# Table of Contents

Part I  What's New ................................................................. 1

Part II  General Information .................................................. 6

  1  Overview ........................................................................ 6
  2  Features ......................................................................... 8
  3  Requirements .................................................................. 9
  4  Compatibility ................................................................... 9
  5  Using Several DAC Products in One IDE ...................... 14
  6  Component List ............................................................. 14
  7  Hierarchy Chart ............................................................. 15
  8  Editions ......................................................................... 15
  9  Licensing ....................................................................... 17
 10  Getting Support ............................................................ 20

Part III  Getting Started ........................................................ 21

  1  Installation ..................................................................... 21
  2  Deployment ..................................................................... 23

Part IV  Reference ................................................................. 24

  1  DALoader ........................................................................ 26
     Classes ......................................................................... 26
       TDALoaderOptions Class ............................................... 26
         Members ..................................................................... 26
         Properties .................................................................... 27
       UseBlankValues Property ........................................... 27
  2  DASQLMonitor ............................................................... 28
     Classes ......................................................................... 28
       TCustomDASQLMonitor Class ....................................... 28
         Members ..................................................................... 29
         Properties .................................................................... 29
         Active Property ......................................................... 30
         DBMonitorOptions Property .................................... 30
         Options Property ....................................................... 31
         TraceFlags Property .................................................. 31
       Events ......................................................................... 32
       OnSQL Event .............................................................. 32
       TDBMonitorOptions Class ........................................... 33
         Members ..................................................................... 33
         Properties .................................................................... 34
         Host Property ............................................................ 34
         Port Property ............................................................. 35
         ReconnectTimeout Property .................................... 35
         SendTimeout Property ............................................... 36
### Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlags Set</td>
<td>36</td>
</tr>
<tr>
<td>TMonitorOptions Set</td>
<td>37</td>
</tr>
<tr>
<td>TOnSQLEvent Procedure Reference</td>
<td>37</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlag Enumeration</td>
<td>38</td>
</tr>
<tr>
<td>TMonitorOption Enumeration</td>
<td>39</td>
</tr>
</tbody>
</table>

### Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomDAConnection Class</td>
<td>40</td>
</tr>
<tr>
<td>Members</td>
<td>41</td>
</tr>
<tr>
<td>Properties</td>
<td>42</td>
</tr>
<tr>
<td>Component Property</td>
<td>43</td>
</tr>
<tr>
<td>ErrorCode Property</td>
<td>43</td>
</tr>
<tr>
<td>TCustomConnectDialog Class</td>
<td>43</td>
</tr>
<tr>
<td>Members</td>
<td>44</td>
</tr>
<tr>
<td>Properties</td>
<td>45</td>
</tr>
<tr>
<td>CancelButton Property</td>
<td>46</td>
</tr>
<tr>
<td>Caption Property</td>
<td>47</td>
</tr>
<tr>
<td>ConnectButton Property</td>
<td>47</td>
</tr>
<tr>
<td>DialogClass Property</td>
<td>47</td>
</tr>
<tr>
<td>LabelSet Property</td>
<td>48</td>
</tr>
<tr>
<td>Passw ordLabel Property</td>
<td>48</td>
</tr>
<tr>
<td>Retries Property</td>
<td>49</td>
</tr>
<tr>
<td>SavePassw ord Property</td>
<td>49</td>
</tr>
<tr>
<td>ServerLabel Property</td>
<td>50</td>
</tr>
<tr>
<td>StoreLogInfo Property</td>
<td>50</td>
</tr>
<tr>
<td>UsernameLabel Property</td>
<td>51</td>
</tr>
<tr>
<td>Methods</td>
<td>51</td>
</tr>
<tr>
<td>Execute Method</td>
<td>51</td>
</tr>
<tr>
<td>GetServerList Method</td>
<td>52</td>
</tr>
<tr>
<td>TCustomDAConnection Class</td>
<td>53</td>
</tr>
<tr>
<td>Members</td>
<td>53</td>
</tr>
<tr>
<td>Properties</td>
<td>55</td>
</tr>
<tr>
<td>ConnectDialog Property</td>
<td>56</td>
</tr>
<tr>
<td>ConnectString Property</td>
<td>57</td>
</tr>
<tr>
<td>ConvertEOL Property</td>
<td>58</td>
</tr>
<tr>
<td>InTransaction Property</td>
<td>58</td>
</tr>
<tr>
<td>LoginPrompt Property</td>
<td>59</td>
</tr>
<tr>
<td>Options Property</td>
<td>59</td>
</tr>
<tr>
<td>Passw ord Property</td>
<td>60</td>
</tr>
<tr>
<td>Pooling Property</td>
<td>61</td>
</tr>
<tr>
<td>PoolingOptions Property</td>
<td>62</td>
</tr>
<tr>
<td>Server Property</td>
<td>63</td>
</tr>
<tr>
<td>Username Property</td>
<td>63</td>
</tr>
<tr>
<td>Methods</td>
<td>64</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>65</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>66</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>66</td>
</tr>
<tr>
<td>Commit Method</td>
<td>67</td>
</tr>
<tr>
<td>Connect Method</td>
<td>67</td>
</tr>
<tr>
<td>CreateSQL Method</td>
<td>68</td>
</tr>
<tr>
<td>Disconnect Method</td>
<td>69</td>
</tr>
<tr>
<td>ExecProc Method</td>
<td>69</td>
</tr>
<tr>
<td>Method/Property</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ExecProcEx Method</td>
<td>71</td>
</tr>
<tr>
<td>ExecSql Method</td>
<td>72</td>
</tr>
<tr>
<td>ExecSqlEx Method</td>
<td>73</td>
</tr>
<tr>
<td>GetDatabaseNames Method</td>
<td>74</td>
</tr>
<tr>
<td>GetKeyFieldNames Method</td>
<td>75</td>
</tr>
<tr>
<td>GetStoredProcNames Method</td>
<td>75</td>
</tr>
<tr>
<td>GetTableNames Method</td>
<td>76</td>
</tr>
<tr>
<td>MonitorMessage Method</td>
<td>77</td>
</tr>
<tr>
<td>Ping Method</td>
<td>78</td>
</tr>
<tr>
<td>RemoveFromPool Method</td>
<td>78</td>
</tr>
<tr>
<td>Rollback Method</td>
<td>79</td>
</tr>
<tr>
<td>StartTransaction Method</td>
<td>79</td>
</tr>
<tr>
<td>ExecSQLEx Method</td>
<td>80</td>
</tr>
<tr>
<td>TCustomDataClass</td>
<td>81</td>
</tr>
<tr>
<td>Members</td>
<td>82</td>
</tr>
<tr>
<td>Events</td>
<td>83</td>
</tr>
<tr>
<td>OnConnectionLost Event</td>
<td>84</td>
</tr>
<tr>
<td>Params Check Property</td>
<td>85</td>
</tr>
<tr>
<td>Options Property</td>
<td>86</td>
</tr>
<tr>
<td>MasterSource Property</td>
<td>87</td>
</tr>
<tr>
<td>MasterFields Property</td>
<td>88</td>
</tr>
<tr>
<td>FetchRow’s Property</td>
<td>89</td>
</tr>
<tr>
<td>FilterSQL Property</td>
<td>90</td>
</tr>
<tr>
<td>FinalSQL Property</td>
<td>91</td>
</tr>
<tr>
<td>IsQuery Property</td>
<td>92</td>
</tr>
<tr>
<td>KeyFields Property</td>
<td>93</td>
</tr>
<tr>
<td>MacroCount Property</td>
<td>94</td>
</tr>
<tr>
<td>Macros Property</td>
<td>95</td>
</tr>
<tr>
<td>MasterFields Property</td>
<td>96</td>
</tr>
<tr>
<td>Options Property</td>
<td>97</td>
</tr>
<tr>
<td>ParamCheck Property</td>
<td>98</td>
</tr>
<tr>
<td>ParamCount Property</td>
<td>99</td>
</tr>
<tr>
<td>Params Property</td>
<td>100</td>
</tr>
<tr>
<td>RefreshOptions Property</td>
<td>101</td>
</tr>
<tr>
<td>RowAffected Property</td>
<td>102</td>
</tr>
<tr>
<td>SQL Property</td>
<td>103</td>
</tr>
<tr>
<td>SQLDelete Property</td>
<td>104</td>
</tr>
<tr>
<td>SQLInsert Property</td>
<td>105</td>
</tr>
<tr>
<td>SQLLock Property</td>
<td>106</td>
</tr>
<tr>
<td>SQLRecCount Property</td>
<td>107</td>
</tr>
<tr>
<td>SQLRefresh Property</td>
<td>108</td>
</tr>
<tr>
<td>SQLUpdate Property</td>
<td>109</td>
</tr>
<tr>
<td>UniDirectional Property</td>
<td>110</td>
</tr>
<tr>
<td>Methods</td>
<td>111</td>
</tr>
<tr>
<td>AddWhere Method</td>
<td>112</td>
</tr>
<tr>
<td>BreakExec Method</td>
<td>113</td>
</tr>
<tr>
<td>CreateBlobstream Method</td>
<td>114</td>
</tr>
<tr>
<td>DeleteWhere Method</td>
<td>115</td>
</tr>
<tr>
<td>OnError Event</td>
<td>116</td>
</tr>
<tr>
<td>OnLineConnect Method</td>
<td>117</td>
</tr>
<tr>
<td>OnLogConntect Method</td>
<td>118</td>
</tr>
<tr>
<td>OnValidateMessage Method</td>
<td>119</td>
</tr>
<tr>
<td>OnWarning Method</td>
<td>120</td>
</tr>
</tbody>
</table>
VirtualDAC

