statements one by one.

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

Unit

TIBCScript

Syntax

TIBCScript = 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 TIBCSQL. Usually it is not a good solution. With only one TIBCScript 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 statements that can contain semicolon (for example CREATE TRIGGER or CREATE PROCEDURE) - only slash . 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 TIBCScript shows exception and continues execution.

Inheritance Hierarchy

TDAScript
  TIBCScript

See Also

- TIBCSQL

© 1997-2021 Devart. All Rights Reserved.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoDDL</strong></td>
<td>Used to indicate whether DDL statements must be executed in a separate transaction.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify the connection in which the script will be executed.</td>
</tr>
<tr>
<td><strong>DataSet</strong></td>
<td>Used to get or set script result dataset.</td>
</tr>
<tr>
<td><strong>Debug</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to display the script execution and all its parameter values.</td>
</tr>
<tr>
<td><strong>Delimiter</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to set the delimiter string that separates script statements.</td>
</tr>
<tr>
<td><strong>EndLine</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to get the current statement last line number in a script.</td>
</tr>
<tr>
<td><strong>EndOffset</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to get the offset in the last line of the current statement.</td>
</tr>
<tr>
<td><strong>EndPos</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to get the end position of the current statement.</td>
</tr>
<tr>
<td><strong>Macros</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to change SQL script text in design- or run-time easily.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to hold the parameters for a SQL script.</td>
</tr>
<tr>
<td><strong>SQL</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to get or set script text.</td>
</tr>
<tr>
<td><strong>StartLine</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to get the current statement start line number in a script.</td>
</tr>
<tr>
<td><strong>StartOffset</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to get the offset in the first line of the current statement.</td>
</tr>
<tr>
<td><strong>StartPos</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to get the start position of the current statement in a script.</td>
</tr>
<tr>
<td><strong>Statements</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Contains a list of statements obtained from the SQL property.</td>
</tr>
<tr>
<td><strong>Transaction</strong></td>
<td>Used to set or return the transaction to be used by the component.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BreakExec</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Stops script execution.</td>
</tr>
<tr>
<td><strong>ErrorOffset</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Used to get the offset of the statement if the Execute method raised an exception.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Executes a script.</td>
</tr>
<tr>
<td><strong>ExecuteFile</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Executes SQL statements contained in a file.</td>
</tr>
<tr>
<td><strong>ExecuteNext</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Executes the next statement in the script and then stops.</td>
</tr>
<tr>
<td><strong>ExecuteStream</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Executes SQL statements contained in a stream object.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Finds a macro with the specified name.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Occurs after a SQL script execution.</td>
</tr>
<tr>
<td><strong>BeforeExecute</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Occurs when taking a specific action before executing the current SQL statement is needed.</td>
</tr>
<tr>
<td><strong>OnError</strong></td>
<td>(inherited from <strong>TDAScript</strong>) Occurs when InterBase raises an error.</td>
</tr>
</tbody>
</table>
### InterBase Data Access Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EndLine</strong></td>
<td>(inherited from TDAScript) Used to get the current statement last line number in a script.</td>
</tr>
<tr>
<td><strong>EndOffset</strong></td>
<td>(inherited from TDAScript) Used to get the offset in the last line of the current statement.</td>
</tr>
<tr>
<td><strong>EndPos</strong></td>
<td>(inherited from TDAScript) Used to get the end position of the current statement.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to hold the parameters for a SQL script.</td>
</tr>
<tr>
<td><strong>StartLine</strong></td>
<td>(inherited from TDAScript) Used to get the current statement start line number in a script.</td>
</tr>
<tr>
<td><strong>StartOffset</strong></td>
<td>(inherited from TDAScript) Used to get the offset in the first line of the current statement.</td>
</tr>
<tr>
<td><strong>StartPos</strong></td>
<td>(inherited from TDAScript) Used to get the start position of the current statement in a script.</td>
</tr>
<tr>
<td><strong>Statements</strong></td>
<td>(inherited from TDAScript) Contains a list of statements obtained from the SQL property.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoDDL</strong></td>
<td>Used to indicate whether DDL statements must be executed in a separate transaction.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify the connection in which the script will be executed.</td>
</tr>
<tr>
<td><strong>DataSet</strong></td>
<td>Used to get or set script result dataset.</td>
</tr>
<tr>
<td><strong>Debug</strong> (inherited from TDAScript)</td>
<td>Used to display the script execution and all its parameter values.</td>
</tr>
<tr>
<td><strong>Delimiter</strong> (inherited from TDAScript)</td>
<td>Used to set the delimiter string that separates script statements.</td>
</tr>
<tr>
<td><strong>Macros</strong> (inherited from TDAScript)</td>
<td>Used to change SQL script</td>
</tr>
<tr>
<td><strong>text in design- or run-time easily.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SQL</strong> (inherited from <strong>TDAScript</strong>)</td>
<td>Used to get or set script text.</td>
</tr>
<tr>
<td><strong>Transaction</strong></td>
<td>Used to set or return the transaction to be used by the component.</td>
</tr>
</tbody>
</table>

### See Also
- **TIBCScript Class**
- **TIBCScript Class Members**

### 5.21.1.1.2.1  AutoDDL Property

#### Class
**TIBCScript**

#### Syntax

```plaintext
property AutoDDL: Boolean default True;
```

#### Remarks

Use the AutoDDL property to determine whether DDL statements must be executed in a separate transaction.

### 5.21.1.1.2.2  Connection Property

#### Class
**TIBCScript**

#### Syntax

```plaintext

```
**property** Connection: **TIBCConnection**;

### Remarks

Use the Connection property to specify the connection in which the script will be executed. If a connection has not been opened before running the Execute method, the TIBCConnection.Connect method is called.

### See Also

- [TIBCConnection](#)

© 1997-2021 Devart. All Rights Reserved.

---

### 5.21.1.1.2.3  DataSet Property

Used to get or set script result dataset.

### Class

**TIBCScript**

### Syntax

**property** DataSet: **TCustomIBCDataset**;

### Remarks

Use the DataSet property to get or set script result dataset.

© 1997-2021 Devart. All Rights Reserved.

---

### 5.21.1.1.2.4  Params Property

Used to hold the parameters for a SQL script.

### Class

**TIBCScript**

### Syntax

**property** Params: **TIBCParams**;
Remarks

Contains the parameters for a SQL script.

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
- TIBCParams

5.21.1.2.5 Transaction Property

Used to set or return the transaction to be used by the component.

Class

TIBCScript

Syntax

| property Transaction: TIBCTransaction stored | IsTransactionStored; |

Remarks

Use the Transaction property to set or return the transaction to be used by the component.

5.22 IBCSQLMonitor

This unit contains implementation of the TIBCSQLMonitor component.

Classes
### TIBCSQLMonitor Class

For a list of all members of this type, see `TIBCSQLMonitor` members.

#### Unit

**IBCSQLMonitor**

#### Syntax

```
TIBCSQLMonitor = class(TCustomDASQLMonitor);
```

#### Remarks

Use TIBCSQLMonitor to monitor dynamic SQL execution in IBDAC-based applications. TIBCSQLMonitor 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 TIBCSQLMonitor object. If an application has no TIBCSQLMonitor instance, the Debug window is available to display SQL statements to be sent.

#### Inheritance Hierarchy
TCustomDASQLMonitor
TIBCSQLMonitor

See Also
- TCustomDATDataSet.Debug
- TCustomDASQL.Debug
- DBMonitor

© 1997-2021 Devart. All Rights Reserved.

5.22.1.1.1 Members

**TIBCSQLMonitor** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from TCustomDASQLMonitor)</td>
<td>Used to activate monitoring of SQL.</td>
</tr>
<tr>
<td><strong>DBMonitorOptions</strong> (inherited from TCustomDASQLMonitor)</td>
<td>Used to set options for dbMonitor.</td>
</tr>
<tr>
<td><strong>Options</strong> (inherited from TCustomDASQLMonitor)</td>
<td>Used to include the desired properties for TCustomDASQLMonitor.</td>
</tr>
<tr>
<td><strong>TraceFlags</strong> (inherited from TCustomDASQLMonitor)</td>
<td>Used to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnSQL</strong> (inherited from TCustomDASQLMonitor)</td>
<td>Occurs when tracing of SQL activity on database components is needed.</td>
</tr>
</tbody>
</table>
5.23 IbDacVcl

This unit contains the visual constituent of IBDAC.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCConnectDialog</td>
<td>A class that provides a dialog box for user to supply his login information.</td>
</tr>
</tbody>
</table>

Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetIBCDatabaseList</td>
<td>Loads database name history from the registry.</td>
</tr>
<tr>
<td>GetIBCServerList</td>
<td>Loads server name history from the registry.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.23.1 Classes

Classes in the IbDacVcl unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCConnectDialog</td>
<td>A class that provides a dialog box for user to supply his login information.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.23.1.1 TIBCConnectDialog 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 TIBCConnectDialog members.
Unit

IbDacVcl

Syntax

TIBCConnectDialog = class(TCustomConnectDialog);

Remarks

The TIBCConnectDialog component is a direct descendant of TCustomConnectDialog class. Use TIBCConnectDialog to provide dialog box for user to supply server name, username, and password. You may want to customize appearance of dialog box using this class’s properties.

Inheritance Hierarchy

TCustomConnectDialog
  TIBCConnectDialog

See Also

- TCustomDAConnection.ConnectDialog

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.23.1.1.1 Members

### TIBCConnectDialog class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelButton</td>
<td>Used to specify the label for the Cancel button.</td>
</tr>
<tr>
<td>Caption</td>
<td>Used to set the caption of dialog box.</td>
</tr>
<tr>
<td>ConnectButton</td>
<td>Used to specify the label for the Connect button.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to indicate the TIBConnection component using the TIBCConnectDialog object.</td>
</tr>
</tbody>
</table>
### DatabaseLabel
- **Description**: Used to specify a prompt for database edit.

### DialogClass (inherited from `TCustomConnectDialog`)
- **Description**: Used to specify the class of the form that will be displayed to enter login information.

### LabelSet (inherited from `TCustomConnectDialog`)
- **Description**: Used to set the language of buttons and labels captions.

### PasswordLabel (inherited from `TCustomConnectDialog`)
- **Description**: Used to specify a prompt for password edit.

### ProtocolLabel
- **Description**: Used to specify a prompt for protocol box.

### Retries (inherited from `TCustomConnectDialog`)
- **Description**: Used to indicate the number of retries of failed connections.

### SavePassword (inherited from `TCustomConnectDialog`)
- **Description**: Used for the password to be displayed in `ConnectDialog` in asterisks.

### ServerLabel (inherited from `TCustomConnectDialog`)
- **Description**: Used to specify a prompt for the server name edit.

### StoreLogInfo (inherited from `TCustomConnectDialog`)
- **Description**: Used to specify whether the login information should be kept in system registry after a connection was established.

### UsernameLabel (inherited from `TCustomConnectDialog`)
- **Description**: Used to specify a prompt for username edit.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execute</strong> (inherited from <code>TCustomConnectDialog</code>)</td>
<td>Displays the connect dialog and calls the connection’s Connect method when user clicks the Connect button.</td>
</tr>
<tr>
<td><strong>GetServerList</strong> (inherited from <code>TCustomConnectDialog</code>)</td>
<td>Retrieves a list of available server names.</td>
</tr>
</tbody>
</table>
5.23.1.2 Properties

Properties of the TIBCConnectDialog class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelButton</td>
<td>(inherited from TCustomConnectDialog) Used to specify the label for the Cancel button.</td>
</tr>
<tr>
<td>Caption</td>
<td>(inherited from TCustomConnectDialog) Used to set the caption of dialog box.</td>
</tr>
<tr>
<td>ConnectButton</td>
<td>(inherited from TCustomConnectDialog) Used to specify the label for the Connect button.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to indicate the TIBConnection component using the TIBCConnectDialog object.</td>
</tr>
<tr>
<td>DatabaseLabel</td>
<td>Used to specify a prompt for database edit.</td>
</tr>
<tr>
<td>DialogClass</td>
<td>(inherited from TCustomConnectDialog) Used to specify the class of the form that will be displayed to enter login information.</td>
</tr>
<tr>
<td>LabelSet</td>
<td>(inherited from TCustomConnectDialog) Used to set the language of buttons and labels captions.</td>
</tr>
<tr>
<td>PasswordLabel</td>
<td>(inherited from TCustomConnectDialog) Used to specify a prompt for password edit.</td>
</tr>
<tr>
<td>ProtocolLabel</td>
<td>Used to specify a prompt for protocol box.</td>
</tr>
<tr>
<td>Retries</td>
<td>(inherited from TCustomConnectDialog) Used to indicate the number of retries of failed connections.</td>
</tr>
<tr>
<td>SavePassword</td>
<td>(inherited from TCustomConnectDialog) Used for the password to be displayed in ConnectDialog in asterisks.</td>
</tr>
<tr>
<td>ServerLabel</td>
<td>(inherited from TCustomConnectDialog) Used to specify a prompt for the server name edit.</td>
</tr>
<tr>
<td>StoreLogInfo</td>
<td>(inherited from TCustomConnectDialog) Used to specify whether the login information should be kept in system registry after a connection was</td>
</tr>
</tbody>
</table>
### UsernameLabel Property
(inherited from TCustomConnectDialog)

Used to specify a prompt for username edit.

See Also
- TIBConnectDialog Class
- TIBConnectDialog Class Members

#### Syntax

```property
Connection: TIBConnection;
```

#### Remarks

Read the Connection property to learn what TIBConnection component uses the TIBConnectDialog object. This property is read only.

See Also
- TCustomDACConnection.ConnectDialog

### DatabaseLabel Property

Used to specify a prompt for database edit.

See Also
- TIBConnectDialog Class Members

#### Syntax

```property
```

© 1997-2021 Devart. All Rights Reserved.
property DatabaseLabel: string;

Remarks
Use the DatabaseLabel property to specify a prompt for database edit.

© 1997-2021 Devart. All Rights Reserved.

5.23.1.2.3 ProtocolLabel Property

Used to specify a prompt for protocol box.

Class
TIBCConnectDialog

Syntax
property ProtocolLabel: string;

Remarks
Use the ProtocolLabel property to specify a prompt for protocol box.

© 1997-2021 Devart. All Rights Reserved.

5.23.2 Routines

Routines in the IbDacVcl unit.

Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetIBCDatabaseList</td>
<td>Loads database name history from the registry.</td>
</tr>
<tr>
<td>GetIBCDatabaseList</td>
<td>Loads database name history from the registry.</td>
</tr>
<tr>
<td>GetIBCServerList</td>
<td>Loads server name history from the registry.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.23.2.1 GetIBCDatabaseList Procedure

Loads database name history from the registry.

Unit

IbDacVcl

Syntax

procedure GetIBCDatabaseList(Server: string; List: TStrings);

Parameters

Server
  Specifies the server name for which the database name history will be loaded.

List
  Defines the string list, into which the database name list will be populated.

Remarks

In addition, the GetIBCDatabaseList procedure can be used when building a custom connection dialog as it is shown in the IBDAC demo.

5.23.2.2 GetIBCDatabaseList Procedure

Loads database name history from the registry.

Unit

IbDacVcl

Syntax

procedure GetIBCDatabaseList(ServerIndex: integer; List: TStrings);

Parameters

ServerIndex
  Specifies the index of the server name in the server list (starts from 1), for which the database name history will be loaded.

List
  Defines the string list, into which the database name list will be populated.
5.23.2.3 GetIBCServerList Procedure

Loads server name history from the registry.

Unit
IbDacVcl

Syntax

procedure GetIBCServerList(List: TStrings; WithEmpties: boolean = False);

Parameters

List
Defines the string list into which the server name list will be populated
WithEmpties
Specifies whether blank server names will be inserted into the list. The default value is False.

Remarks

In addition, the GetIBCServerList procedure can be used when building a custom connection dialog as it is shown in the IBDAC demo.

5.24 MemData

This unit contains classes for storing data in memory.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 2021 Devart
### TAttribute
TAttribute is not used in IBDAC.

### 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLocateExOptions</td>
<td>Represents the set of TLocateExOption.</td>
</tr>
<tr>
<td>TUpdateRecKinds</td>
<td>Represents the set of TUpdateRecKind.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCompressBlobMode</td>
<td>Specifies when the values should be compressed and the way they should be stored.</td>
</tr>
<tr>
<td>TConnLostCause</td>
<td>Specifies the cause of the connection loss.</td>
</tr>
<tr>
<td>TDANumericType</td>
<td>Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TLocateExOption</td>
<td>Allows to set additional search parameters which will be used by the LocateEx method.</td>
</tr>
<tr>
<td>TSortType</td>
<td>Specifies a sort type for string fields.</td>
</tr>
<tr>
<td>TUpdateRecKind</td>
<td>Indicates records for which the ApplyUpdates method will be performed.</td>
</tr>
</tbody>
</table>

5.24.1 Classes

Classes in the **MemData** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAttribute</td>
<td>TAttribute is not used in IBDAC.</td>
</tr>
<tr>
<td>TBlob</td>
<td>Holds large object value for field and parameter dtBlob, dtMemo data types.</td>
</tr>
<tr>
<td>TCompressedBlob</td>
<td>Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.</td>
</tr>
<tr>
<td>TDBObject</td>
<td>A base class for classes that work with user-defined data types that have attributes.</td>
</tr>
<tr>
<td>TMemData</td>
<td>Implements in-memory database.</td>
</tr>
<tr>
<td>TObjectType</td>
<td>This class is not used.</td>
</tr>
<tr>
<td>TSharedObject</td>
<td>A base class that allows to simplify memory management for object referenced by several other objects.</td>
</tr>
</tbody>
</table>
5.24.1.1  TAttribute Class

TAttribute is not used in IBDAC.

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

Unit
MemData

Syntax

TAttribute = class(System.TObject);

5.24.1.1.1  Members

TAttribute class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeNo</td>
<td>Returns an attribute's ordinal position in object.</td>
</tr>
<tr>
<td>DataSize</td>
<td>Returns the size of an attribute value in internal representation.</td>
</tr>
<tr>
<td>DataType</td>
<td>Returns the type of data that was assigned to the Attribute.</td>
</tr>
<tr>
<td>Length</td>
<td>Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Returns a TObjectType object for an object attribute.</td>
</tr>
<tr>
<td>Offset</td>
<td>Returns an offset of the attribute value in internal representation.</td>
</tr>
<tr>
<td>Owner</td>
<td>Indicates TObjectType that uses the attribute to represent one of its attributes.</td>
</tr>
</tbody>
</table>
Properties of the `TAttribute` class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeNo</td>
<td>Returns an attribute's ordinal position in object.</td>
</tr>
<tr>
<td>DataSize</td>
<td>Returns the size of an attribute value in internal representation.</td>
</tr>
<tr>
<td>DataType</td>
<td>Returns the type of data that was assigned to the Attribute.</td>
</tr>
<tr>
<td>Length</td>
<td>Returns the length of the string for <code>dtString</code> attribute and precision for <code>dtInteger</code> and <code>dtFloat</code> attribute.</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Returns a TObjectType object for an object attribute.</td>
</tr>
<tr>
<td>Offset</td>
<td>Returns an offset of the attribute value in internal representation.</td>
</tr>
<tr>
<td>Owner</td>
<td>Indicates TObjectType that uses the attribute to represent one of its attributes.</td>
</tr>
<tr>
<td>Scale</td>
<td>Returns the scale of <code>dtFloat</code> and <code>dtInteger</code> attributes.</td>
</tr>
<tr>
<td>Size</td>
<td>Returns the size of an attribute value in external representation.</td>
</tr>
</tbody>
</table>
5.24.1.1.2.1  AttributeNo Property

Returns an attribute's ordinal position in object.