TCustomDASQL Class

Events

Execute Method ........................................................................................................................................ 121
Execute Method ........................................................................................................................................ 121
Execute Method ........................................................................................................................................ 122
Executing Method ...................................................................................................................................... 122
Fetched Method ......................................................................................................................................... 123
Fetching Method ......................................................................................................................................... 123
Fetching Method ......................................................................................................................................... 123
FetchingAll Method ................................................................................................................................. 124
FindKey Method ......................................................................................................................................... 124
FindMacro Method .................................................................................................................................... 125
FindNearest Method ................................................................................................................................. 125
FindParam Method .................................................................................................................................... 126
GetDataType Method .................................................................................................................................. 127
GetFieldObject Method ........................................................................................................................... 127
GetFieldPrecision Method ....................................................................................................................... 128
GetFieldScale Method ............................................................................................................................. 129
GetKeyFieldNames Method ...................................................................................................................... 129
GetOrderBy Method ................................................................................................................................... 130
GotoCurrent Method ................................................................................................................................ 131
Lock Method ............................................................................................................................................... 131
MacroByName Method .............................................................................................................................. 132
ParamByName Method .............................................................................................................................. 133
Prepare Method .......................................................................................................................................... 134
RefreshRecord Method ............................................................................................................................ 134
RestoreSQL Method .................................................................................................................................. 135
SaveSQL Method ....................................................................................................................................... 135
SetOrderBy Method .................................................................................................................................... 136
SQLSaved Method .................................................................................................................................... 137
UnLock Method ......................................................................................................................................... 137
Events

AfterExecute Event ................................................................................................................................... 138
AfterFetch Event ......................................................................................................................................... 139
AfterUpdateExecute Event ....................................................................................................................... 139
BeforeFetch Event ..................................................................................................................................... 140
BeforeUpdateExecute Event ...................................................................................................................... 140

TCustomDASQL Class

Members ..................................................................................................................................................... 141
Properties .................................................................................................................................................. 143