Class

TAttribute

Syntax

property AttributeNo: Word;

Remarks

Use the AttributeNo property to learn an attribute's ordinal position in object, where 1 is the first field.

See Also

- TObjectType.Attributes

5.24.1.1.2.2  DataSize Property

Returns the size of an attribute value in internal representation.

Class

TAttribute

Syntax

property DataSize: Integer;

Remarks
Use the DataSize property to learn the size of an attribute value in internal representation.

For example:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtDate</td>
<td>17 (sizeof(OCIDate))</td>
</tr>
<tr>
<td>dtFloat</td>
<td>22 (sizeof(OCINumber))</td>
</tr>
<tr>
<td>dtInteger</td>
<td>22 (sizeof(OCINumber))</td>
</tr>
</tbody>
</table>

5.24.1.1.2.3  DataType Property

Returns the type of data that was assigned to the Attribute.

Class

TAttribute

Syntax

```
property DataType: Word;
```

Remarks

Use the DataType property to discover the type of data that was assigned to theAttribute.

Possible values: dtDate, dtFloat, dtInteger, dtString, dtObject.

5.24.1.1.2.4  Length Property

Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.

Class

TAttribute
Syntax

```pascal
property Length: Word;
```

Remarks

Use the `Length` property to learn the length of the string for `dtString` attribute and precision for `dtInteger` and `dtFloat` attribute.

See Also

- `Scale`

© 1997-2021
Devart. All Rights Reserved.

5.24.1.1.2.5 ObjectType Property

Returns a TObjectType object for an object attribute.

Class

`TAttribute`

Syntax

```pascal
property ObjectType: TObjectType;
```

Remarks

Use the `ObjectType` property to return a TObjectType object for an object attribute.

© 1997-2021
Devart. All Rights Reserved.

5.24.1.1.2.6 Offset Property

Returns an offset of the attribute value in internal representation.

Class

`TAttribute`

Syntax

```pascal
property Offset: Integer;
```
Remarks

Use the DataSize property to learn an offset of the attribute value in internal representation.

Class

TAttribute

Syntax

```delphi
property Owner: TObjectType;
```

Remarks

Indicates TObjectType that uses the attribute to represent one of its attributes.

5.24.1.1.2.7 Owner Property

Check the value of the Owner property to determine TObjectType that uses the attribute to represent one of its attributes. Applications should not assign the Owner property directly.

It is assigned automatically when the attribute is created from a TOraType.Describe.

5.24.1.1.2.8 Scale Property

Returns the scale of dtFloat and dtInteger attributes.

Class

TAttribute

Syntax

```delphi
property Scale: Word;
```

Remarks

Use the Scale property to learn the scale of dtFloat and dtInteger attributes.

See Also
5.24.1.9 Size Property

Returns the size of an attribute value in external representation.

Class

TAttribute

Syntax

```plaintext
property Size: Integer;
```

Remarks

Read Size to learn the size of an attribute value in external representation.

For example:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (in bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtDate</td>
<td>8</td>
<td>sizeof(TDateTime)</td>
</tr>
<tr>
<td>dtFloat</td>
<td>8</td>
<td>sizeof(Double)</td>
</tr>
<tr>
<td>dtInteger</td>
<td>4</td>
<td>sizeof(Integer)</td>
</tr>
</tbody>
</table>

See Also

- DataSize

5.24.1.2 TBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types.

For a list of all members of this type, see TBlob members.
**Syntax**

```cpp
TBlob = class(TSharedObject);
```

**Remarks**

Object TBlob holds large object value for the field and parameter dtBlob, dtMemo, dtWideMemo data types.

**Inheritance Hierarchy**

```
TSharedObject
   TBlob
```

**See Also**

- `TMemDataSet.GetBlob`

---

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>IsUnicode</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

---

### Methods
## Properties

Properties of the `TBlob` class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsUnicode</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td>RefCount</td>
<td>(inherited from TSharedObject) Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

See Also
- TBlob Class
- TBlob Class Members

5.24.1.2.2.1 AsString Property

Used to manipulate BLOB value as string.

Class
- TBlob

Syntax

```property
property AsString: string;
```

Remarks

Use the AsString property to manipulate BLOB value as string.

See Also
- Assign
- AsWideString
5.24.1.2.2.2  AsWideString Property

Used to manipulate BLOB value as Unicode string.

Class

TBlob

Syntax

property AsWideString: string;

Remarks

Use the AsWideString property to manipulate BLOB value as Unicode string.

See Also

• Assign
• AsString

5.24.1.2.2.3  IsUnicode Property

Gives choice of making TBlob store and process data in Unicode format or not.

Class

TBlob

Syntax

property IsUnicode: boolean;

Remarks

Set IsUnicode to True if you want TBlob to store and process data in Unicode format.

Note: changing this property raises an exception if TBlob is not empty.
5.24.1.2.2.4 Size Property

Used to learn the size of the TBlob value in bytes.

Class

TBlob

Syntax

property size: Cardinal;

Remarks

Use the Size property to find out the size of the TBlob value in bytes.

© 1997-2021 Devart. All Rights Reserved.

5.24.1.2.3 Methods

Methods of the TBlob class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Copies the contents of a stream into the TBlob object.</td>
</tr>
<tr>
<td>Read</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlob.</td>
</tr>
<tr>
<td>Release (inherited from TSharedObject)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>
### Assign Method

Sets BLOB value from another TBlob object.

#### Class

**TBlob**

#### Syntax

```pascal
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

- **TBlob Class**
- **TBlob Class Members**

© 1997-2021 Devart. All Rights Reserved.
5.24.1.2.3.2 Clear Method

Deletes the current value in TBlob object.

Class

TBlob

Syntax

procedure Clear; virtual;

Remarks

Call the Clear method to delete the current value in TBlob object.

5.24.1.2.3.3 LoadFromFile Method

Loads the contents of a file into a TBlob object.

Class

TBlob

Syntax

procedure LoadFromFile(const FileName: string);

Parameters

FileName

Holds the name of the file from which the TBlob value is loaded.

Remarks

Call the LoadFromFile method to load the contents of a file into a TBlob object. Specify the name of the file to load into the field as the value of the FileName parameter.

See Also

• SaveToFile
5.24.1.2.3.4 LoadFromStream Method

Copies the contents of a stream into the TBlob object.

Class
TBlob

Syntax

```delphi
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

© 1997-2021 Devart. All Rights Reserved.

5.24.1.2.3.5 Read Method

Acquires a raw sequence of bytes from the data stored in TBlob.

Class
TBlob

Syntax

```delphi
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.

\textit{Dest}
Holds a pointer to the memory area where to store the sequence.

**Return Value**
Actually read byte count if the sequence crosses object size limit.

**Remarks**
Call the Read method to acquire a raw sequence of bytes from the data stored in TBlob.

The Position parameter is the starting point of byte sequence which lasts Count number of
bytes. The Dest parameter is a pointer to the memory area where to store the sequence.
If the sequence crosses object size limit, function will return actually read byte count.

**See Also**
- Write

5.24.1.2.3.6 SaveToFile Method

Saves the contents of the TBlob object to a file.

**Class**
TBlob

**Syntax**

\texttt{procedure SaveToFile(\texttt{const FileName: string});}

**Parameters**

\textit{FileName}
Holds a string that contains the name of the file.

**Remarks**
Call the SaveToFile method to save the contents of the TBlob object to a file. Specify the
name of the file as the value of the FileName parameter.

**See Also**
- LoadFromFile
5.24.1.2.3.7  SaveToStream Method

Copies the contents of a TBlob object to a stream.

Class

TBlob

Syntax

procedure SaveToStream(Stream: TStream); virtual;

Parameters

Stream
Holds the name of the stream.

Remarks

Call the SaveToStream method to copy the contents of a TBlob object to a stream. Specify
the name of the stream to which the field's value is saved as the value of the Stream
parameter.

See Also

• LoadFromStream

5.24.1.2.3.8  Truncate Method

Sets new TBlob size and discards all data over it.

Class

TBlob

Syntax

procedure Truncate(NewSize: Cardinal); virtual;

Parameters

NewSize
Holds the new size of TBlob.
Remarks

Call the Truncate method to set new TBlob size and discard all data over it. If NewSize is greater or equal TBlob.Size, it does nothing.

Stores a raw sequence of bytes into a TBlob object.

Class

TBlob

Syntax

```
procedure Write(Position: Cardinal; Count: Cardinal; Source: IntPtr); virtual;
```

Parameters

- **Position**
  - Holds the starting point of the byte sequence.

- **Count**
  - Holds the size of the sequence in bytes.

- **Source**
  - Holds a pointer to a source memory area.

Remarks

Call the Write method to store a raw sequence of bytes into a TBlob object.

The Position parameter is the starting point of byte sequence which lasts Count number of bytes. The Source parameter is a pointer to a source memory area.

If the value of the Position parameter crosses current size limit of TBlob object, source data will be appended to the object data.

See Also

- [Read](#)
5.24.1.3 TCompressedBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.

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

Unit

MemData

Syntax

TCompressedBlob = class(TBlob);

Remarks

TCompressedBlob is a descendant of the TBlob class. It holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data. For more information about using BLOB compression see TCustomDADataSet.Options.

Note: Internal compression functions are available in CodeGear Delphi 2007 for Win32, Borland Developer Studio 2006, Borland Delphi 2005, and Borland Delphi 7. To use BLOB compression under Borland Delphi 6 and Borland C++ Builder you should use your own compression functions. To use them set the CompressProc and UncompressProc variables declared in the MemUtils unit.

Example

```pascal
type
  TCompressProc = function(dest: IntPtr; destLen: IntPtr; const source: IntPtr; sourceLen: longint): longint;
  TUncompressProc = function(dest: IntPtr; destlen: IntPtr; source: IntPtr; sourceLine: longint): longint;
var
  CompressProc: TCompressProc;
  UncompressProc: TUncompressProc;
```

Inheritance Hierarchy

```
TSharedObject
  TBlob
    TCompressedBlob
```

See Also

- TBlob
- TMemDataSet.GetBlob
### TCompressedBlob class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString (inherited from TBlob)</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString (inherited from TBlob)</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>Compressed</td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td>CompressedSize</td>
<td>Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td>IsUnicode (inherited from TBlob)</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td>RefCount (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size (inherited from TBlob)</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign (inherited from TBlob)</td>
<td>Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td>Clear (inherited from TBlob)</td>
<td>Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td>LoadFromFile (inherited from TBlob)</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
</tbody>
</table>
**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.

---

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

**5.24.1.3.2 Properties**

Properties of the **TCompressedBlob** class.

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

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsString</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td><strong>AsWideString</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td><strong>Compressed</strong></td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td><strong>CompressedSize</strong></td>
<td>Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td><strong>IsUnicode</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td><strong>RefCount</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a</td>
</tr>
</tbody>
</table>
See Also
- [TCompressedBlob Class](#)
- [TCompressedBlob Class Members](#)

### 5.24.1.3.2.1 Compressed Property

**Syntax**

```delphi
class TCompressedBlob

property Compressed: boolean;

end
```

**Remarks**

Indicates whether the Blob is compressed. Set this property to True or False to compress or decompress the Blob.

### 5.24.1.3.2.2 CompressedSize Property

**Syntax**

```delphi
class TCompressedBlob

property CompressedSize: Cardinal;

end
```

**Remarks**

Used to indicate compressed size of the Blob data.
Indicates compressed size of the Blob data.

5.24.1.4  **TDBObject Class**

A base class for classes that work with user-defined data types that have attributes.

For a list of all members of this type, see **TDBObject** members.

**Unit**  
**MemData**

**Syntax**

```pascal
TDBObject = class(TSharedObject);
```

**Remarks**

TDBObject is a base class for classes that work with user-defined data types that have attributes.

**Inheritance Hierarchy**

- **TSharedObject**
- **TDBObject**

5.24.1.4.1  **Members**

**TDBObject** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>(inherited from <strong>TSharedObject</strong>) Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

**Methods**
5.24.1.5 TMemData Class

Implements in-memory database.

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

Unit

MemData

Syntax

TMemData = class(TData);

Inheritance Hierarchy

TData

  TMemData

5.24.1.5.1 Members

TMemData class overview.

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.24.1.6 TObjectType Class

This class is not used.

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

© 2021 Devart
Unit

MemData

Syntax

TObjectType = class(TSharedObject);

Inheritance Hierarchy

TSharedObject

TObjectType

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.24.1.6.1 Members

**TObjectType** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeCount</td>
<td>Used to indicate the number of attributes of type.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Used to access separate attributes.</td>
</tr>
<tr>
<td>DataType</td>
<td>Used to indicate the type of object dtObject, dtArray or dtTable.</td>
</tr>
<tr>
<td>RefCount</td>
<td>(inherited from TSharedObject) Used to return the count of reference to a</td>
</tr>
<tr>
<td></td>
<td>TSharedObject object.</td>
</tr>
<tr>
<td>Size</td>
<td>Used to learn the size of an object instance.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>(inherited from TSharedObject) Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AttributeCount</td>
<td>Used to indicate the number of attributes of type.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Used to access separate attributes.</td>
</tr>
<tr>
<td>DataType</td>
<td>Used to indicate the type of object dtObject, dtArray or dtTable.</td>
</tr>
<tr>
<td>RefCount (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size</td>
<td>Used to learn the size of an object instance.</td>
</tr>
</tbody>
</table>

See Also
- TObjectType Class
- TObjectType Class Members
5.24.1.6.2.1 AttributeCount Property

Used to indicate the number of attributes of type.

Class

TObjectType

Syntax

property AttributeCount: Integer;

Remarks

Use the AttributeCount property to determine the number of attributes of type.

5.24.1.6.2.2 Attributes Property(Indexer)

Used to access separate attributes.

Class

TObjectType

Syntax

property Attributes[Index: integer]: TAttribute;

Parameters

Index

Holds the attribute’s ordinal position.

Remarks

Use the Attributes property to access individual attributes. The value of the Index parameter corresponds to the AttributeNo property of TAttribute.

See Also

- TAttribute
- FindAttribute
5.24.1.6.2.3  **DataType Property**

Used to indicate the type of object dtObject, dtArray or dtTable.

**Class**

`TObjectType`

**Syntax**

```delphi
property DataType: Word;
```

**Remarks**

Use the `DataType` property to determine the type of object dtObject, dtArray or dtTable.

© 1997-2021 Devart. All Rights Reserved.

5.24.1.6.2.4  **Size Property**

Used to learn the size of an object instance.

**Class**

`TObjectType`

**Syntax**

```delphi
property Size: Integer;
```

**Remarks**

Use the `Size` property to find out the size of an object instance. Size is a sum of all attribute sizes.

**See Also**

- `TAttribute.Size`

© 1997-2021 Devart. All Rights Reserved.

Request Support   DAC Forum   Provide Feedback
5.24.1.6.3 Methods

Methods of the TObjectType class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>FindAttribute</td>
<td>Indicates whether a specified Attribute component is referenced in the TAttributes object.</td>
</tr>
<tr>
<td>Release (inherited from TSharedObject)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

See Also
- TObjectType Class
- TObjectType Class Members

© 1997-2021 Devart. All Rights Reserved.

5.24.1.6.3.1 FindAttribute Method

Indicates whether a specified Attribute component is referenced in the TAttributes object.

Class
- TObjectType

Syntax

```
function FindAttribute(const Name: string): TAttribute; virtual;
```

Parameters

Name
- Holds the name of the attribute to search for.

Return Value
TAttribute, if an attribute with a matching name was found. Nil Otherwise.

Remarks
Call FindAttribute to determine if a specified Attribute component is referenced in the TAttributes object. Name is the name of the Attribute for which to search. If FindAttribute finds an Attribute with a matching name, it returns the TAttribute. Otherwise it returns nil.

See Also
- TAttribute
- Attributes

5.24.1.7 TSharedObject Class
A base class that allows to simplify memory management for object referenced by several other objects.
For a list of all members of this type, see TSharedObject members.

Unit
MemData

Syntax

```
TSharedObject = class(System.TObject);
```

Remarks
TSharedObject allows to simplify memory management for object referenced by several other objects. TSharedObject holds a count of references to itself. When any object (referer object) is going to use TSharedObject, it calls the TSharedObject.AddRef method. Referer object has to call the TSharedObject.Release method after using TSharedObject.

See Also
- TBlob
- TObjectType
5.24.1.7.1 Members

**TSharedObject** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

5.24.1.7.2 Properties

Properties of the **TSharedObject** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

See Also

- **TSharedObject Class**
- **TSharedObject Class Members**
5.24.1.7.2.1 RefCount Property

Used to return the count of reference to a TSharedObject object.

Class
TSharedObject

Syntax

```plaintext
property RefCount: Integer;
```

Remarks

Returns the count of reference to a TSharedObject object.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

See Also

- TSharedObject Class
- TSharedObject Class Members

© 1997-2021 Devart. All Rights Reserved.
5.24.1.7.3.1 AddRef Method

Increments the reference count for the number of references dependent on the TSharedObject object.

Class

TSharedObject

Syntax

procedure AddRef;

Remarks

Increments the reference count for the number of references dependent on the TSharedObject object.

See Also

• Release

5.24.1.7.3.2 Release Method

Decrements the reference count.

Class

TSharedObject

Syntax

procedure Release;

Remarks

Call the Release method to decrement the reference count. When RefCount is 1, TSharedObject is deleted from memory.

See Also

• AddRef
5.24.2 Types

Types in the **MemData** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLocateExOptions</td>
<td>Represents the set of TLocateExOption.</td>
</tr>
<tr>
<td>TUpdateRecKinds</td>
<td>Represents the set of TUpdateRecKind.</td>
</tr>
</tbody>
</table>

### 5.24.2.1 TLocateExOptions Set

Represents the set of TLocateExOption.

**Unit**

**MemData**

**Syntax**

```
TLocateExOptions = set of TLocateExOption;
```

### 5.24.2.2 TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

**Unit**

**MemData**

**Syntax**

```
TUpdateRecKinds = set of TUpdateRecKind;
```
5.24.3 Enumerations

Enumerations in the **MemData** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCompressBlobMode</td>
<td>Specifies when the values should be compressed and the way they should be stored.</td>
</tr>
<tr>
<td>TConnLostCause</td>
<td>Specifies the cause of the connection loss.</td>
</tr>
<tr>
<td>TDANumericType</td>
<td>Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.</td>
</tr>
<tr>
<td>TLocateExOption</td>
<td>Allows to set additional search parameters which will be used by the LocateEx method.</td>
</tr>
<tr>
<td>TSortType</td>
<td>Specifies a sort type for string fields.</td>
</tr>
<tr>
<td>TUpdateRecKind</td>
<td>Indicates records for which the ApplyUpdates method will be performed.</td>
</tr>
</tbody>
</table>

### 5.24.3.1 TCompressBlobMode Enumeration

Specifies when the values should be compressed and the way they should be stored.

**Unit**

**MemData**

**Syntax**