ChangeCursor Property ........................................................................................................................... 145
Connection Property ................................................................................................................................. 145
Debug Property .......................................................................................................................................... 146
FinalSQL Property ...................................................................................................................................... 146
MacroCount Property ............................................................................................................................... 147
Macros Property ....................................................................................................................................... 147
ParamCheck Property ............................................................................................................................... 148
ParamCount Property ............................................................................................................................... 149
Params Property ......................................................................................................................................... 149
ParamValues Property (Indexer) ............................................................................................................... 150
Prepared Property ..................................................................................................................................... 151
RowsAffected Property ............................................................................................................................. 151
SQL Property ............................................................................................................................................ 152
Methods .................................................................................................................................................... 152

Execute Method ......................................................................................................................................... 153
Execute Method ......................................................................................................................................... 154
Execute Method ......................................................................................................................................... 154

© 2019 Devart
### TDAConnectionOptions Class

<table>
<thead>
<tr>
<th>Members</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>185</td>
</tr>
<tr>
<td>Allow ImplicitConnect Property</td>
<td>187</td>
</tr>
<tr>
<td>DefaultSortType Property</td>
<td>188</td>
</tr>
<tr>
<td>DisconnectedMode Property</td>
<td>188</td>
</tr>
<tr>
<td>KeepDesignConnected Property</td>
<td>189</td>
</tr>
<tr>
<td>LocalFailover Property</td>
<td>189</td>
</tr>
</tbody>
</table>

### TDADataSetOptions Class

<table>
<thead>
<tr>
<th>Members</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>190</td>
</tr>
<tr>
<td>AutoPrepare Property</td>
<td>193</td>
</tr>
<tr>
<td>CacheCalcFields Property</td>
<td>195</td>
</tr>
<tr>
<td>CompressBlobMode Property</td>
<td>196</td>
</tr>
<tr>
<td>DefaultValues Property</td>
<td>196</td>
</tr>
<tr>
<td>DetailDelay Property</td>
<td>197</td>
</tr>
<tr>
<td>FieldsOrigin Property</td>
<td>197</td>
</tr>
<tr>
<td>FlatBuffers Property</td>
<td>198</td>
</tr>
<tr>
<td>InsertAllSetFields Property</td>
<td>199</td>
</tr>
<tr>
<td>LocalMasterDetail Property</td>
<td>199</td>
</tr>
<tr>
<td>LongStrings Property</td>
<td>200</td>
</tr>
<tr>
<td>MasterFieldsNullable Property</td>
<td>200</td>
</tr>
<tr>
<td>NumberRange Property</td>
<td>200</td>
</tr>
<tr>
<td>QueryRecCount Property</td>
<td>201</td>
</tr>
<tr>
<td>QuoteNames Property</td>
<td>201</td>
</tr>
<tr>
<td>RemoveOnRefresh Property</td>
<td>202</td>
</tr>
<tr>
<td>RequiredFields Property</td>
<td>202</td>
</tr>
<tr>
<td>ReturnParams Property</td>
<td>203</td>
</tr>
<tr>
<td>SetFieldsReadOnly Property</td>
<td>203</td>
</tr>
<tr>
<td>StrictUpdate Property</td>
<td>204</td>
</tr>
<tr>
<td>TrimFixedChar Property</td>
<td>204</td>
</tr>
<tr>
<td>UpdateAllFields Property</td>
<td>205</td>
</tr>
<tr>
<td>UpdateBatchSize Property</td>
<td>205</td>
</tr>
</tbody>
</table>