```
TCompressBlobMode = (cbNone, cbClient, cbServer, cbClientServer);
```
Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbClient</td>
<td>Values are compressed and stored as compressed data at the client side. Before posting data to the server decompression is performed and data at the server side stored in the original form. Allows to reduce used client memory due to increase access time to field values. The time spent on the opening DataSet and executing Post increases.</td>
</tr>
<tr>
<td>cbClientServer</td>
<td>Values are compressed and stored in compressed form. Allows to decrease the volume of used memory at client and server sides. Access time to the field values increases as for cbClient. The time spent on opening DataSet and executing Post decreases. <strong>Note:</strong> On using cbServer or cbClientServer data on the server is stored as compressed. Other applications can add records in uncompressed format but can't read and write already compressed data. If compressed BLOB is partially changed by another application (if signature was not changed), DAC will consider its value as NULL. Blob compression is not applied to Memo fields because of possible cutting.</td>
</tr>
<tr>
<td>cbNone</td>
<td>Values not compressed. The default value.</td>
</tr>
<tr>
<td>cbServer</td>
<td>Values are compressed before passing to the server and store at the server in compressed form. Allows to decrease database size on the server. Access time to the field values does not change. The time spent on opening DataSet and executing Post usually decreases.</td>
</tr>
</tbody>
</table>

5.24.3.2 TConnLostCause Enumeration

Specifies the cause of the connection loss.

Unit

**MemData**

Syntax

```pascal
TConnLostCause = (clUnknown, clExecute, clOpen, clRefresh, clApply, clServiceQuery, clTransStart, clConnectionApply, clConnect);
```

Values
### 5.24.3.3 TDANumericType Enumeration

Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.

**Unit**

MemData

**Syntax**

```plaintext
tDANumericType = (ntFloat, ntBCD, ntFmtBCD);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntBCD</td>
<td>Data is stored on the client side as currency and represented as TBCDFIELD. This format allows storing data with precision up to 0,0001.</td>
</tr>
<tr>
<td>ntFloat</td>
<td>Data stored on the client side is in double format and represented as TFloatField. The default value.</td>
</tr>
<tr>
<td>ntFmtBCD</td>
<td>Data is represented as TFMTBCDFIELD. TFMTBCDFIELD gives</td>
</tr>
</tbody>
</table>
greater precision and accuracy than TBCDField, but it is slower.

5.24.3.4 TLocateExOption Enumeration

Allows to set additional search parameters which will be used by the LocateEx method.

Unit

MemData

Syntax

TLocateExOption = (lxCaselnSensitive, lxPartialKey, lxNearest, lxNext, lxUp, lxPartialCompare);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lxCaseInsensitive</td>
<td>Similar to loCaselnSensitive. Key fields and key values are matched without regard to the case.</td>
</tr>
<tr>
<td>lxNearest</td>
<td>LocateEx moves the cursor to a specific record in a dataset or to the first record in the dataset that is greater than the values specified in the KeyValues parameter. For this option to work correctly, the dataset should be sorted by the fields the search is performed in. If the dataset is not sorted, the function may return a line that is not connected with the search condition.</td>
</tr>
<tr>
<td>lxNext</td>
<td>LocateEx searches from the current record.</td>
</tr>
<tr>
<td>lxPartialCompare</td>
<td>Similar to lxPartialKey, but the difference is that it can process value entries in any position. For example, 'HAM' would match both 'HAMM', 'HAMMER.', and also 'MR HAMMER'.</td>
</tr>
<tr>
<td>lxPartialKey</td>
<td>Similar to lxPartialKey. Key values can include only a part of the matching key field value. For example, 'HAM' would match both 'HAMM' and 'HAMMER.', but not 'MR HAMMER'.</td>
</tr>
<tr>
<td>lxUp</td>
<td>LocateEx searches from the current record to the first record.</td>
</tr>
</tbody>
</table>
5.24.3.5  TSortType Enumeration

Specifies a sort type for string fields.

Unit

MemData

Syntax

TSortType = (stCaseSensitive, stCaseInsensitive, stBinary);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>stBinary</td>
<td>Sorting by character ordinal values (this comparison is also case sensitive).</td>
</tr>
<tr>
<td>stCaseInsensitive</td>
<td>Sorting without case sensitivity.</td>
</tr>
<tr>
<td>stCaseSensitive</td>
<td>Sorting with case sensitivity.</td>
</tr>
</tbody>
</table>

5.24.3.6  TUpdateRecKind Enumeration

Indicates records for which the ApplyUpdates method will be performed.

Unit

MemData

Syntax

TUpdateRecKind = (ukUpdate, ukInsert, ukDelete);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ukDelete</td>
<td>ApplyUpdates will be performed for deleted records.</td>
</tr>
<tr>
<td>ukInsert</td>
<td>ApplyUpdates will be performed for inserted records.</td>
</tr>
<tr>
<td>ukUpdate</td>
<td>ApplyUpdates will be performed for updated records.</td>
</tr>
</tbody>
</table>
5.25 MemDS

This unit contains implementation of the TMemDataSet class.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMemDataSet</td>
<td>A base class for working with data and manipulating data in memory.</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoNotRaiseExceptionOnUAFail</td>
<td>An exception will be raised if the value of the UpdateAction parameter is uaFail.</td>
</tr>
<tr>
<td>SendDataSetChangeEventAfterOpen</td>
<td>The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.</td>
</tr>
</tbody>
</table>

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

5.25.1 Classes

Classes in the MemDS unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMemDataSet</td>
<td>A base class for working with data and manipulating data in memory.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
5.25.1.1 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.

© 1997-2021 Devart. All Rights Reserved.

5.25.1.1.1 Members

TMemDataSet class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit</td>
</tr>
</tbody>
</table>
update of rows on database server.

**Prepared**
Determines whether a query is prepared for execution or not.

**Ranged**
Indicates whether a range is applied to a dataset.

**UpdateRecordTypes**
Used to indicate the update status for the current record when cached updates are enabled.

**UpdatesPending**
Used to check the status of the cached updates buffer.

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong></td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are</td>
</tr>
</tbody>
</table>
## 5.25.1.1.2 Properties

Properties of the `TMemDataSet` class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CachedUpdates</code></td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><code>IndexFieldNames</code></td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><code>KeyExclusive</code></td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><code>LocalConstraints</code></td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of</td>
</tr>
</tbody>
</table>
### CachedUpdates Property

Used to enable or disable the use of cached updates for a dataset.

**Class**

**TMemDataSet**

**Syntax**

```delphi
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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>
Cached updates are especially useful for client applications working with remote database servers. Enabling cached updates brings up the following benefits:

- Fewer transactions and shorter transaction times.
- Minimized network traffic.

The potential drawbacks of enabling cached updates are:

- Other applications can access and change the actual data on the server while users are editing local copies of data, resulting in an update conflict when cached updates are applied to the database.
- Other applications cannot access data changes made by an application until its cached updates are applied to the database.

The default value is False.

**Note:** When establishing master/detail relationship the CachedUpdates property of detail dataset works properly only when `TDADatasetOptions.LocalMasterDetail` is set to True.

**See Also**

- `UpdatesPending`
- `TMemDataSet.ApplyUpdates`
- `RestoreUpdates`
- `CommitUpdates`
- `CancelUpdates`
- `UpdateStatus`
- `TCustomDADataset.Options`

© 1997-2021 Devart. All Rights Reserved.

5.25.1.2.2 IndexFieldNames Property

Used to get or set the list of fields on which the recordset is sorted.

**Class**

`TMemDataSet`

**Syntax**

```
property IndexFieldNames: string;
```

**Remarks**
Use the IndexFieldNames property to get or set the list of fields on which the recordset is sorted. Specify the name of each column in IndexFieldNames to use as an index for a table. Ordering of column names is significant. Separate names with semicolon. The specified columns don't need to be indexed. Set IndexFieldNames to an empty string to reset the recordset to the sort order originally used when the recordset's data was first retrieved.

Each field may optionally be followed by the keyword ASC / DESC or CIS / CS / BIN.

Use ASC, DESC keywords to specify a sort direction for the field. If one of these keywords is not used, the default sort direction for the field is ascending.

Use CIS, CS or BIN keywords to specify a sort type for string fields:

CIS - compare without case sensitivity;
CS - compare with case sensitivity;
BIN - compare by character ordinal values (this comparison is also case sensitive).

If a dataset uses a TCustomDAConnection component, the default value of sort type depends on the TCustomDAConnection.Options option of the connection. If a dataset does not use a connection (TVirtualTable dataset), the default is CS.

Read IndexFieldNames to determine the field (or fields) on which the recordset is sorted.

Ordering is processed locally.

**Note:** You cannot process ordering by BLOB fields. IndexFieldNames cannot be set to True when TCustomDADataset.UniDirectional=True.

**Example**

The following procedure illustrates how to set IndexFieldNames in response to a button click:

```pascal
DataSet1.IndexFieldNames := 'LastName ASC CIS; DateDue DESC';
```

---

5.25.1.2.3  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

© 1997-2021 Devart. All Rights Reserved.

5.25.1.2.4 LocalConstraints Property

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Class

TMemDataSet

Syntax

```property```

LocalConstraints: boolean default True;
```

Remarks

Use the LocalConstraints property to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet. When LocalConstraints is True, TMemDataSet ignores NOT NULL server constraints. It is useful for tables that have fields updated by triggers.

LocalConstraints is obsolete, and is only included for backward compatibility.
The default value is True.

5.25.1.1.2.5 LocalUpdate Property

Used to prevent implicit update of rows on database server.

Class

TMemDataSet

Syntax

```property LocalUpdate: boolean default False;```

Remarks

Set the LocalUpdate property to True to prevent implicit update of rows on database server. Data changes are cached locally in client memory.

5.25.1.1.2.6 Prepared Property

Determines whether a query is prepared for execution or not.