### TDAMapRule Class

<table>
<thead>
<tr>
<th>Members</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>206</td>
</tr>
<tr>
<td>DBLengthMax Property</td>
<td>207</td>
</tr>
<tr>
<td>DBLengthMin Property</td>
<td>208</td>
</tr>
<tr>
<td>DBScaleMax Property</td>
<td>208</td>
</tr>
<tr>
<td>DBScaleMin Property</td>
<td>209</td>
</tr>
<tr>
<td>DBType Property</td>
<td>209</td>
</tr>
<tr>
<td>FieldLength Property</td>
<td>210</td>
</tr>
<tr>
<td>FieldName Property</td>
<td>210</td>
</tr>
<tr>
<td>FieldScale Property</td>
<td>210</td>
</tr>
<tr>
<td>FieldType Property</td>
<td>211</td>
</tr>
<tr>
<td>IgnoreErrors Property</td>
<td>211</td>
</tr>
</tbody>
</table>

### TDAMapRules Class

<table>
<thead>
<tr>
<th>Members</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>212</td>
</tr>
<tr>
<td>IgnoreInvalidRules Property</td>
<td>213</td>
</tr>
</tbody>
</table>

### TDAMetaData Class

<table>
<thead>
<tr>
<th>Members</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>214</td>
</tr>
<tr>
<td>Remove Method</td>
<td>184</td>
</tr>
<tr>
<td>Update Method</td>
<td>185</td>
</tr>
</tbody>
</table>

© 2019 Devart
Properties .................................................................................................................. 218
Connection Property .................................................................................................. 219
MetaDataKind Property ............................................................................................... 220
Restrictions Property .................................................................................................. 221
Methods ...................................................................................................................... 221
GetMetaDataKinds Method ......................................................................................... 224
GetRestrictions Method .............................................................................................. 224
TDAParams Class ....................................................................................................... 225
Members ..................................................................................................................... 226
Properties .................................................................................................................. 227
AsBlob Property ......................................................................................................... 229
AsBlobRef Property .................................................................................................... 229
AsFloat Property ....................................................................................................... 230
AsInteger Property .................................................................................................... 230
AsLargeInt Property .................................................................................................. 231
AsMemo Property ...................................................................................................... 231
AsMemoRef Property ................................................................................................. 232
AsSQLTimeStamp Property ...................................................................................... 232
AsString Property ..................................................................................................... 232
AsStringProperty ....................................................................................................... 233
DataType Property .................................................................................................... 233
IsNull Property ......................................................................................................... 234
ParamType Property .................................................................................................. 234
Size Property ............................................................................................................. 235
Value Property .......................................................................................................... 235
Methods ...................................................................................................................... 236
AssignField Method .................................................................................................. 236
AssignFieldValue Method ........................................................................................... 237
LoadFromFile Method ............................................................................................... 237
LoadFromStream Method ........................................................................................... 238
SetBlobData Method ................................................................................................. 239
SetBlobData Method ................................................................................................. 239
TDAParams Method ................................................................................................. 240
Members ..................................................................................................................... 241
Properties .................................................................................................................. 241
Items Property (Indexer) .......................................................................................... 242
Methods ...................................................................................................................... 242
FindParam Method ..................................................................................................... 243
ParamByName Method ............................................................................................... 243
TMacro Class ............................................................................................................. 244
Members ..................................................................................................................... 245
Properties .................................................................................................................. 245
Active Property ......................................................................................................... 246
AsDateTime Property ................................................................................................. 247
AsFloat Property ....................................................................................................... 247
AsInteger Property .................................................................................................... 247
AsString Property ..................................................................................................... 248
Name Property .......................................................................................................... 248
Value Property .......................................................................................................... 249
TMacros Class ........................................................................................................... 249
Members ..................................................................................................................... 250
Properties .................................................................................................................. 250
Items Property (Indexer) .......................................................................................... 251
© 2019 Devart
Methods
AssignValues Method .............................................................. 252
Expand Method ........................................................................ 252
FindMacro Method ................................................................. 253
IsEqual Method ...................................................................... 254
MacroByName Method ............................................................ 254
Scan Method .......................................................................... 255
TPoolingOptions Class ........................................................... 255
Members ............................................................................... 256
Properties ............................................................................ 256
  ConnectionLifetime Property ........................................... 257
  MaxPoolSize Property ....................................................... 258
  MinPoolSize Property ....................................................... 258
  Validate Property ............................................................ 258
TSmartFetchOptions Class ....................................................... 259
Members ............................................................................... 259
Properties ............................................................................ 260
  Enabled Property ............................................................. 260
  LiveBlock Property .......................................................... 261
  PrefetchedFields Property .............................................. 261
  SQLGetKeyValues Property .............................................. 262
Types .................................................................................. 262
TAfterExecuteEvent Procedure Reference .......................... 263
TAfterFetchEvent Procedure Reference .............................. 263
TBeforeFetchEvent Procedure Reference ............................ 264
TConnectionLostEvent Procedure Reference ...................... 264
TDAConnectionErrorEvent Procedure Reference ................. 265
TDATransactionErrorEvent Procedure Reference ............... 265
TRefreshOptions Set ........................................................... 266
TUpdateExecuteEvent Procedure Reference ....................... 266
Enumerations ........................................................................ 267
TCheckMode Enumeration .................................................... 267
TLabelSet Enumeration ......................................................... 268
TRefreshOption Enumeration .............................................. 269
TRetryMode Enumeration ..................................................... 269
Variables ............................................................................ 270
BaseSQLOldBehavior Variable ............................................ 270
ChangeCursor Variable ......................................................... 271
SQLGeneratorCompatibility Variable .................................. 271
4 MemData ........................................................................... 272
Classes ............................................................................... 272
TAttribute Class ................................................................. 272
Members ............................................................................... 273
Properties ............................................................................ 274
  AttributeNo Property ......................................................... 275
  DataType Property .......................................................... 275
  Length Property ............................................................. 276
  ObjectType Property ......................................................... 276
  Offset Property ............................................................... 277
  Owner Property ............................................................... 277
  Scale Property .................................................................. 278
  Size Property .................................................................. 278
TBlob Class .......................................................................... 279
<table>
<thead>
<tr>
<th>Members</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString Property</td>
<td>AsWideString Property</td>
</tr>
<tr>
<td>IsUnicode Property</td>
<td>Size Property</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign Method</td>
</tr>
<tr>
<td>Clear Method</td>
</tr>
<tr>
<td>LoadFromFile Method</td>
</tr>
<tr>
<td>LoadFromStream Method</td>
</tr>
<tr>
<td>Read Method</td>
</tr>
<tr>
<td>SaveToFile Method</td>
</tr>
<tr>
<td>SaveToStream Method</td>
</tr>
<tr>
<td>Truncate Method</td>
</tr>
<tr>
<td>Write Method</td>
</tr>
</tbody>
</table>