Class

TMemDataSet

Syntax

```property Prepared: boolean;```

Remarks

Determines whether a query is prepared for execution or not.

See Also

- Prepare
5.25.1.1.2.7  Ranged Property

Indicates whether a range is applied to a dataset.

Class
TMemDataSet

Syntax

```delphi
property Ranged: Boolean;
```

Remarks
Use the Ranged property to detect whether a range is applied to a dataset.

See Also
- SetRange
- SetRangeEnd
- SetRangeStart

5.25.1.1.2.8  UpdateRecordTypes Property

Used to indicate the update status for the current record when cached updates are enabled.

Class
TMemDataSet

Syntax

```delphi
property UpdateRecordTypes: TUpdateRecordTypes default [rtModified, rtInserted, rtUnmodified];
```

Remarks
Use the UpdateRecordTypes property to determine the update status for the current record when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateRecordTypes offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the
update status of records.

See Also
• CachedUpdates

Class
TMemDataSet

Syntax

```property
UpdatesPending: boolean;
```

Remarks
Use the UpdatesPending property to check the status of the cached updates buffer. If UpdatesPending is True, then there are edited, deleted, or inserted records remaining in local cache and not yet applied to the database. If UpdatesPending is False, there are no such records in the cache.

See Also
• CachedUpdates

Methods

Methods of the TMemDataSet class.

For a complete list of the TMemDataSet class members, see the TMemDataSet Members topic.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong></td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>Locate</strong></td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong></td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td>Cancels changes made to the current record when</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

See Also
- **TMemDataSet Class**
- **TMemDataSet Class Members**
5.25.1.1.3.1 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

© 1997-2021 Devart. All Rights Reserved.

5.25.1.1.3.2 ApplyUpdates Method

Writes dataset's pending cached updates to a database.

Class

TMemDataSet

Overload List

© 2021 Devart
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>ApplyUpdates(const UpdateRecKinds: TUpdateRecKinds)</strong></td>
<td>Writes dataset's pending cached updates of specified records to a database.</td>
</tr>
</tbody>
</table>

### Remarks

Call the `ApplyUpdates` method to write a dataset's pending cached updates to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's `Commit` method to commit the changes to the database if the write is successful, or call the database's `Rollback` method to undo the changes if there is an error.

Following a successful write to the database, and following a successful call to a connection's `Commit` method, an application should call the `CommitUpdates` method to clear the cached update buffer.

**Note:** The preferred method for updating datasets is to call a connection component's `ApplyUpdates` method rather than to call each individual dataset's `ApplyUpdates` method. The connection component's `ApplyUpdates` method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

### See Also

- [TMemDataSet.CachedUpdates](#)
- [TMemDataSet.CancelUpdates](#)
- [TMemDataSet.CommitUpdates](#)
- [TMemDataSet.UpdateStatus](#)
Writes dataset's pending cached updates of specified records to a database.

Class

TMemDataSet

Syntax

```pascal
procedure ApplyUpdates(const UpdateRecKinds: TUpdateRecKinds);
```

Parameters

UpdateRecKinds

Indicates records for which the ApplyUpdates method will be performed.

Remarks

Call the ApplyUpdates method to write a dataset's pending cached updates of specified records to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error.

Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.

Note: The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.
5.25.1.1.3.3 CancelRange Method

Removes any ranges currently in effect for a dataset.

Class

TMemDataSet

Syntax

procedure CancelRange;

Remarks

Call CancelRange to remove a range currently applied to a dataset. Canceling a range reenables access to all records in the dataset.

See Also

- ApplyRange
- EditRangeEnd
- EditRangeStart
- IndexFieldNames
- SetRange
- SetRangeEnd
- SetRangeStart

5.25.1.1.3.4 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

Clears the cached updates buffer.

Class
TMemDataSet

Syntax

```plaintext
procedure CommitUpdates;
```

Remarks

Call the CommitUpdates method to clear the cached updates buffer after both a successful call to ApplyUpdates and a database component’s Commit method. Clearing the cache after applying updates ensures that the cache is empty except for records that could not be processed and were skipped by the OnUpdateRecord or OnUpdateError event handlers. An application can attempt to modify the records still in cache.

CommitUpdates also checks whether there are pending updates in dataset. And if there are, it calls ApplyUpdates.

Record modifications made after a call to CommitUpdates repopulate the cached update buffer and require a subsequent call to ApplyUpdates to move them to the database.

See Also
- CachedUpdates
5.25.1.1.3.6  DeferredPost Method

Makes permanent changes to the database server.

Class

TMemDataSet

Syntax

procedure DeferredPost;

Remarks

Call DeferredPost to make permanent changes to the database server while retaining dataset in its state whether it is dsEdit or dsInsert.

Explicit call to the Cancel method after DeferredPost has been applied does not abandon modifications to a dataset already fixed in database.

5.25.1.1.3.7  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**

Enables changing the starting value for an existing range.

Class

**TMemDataSet**

Syntax

```delphi
procedure EditRangeStart;
```

Remarks

Call **EditRangeStart** to change the starting value for an existing range.

To specify a start range value, call **FieldName** after calling **EditRangeStart**.

After assigning a new ending value, call **ApplyRange** to activate the modified range.

See Also
- **ApplyRange**
- **CancelRange**
- **EditRangeEnd**
- **IndexFieldNames**
- **SetRange**
- **SetRangeEnd**
- **SetRangeStart**
5.25.1.3.9 GetBlob Method

Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Class

TMemDataSet

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBlob(Field: TField)</td>
<td>Retrieves TBlob object for a field or current record when the field itself is known.</td>
</tr>
<tr>
<td>GetBlob(const FieldName: string)</td>
<td>Retrieves TBlob object for a field or current record when its name is known.</td>
</tr>
</tbody>
</table>

Function

function GetBlob(Field: TField): TBlob; overload;

Parameters

Field
    Holds an existing TField object.

Return Value
    TBlob object that was retrieved.

Remarks

Call the GetBlob method to retrieve TBlob object for a field or current record when only its name or the field itself is known. FieldName is the name of an existing field. The field should
have MEMO or BLOB type.

Retrieves TBlob object for a field or current record when its name is known.

Class

**TMemDataSet**

Syntax

```delphi
function GetBlob(const FieldName: string): TBlob; overload;
```

**Parameters**

- **FieldName**
  - Holds the name of an existing field.

**Return Value**

- TBlob object that was retrieved.

**Example**

```
IBCQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
```

**See Also**

- **TBlob**

5.25.1.1.3.10  Locate Method

Searches a dataset for a specific record and positions the cursor on it.

Class

**TMemDataSet**

**Overload List**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locate(const KeyFields: array of TField; const KeyValues: variant; Options:</td>
<td>Searches a dataset by the specified fields for a specific record and positions cursor</td>
</tr>
</tbody>
</table>
Locate(const KeyFields: string; const KeyValues: variant; Options: TLocateOptions)

Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

Class
TMemDataSet

Syntax

function Locate(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateOptions): boolean; reintroduce; overload;

Parameters

KeyFields
Holds TField objects in which to search.

KeyValues
Holds the variant that specifies the values to match in the key fields.

Options
Holds additional search latitude when searching in string fields.

Return Value
True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

Class
TMemDataSet

Syntax
function Locate(const KeyFields: string; const KeyValues: variant; Options: TLocateOptions): boolean; overload; override;

**Parameters**

- **KeyFields**
  - Holds a semicolon-delimited list of field names in which to search.

- **KeyValues**
  - Holds the variant that specifies the values to match in the key fields.

- **Options**
  - Holds additional search latitude when searching in string fields.

**Return Value**

True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

**Remarks**

Call the Locate method to search a dataset for a specific record and position cursor on it.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. An example is provided below.

Options is a set that optionally specifies additional search latitude when searching in string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

Locate returns True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

The Locate function works faster when dataset is locally sorted on the KeyFields fields. Local dataset sorting can be set with the TMemDataSet.IndexFieldNames property.

**Example**

An example of specifying multiple search values:

```pascal
with CustTable do
  Locate(['Company;Contact;Phone'], VarArrayOf(['Sight Diver', 'P', '408-431-1000']), [loPartialKey]);
```
5.25.1.3.11 LocateEx Method

Excludes features that don't need to be included to the `TMemDataSet.Locate` method of TDataSet.

Class

`TMemDataSet`

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LocateEx(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateExOptions)</code></td>
<td>Excludes features that don't need to be included to the <code>TMemDataSet.Locate</code> method of TDataSet by the specified fields.</td>
</tr>
<tr>
<td><code>LocateEx(const KeyFields: string; const KeyValues: variant; Options: TLocateExOptions)</code></td>
<td>Excludes features that don't need to be included to the <code>TMemDataSet.Locate</code> method of TDataSet by the specified field names.</td>
</tr>
</tbody>
</table>

Parameters

```pascal
function LocateEx(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateExOptions): boolean; overload;
```
KeyFields
   Holds TField objects to search in.

KeyValues
   Holds the values of the fields to search for.

Options
   Holds additional search parameters which will be used by the LocateEx method.

Return Value
   True, if a matching record was found. Otherwise returns False.

Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified field names.

Class
   TMemDataSet

Syntax

function LocateEx(const KeyFields: string; const KeyValues: variant; Options: TLocateExOptions): boolean; overload;

Parameters

KeyFields
   Holds the fields to search in.

KeyValues
   Holds the values of the fields to search for.

Options
   Holds additional search parameters which will be used by the LocateEx method.

Return Value
   True, if a matching record was found. Otherwise returns False.

Remarks

Call the LocateEx method when you need some features not to be included to the TMemDataSet.Locate method of TDataSet.

LocateEx returns True if it finds a matching record, and makes that record the current one. Otherwise LocateEx returns False.

The LocateEx function works faster when dataset is locally sorted on the KeyFields fields.
Local dataset sorting can be set with the `TMemDataSet.IndexFieldNames` property.

**Note:** Please add the MemData unit to the "uses" list to use the TLocalExOption enumeration.

See Also

- `TMemDataSet.IndexFieldNames`
- `TMemDataSet.Locate`

© 1997-2021 Devart. All Rights Reserved.

Reprinted by permission of Delphi.
5.25.1.1.3.13 RestoreUpdates Method

Marks all records in the cache of updates as unapplied.

Class

TMemDataSet

Syntax

procedure RestoreUpdates;

Remarks

Call the RestoreUpdates method to return the cache of updates to its state before calling ApplyUpdates. RestoreUpdates marks all records in the cache of updates as unapplied. It is useful when ApplyUpdates fails.

See Also

- CachedUpdates
- TMemDataSet.ApplyUpdates
- CancelUpdates
- UpdateStatus

5.25.1.1.3.14 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

© 1997-2021 Devart. All Rights Reserved.

5.25.1.3.15  SaveToXML Method

Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Class
TMemDataSet

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SaveToXML(Destination: TStream)</code></td>
<td>Saves the current dataset data to a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><code>SaveToXML(const FileName: string)</code></td>
<td>Saves the current dataset data to a file in the XML format compatible with ADO format.</td>
</tr>
</tbody>
</table>

Saves the current dataset data to a stream in the XML format compatible with ADO format.

Class
TMemDataSet

Syntax

```pascal
procedure SaveToXML(Destination: TStream); overload;
```

Parameters

*Destination*
Holds a TStream object.
Remarks

Call the SaveToXML method to save the current dataset data to a file or a stream in the XML format compatible with ADO format.

If the destination file already exists, it is overwritten. It remains open from the first call to SaveToXML until the dataset is closed. This file can be read by other applications while it is opened, but they cannot write to the file.

When saving data to a stream, a TStream object must be created and its position must be set in a preferable value.

See Also

- TVirtualTable.LoadFromFile
- TVirtualTable.LoadFromStream

Syntax

```pascal
procedure SaveToXML(const FileName: string); overload;
```

Parameters

- **FileName**: Holds the name of a destination file.

See Also

- TVirtualTable.LoadFromFile
- TVirtualTable.LoadFromStream
Sets the starting and ending values of a range, and applies it.

**Class**

`TMemDataSet`

**Syntax**

```pascal
procedure SetRange(const StartValues: array of System.TVarRec;
const EndValues: array of System.TVarRec; StartExclusive: Boolean = False;
EndExclusive: Boolean = False);
```

**Parameters**

- **StartValues**
  Indicates the field values that designate the first record in the range. In C++, `StartValues_Size` is the index of the last value in the `StartValues` array.

- **EndValues**
  Indicates the field values that designate the last record in the range. In C++, `EndValues_Size` is the index of the last value in the `EndValues` array.

- **StartExclusive**
  Indicates the upper and lower boundaries of the start range.

- **EndExclusive**
  Indicates the upper and lower boundaries of the end range.

**Remarks**

Call `SetRange` to specify a range and apply it to the dataset. The new range replaces the currently specified range, if any.

`SetRange` combines the functionality of `SetRangeStart`, `SetRangeEnd`, and `ApplyRange` in a single procedure call. `SetRange` performs the following functions:

1. Puts the dataset into `dsSetKey` state.
2. Erases any previously specified starting range values and ending range values.
3. Sets the start and end range values.
4. Applies the range to the dataset.

After a call to `SetRange`, the cursor is left on the first record in the range.

If either `StartValues` or `EndValues` has fewer elements than the number of fields in the current index, then the remaining entries are ignored when performing a search.

**See Also**
Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

**Class**

`TMemDataSet`

**Syntax**

```plaintext
procedure SetRangeEnd;
```

**Remarks**

Call `SetRangeEnd` to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL.

Subsequent field assignments made with `FieldByName` specify the actual set of ending values for a range.

After assigning end-range values, call `ApplyRange` to activate the modified range.

**See Also**

- `ApplyRange`
- `CancelRange`
- `EditRangeStart`
- `IndexFieldNames`
- `SetRange`
- `SetRangeStart`
5.25.1.1.3.18 SetRangeStart Method

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

Class

TMemDataSet

Syntax

procedure SetRangeStart;

Remarks

Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL.

Subsequent field assignments to FieldByName specify the actual set of starting values for a range.

After assigning start-range values, call **ApplyRange** to activate the modified range.

See Also

- **ApplyRange**
- **CancelRange**
- **EditRangeStart**
- **IndexFieldNames**
- **SetRange**
- **SetRangeEnd**

5.25.1.1.3.19 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class
**TMemDataSet**

**Syntax**

```delphi
procedure UnPrepare; virtual;
```

**Remarks**

Call the UnPrepare method to free the resources allocated for a previously prepared query on the server and client sides.

**Note:** When you change the text of a query at runtime, the query is automatically closed and unprepared.

**See Also**

- [Prepare](#)

---

**5.25.1.1.3.20 UpdateResult Method**

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**Class**

**TMemDataSet**

**Syntax**

```delphi
function UpdateResult: TUpdateAction;
```

**Return Value**

A value of the TUpdateAction enumeration.

**Remarks**

Call the UpdateResult method to read the status of the latest call to the ApplyUpdates method while cached updates are enabled. UpdateResult reflects updates made on the records that have been edited, inserted, or deleted.

UpdateResult works on the record by record basis and is applicable to the current record only.
5.25.1.1.3.21 UpdateStatus Method

Indicates the current update status for the dataset when cached updates are enabled.

Class

TMemDataSet

Syntax

function UpdateStatus: TUpdateStatus; override;

Return Value

a value of the TUpdateStatus enumeration.

Remarks

Call the UpdateStatus method to determine the current update status for the dataset when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateStatus offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of the dataset.

See Also

• CachedUpdates

© 1997-2021

Devart. All Rights Reserved.

5.25.1.1.4 Events

Events of the TMemDataSet class.

For a complete list of the TMemDataSet class members, see the TMemDataSet Members topic.
### Name | Description
--- | ---
OnUpdateError | Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord | Occurs when a single update component can not handle the updates.

### See Also
- [TMemDataSet Class](#)
- [TMemDataSet Class Members](#)

### 5.25.1.4.1 OnUpdateError Event

Occurs when an exception is generated while cached updates are applied to a database.

### Class

**TMemDataSet**

### Syntax

```
property OnUpdateError: TUpdateErrorEvent;
```

### Remarks

Write the OnUpdateError event handler to respond to exceptions generated when cached updates are applied to a database.

E is a pointer to an EDatabaseError object from which application can extract an error message and the actual cause of the error condition. The OnUpdateError handler can use this information to determine how to respond to the error condition.

UpdateKind describes the type of update that generated the error.

UpdateAction indicates the action to take when the OnUpdateError handler exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateError can handle or correct the error, set UpdateAction to uaRetry before exiting the error handler.

The error handler can use the TField.OldValue and TField.NewValue properties to evaluate...
error conditions and set TField.NewValue to a new value to reapply. In this case, set UpdateAction to uaRetry before exiting.

**Note:** If a call to ApplyUpdates raises an exception and ApplyUpdates is not called within the context of a try...except block, an error message is displayed. If the OnUpdateError handler cannot correct the error condition and leaves UpdateAction set to uaFail, the error message is displayed twice. To prevent redisplay, set UpdateAction to uaAbort in the error handler.

### See Also
- [CachedUpdates](#)

---

#### 5.25.1.1.4.2  OnUpdateRecord Event

Occurs when a single update component can not handle the updates.

**Class**

TMemDataSet

**Syntax**

```plaintext
property OnUpdateRecord: TUpdateRecordEvent;
```

**Remarks**

Write the OnUpdateRecord event handler to process updates that cannot be handled by a single update component, such as implementation of cascading updates, insertions, or deletions. This handler is also useful for applications that require additional control over parameter substitution in update components.

UpdateKind describes the type of update to perform.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful, it should set UpdateAction to uaApplied before exiting.

### See Also
- [CachedUpdates](#)
5.25.2 Variables

Variables in the MemDS unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoNotRaiseExcetionOnUaFail</td>
<td>An exception will be raised if the value of the UpdateAction parameter is uaFail.</td>
</tr>
<tr>
<td>SendDataSetChangeEventAfterOpen</td>
<td>The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.</td>
</tr>
</tbody>
</table>

5.25.2.1 DoNotRaiseExcetionOnUaFail Variable

An exception will be raised if the value of the UpdateAction parameter is uaFail.

Unit

MemDS

Syntax

DoNotRaiseExcetionOnUaFail: boolean = False;

Remarks

Starting with IBDAC 2.20.0.12, if the OnUpdateRecord event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged.

To restore the old behaviour, set DoNotRaiseExcetionOnUaFail to True.
5.25.2.2 SendDataSetChangeEventAfterOpen Variable

The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

Unit

MemDS

Syntax

SendDataSetChangeEventAfterOpen: boolean = True;

Remarks

Starting with IBDAC 2.20.0.11, the DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled.

To disable sending this event, change the value of this variable to False.

© 1997-2021 Devart. All Rights Reserved.

5.26 VirtualDataSet

This unit contains implementation of the TVirtualDataSet component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomVirtualDataSet</td>
<td>A base class for representation of arbitrary data in tabular form.</td>
</tr>
<tr>
<td>TVirtualDataSet</td>
<td>Dataset that processes arbitrary non-tabular data.</td>
</tr>
</tbody>
</table>

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOnDeleteRecordEvent</td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.</td>
</tr>
</tbody>
</table>
### Classes

Classes in the `VirtualDataSet` unit.

#### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TCustomVirtualDataSet</code></td>
<td>A base class for representation of arbitrary data in tabular form.</td>
</tr>
<tr>
<td><code>TVirtualDataSet</code></td>
<td>Dataset that processes arbitrary non-tabular data.</td>
</tr>
</tbody>
</table>

#### `TCustomVirtualDataSet` Class

A base class for representation of arbitrary data in tabular form.

For a list of all members of this type, see `TCustomVirtualDataSet` members.

**Unit**

`VirtualDataSet`

**Syntax**
TCustomVirtualDataSet = class(TMemDataSet);

Inheritance Hierarchy

TMemDataSet
   TCustomVirtualDataSet

© 1997-2021 Devart. All Rights Reserved.

Members

**TCustomVirtualDataSet** class overview.

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from TMemDataSet) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from TMemDataSet) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>(inherited from TMemDataSet) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>(inherited from TMemDataSet) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from TMemDataSet) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from TMemDataSet) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from TMemDataSet) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong></td>
<td>(inherited from TMemDataSet) Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong></td>
<td>(inherited from TMemDataSet) Used to check the status of</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>Locate</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Overloaded. Excludes features that don't need to be included to the <code>TMemDataSet.Locate</code> method of <code>TDataSet</code>.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached</td>
</tr>
</tbody>
</table>
updates are applied to a database.

**OnUpdateRecord** (inherited from **TMemDataSet**)  
Occurs when a single update component can not handle the updates.

---

### 5.26.1.2 TVirtualDataSet Class

Dataset that processes arbitrary non-tabular data.

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

**Unit**

`VirtualDataSet`

**Syntax**

```delphi
TVirtualDataSet = class(TCustomVirtualDataSet);
```

**Inheritance Hierarchy**

- **TMemDataSet**
  - **TCustomVirtualDataSet**
    - **TVirtualDataSet**

---

#### 5.26.1.2.1 Members

**TVirtualDataSet** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to get or set the list of fields on which the recordset</td>
</tr>
</tbody>
</table>
### KeyExclusive (inherited from TMemDataSet)
Specifies the upper and lower boundaries for a range.

### LocalConstraints (inherited from TMemDataSet)
Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

### LocalUpdate (inherited from TMemDataSet)
Used to prevent implicit update of rows on database server.

### Prepared (inherited from TMemDataSet)
Determines whether a query is prepared for execution or not.

### Ranged (inherited from TMemDataSet)
Indicates whether a range is applied to a dataset.

### UpdateRecordTypes (inherited from TMemDataSet)
Used to indicate the update status for the current record when cached updates are enabled.

### UpdatesPending (inherited from TMemDataSet)
Used to check the status of the cached updates buffer.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong></td>
<td>(inherited from TMemDataSet) Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>(inherited from TMemDataSet) Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>(inherited from TMemDataSet) Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>(inherited from TMemDataSet) Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>(inherited from TMemDataSet) Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>(inherited from TMemDataSet) Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Excludes features that don't need to be included to the <strong>TMemDataSet.Locate</strong> method of <strong>TDataSet</strong>.</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
</tbody>
</table>
specify the start of the range of rows to include in the dataset.

**UnPrepare** (inherited from **TMemDataSet**) Frees the resources allocated for a previously prepared query on the server and client sides.

**UpdateResult** (inherited from **TMemDataSet**) Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**UpdateStatus** (inherited from **TMemDataSet**) Indicates the current update status for the dataset when cached updates are enabled.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnUpdateError</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td><strong>OnUpdateRecord</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

### Types

Types in the **VirtualDataSet** unit.

**Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TOnDeleteRecordEvent</strong></td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.</td>
</tr>
<tr>
<td><strong>TOnGetFieldValueEvent</strong></td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet event.</td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TOnGetRecordCountEvent</td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.</td>
</tr>
<tr>
<td>TOnModifyRecordEvent</td>
<td>This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.</td>
</tr>
</tbody>
</table>

**5.26.2.1 TOnDeleteRecordEvent Procedure Reference**

This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.

**Unit**

`virtualDataSet`

**Syntax**

```delphi
TOnDeleteRecordEvent = procedure (Sender: TObject; RecNo: Integer) of object;
```

**Parameters**

- **Sender**
  - An object that raised the event.
- **RecNo**
  - Number of the record being deleted.

**5.26.2.2 TOnGetFieldValueEvent Procedure Reference**

This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.

**Unit**

`virtualDataSet`

**Syntax**
### TOnGetFieldValueEvent

**Syntax**

```pascal
TOnGetFieldValueEvent = procedure (Sender: TObject; Field: TField; RecNo: Integer; out Value: Variant) of object;
```

**Parameters**

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

- **Field**
  - The field, which data has to be returned.

- **RecNo**
  - The number of the record, which data has to be returned.

- **Value**
  - Requested field value.

---

### TOnGetRecordCountEvent

**Procedure Reference**

This type is used for the `E:Devart.Dac.TVirtualDataSet.OnGetRecordCount` event.

**Unit**

`VirtualDataSet`

**Syntax**

```pascal
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.

---

### TOnModifyRecordEvent

**Procedure Reference**

This type is used for `E:Devart.Dac.TVirtualDataSet.OnInsertRecord` and `E:Devart.Dac.TVirtualDataSet.OnModifyRecord` events.

**Unit**

`VirtualDataSet`
**VirtualDataSet**

**Syntax**

```plaintext
TOnModifyRecordEvent = procedure (Sender: TObject; var RecNo: Integer) of object;
```

**Parameters**

*Sender*

An object that raised the event.

*RecNo*

Number of the record being inserted or modified.

---

**5.27 VirtualTable**

This unit contains implementation of the TVirtualTable component.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVirtualTable</td>
<td>Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.</td>
</tr>
</tbody>
</table>

---

**5.27.1 Classes**

Classes in the **VirtualTable** unit.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVirtualTable</td>
<td>Dataset that stores data in memory. This component is placed on the Data Access page of the Component</td>
</tr>
</tbody>
</table>
5.27.1.1 TVirtualTable Class

Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

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

**Unit**

virtualTable

**Syntax**

```
TVirtualTable = class(TMemDataSet);
```

**Inheritance Hierarchy**

TMemDataSet

TVirtualTable

**5.27.1.1.1 Members**

**TVirtualTable** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields.</td>
</tr>
<tr>
<td>IndexFieldNames (inherited from TMemDataSet)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
</tbody>
</table>
### KeyExclusive (inherited from `TMemDataSet`)
Specifies the upper and lower boundaries for a range.

### LocalConstraints (inherited from `TMemDataSet`)
Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

### LocalUpdate (inherited from `TMemDataSet`)
Used to prevent implicit update of rows on database server.

### Prepared (inherited from `TMemDataSet`)
Determines whether a query is prepared for execution or not.

### Ranged (inherited from `TMemDataSet`)
Indicates whether a range is applied to a dataset.

### UpdateRecordTypes (inherited from `TMemDataSet`)
Used to indicate the update status for the current record when cached updates are enabled.

### UpdatesPending (inherited from `TMemDataSet`)
Used to check the status of the cached updates buffer.

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>Assign</strong></td>
<td>Copies fields and data from another TDataSet component.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>GetBlob</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td></td>
</tr>
<tr>
<td>LoadFromStream</td>
<td></td>
</tr>
<tr>
<td>Locate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocateEx</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Prepare</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SetRange</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnUpdateError</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td><strong>OnUpdateRecord</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

Properties of the **TVirtualTable** class.

For a complete list of the **TVirtualTable** class members, see the **TVirtualTable Members**
Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong> (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong> (inherited from TMemDataSet)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from TMemDataSet)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong> (inherited from TMemDataSet)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong> (inherited from TMemDataSet)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from TMemDataSet)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from TMemDataSet)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong> (inherited from TMemDataSet)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong> (inherited from TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DefaultSortType</strong></td>
<td>Used to determine the default type of local sorting for string fields.</td>
</tr>
</tbody>
</table>

See Also

- TVirtualTable Class
5.27.1.1.2.1 DefaultSortType Property

Used to determine the default type of local sorting for string fields.

Class

TVirtualTable

Syntax

```delphi
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.

Public

Methods of the **TVirtualTable** class.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>Assign</strong></td>
<td>Copies fields and data from another TDataSet component.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Removes any ranges currently in effect for a</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Loads data from a file into a TVirtualTable component.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Copies data from a stream into a TVirtualTable component.</td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong></td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong></td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong></td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong></td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

**See Also**
- TVirtualTable Class
- TVirtualTable Class Members

© 1997-2021 Devart. All Rights Reserved.

5.27.1.1.3.1 Assign Method

Copies fields and data from another TDataSet component.

Class
**TVirtualTable**

**Syntax**

```delphi
procedure Assign(Source: TPersistent); override;
```

**Parameters**

**Source**

Holds the TDataSet component to copy fields and data from.

**Remarks**

Call the Assign method to copy fields and data from another TDataSet component.

**Note:** Unsupported field types are skipped (i.e. destination dataset will contain less fields than the source one). This may happen when Source is not a TVirtualTable component but some server-oriented dataset.

**Example**

```delphi
Query1.SQL.Text := 'SELECT * FROM DEPT';
Query1.Active := True;
VirtualTable1.Assign(Query1);
VirtualTable1.Active := True;
```

**See Also**

- TVirtualTable

---

**5.27.1.1.3.2 LoadFromFile Method**

Loads data from a file into a TVirtualTable component.

**Class**

**TVirtualTable**

**Syntax**

```delphi
procedure LoadFromFile(const FileName: string; LoadFields: boolean = True; DecodeHTMLEntities: boolean = True);
```

**Parameters**

**FileName**

Holds the name of the file to load data from.
LoadFields
- Indicates whether to load fields from the file.

DecodeHTMLEntities
- Indicates whether to decode HTML entities from the file.

Remarks
Call the LoadFromFile method to load data from a file into a TVirtualTable component. Specify the name of the file to load into the field as the value of the FileName parameter. This file may be an XML document in ADO-compatible format or in virtual table data format. The file format is detected automatically.

5.27.1.3.3 LoadFromStream Method

Copies data from a stream into a TVirtualTable component.

Class
TVirtualTable

Syntax

```pascal
procedure LoadFromStream(Stream: TStream; LoadFields: boolean = True; DecodeHTMLEntities: boolean = True);
```

Parameters

Stream
- Holds the stream from which the field's value is copied.

LoadFields
- Indicates whether to load fields from the stream.

DecodeHTMLEntities
- Indicates whether to decode HTML entities from the stream.

Remarks
Call the LoadFromStream method to copy data from a stream into a TVirtualTable component. Specify the stream from which the field's value is copied as the value of the Stream parameter. Data in the stream may be in ADO-compatible format or in virtual table data format. The data format is detected automatically.
Reserved.