TCompressedBlob Class

<table>
<thead>
<tr>
<th>Members</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed Property</td>
<td>CompressedSize Property</td>
</tr>
</tbody>
</table>

TDBObject Class

<table>
<thead>
<tr>
<th>Members</th>
</tr>
</thead>
</table>

TMemData Class

<table>
<thead>
<tr>
<th>Members</th>
</tr>
</thead>
</table>

TObjectType Class

<table>
<thead>
<tr>
<th>Members</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeCount Property</td>
<td>Attributes Property(Indexer)</td>
</tr>
<tr>
<td>DataType Property</td>
<td>Size Property</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindAttribute Method</td>
</tr>
</tbody>
</table>

TSharedObject Class

<table>
<thead>
<tr>
<th>Members</th>
<th>Properties</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef Method</td>
</tr>
<tr>
<td>Release Method</td>
</tr>
</tbody>
</table>

Types

<table>
<thead>
<tr>
<th>Members</th>
<th>Properties</th>
</tr>
</thead>
</table>

Enumerations

<table>
<thead>
<tr>
<th>Members</th>
<th>Properties</th>
</tr>
</thead>
</table>

5 MemDS

<table>
<thead>
<tr>
<th>Classes</th>
</tr>
</thead>
</table>

© 2019 Devart
6 VirtualDataSet

VirtualDataSet Class

<table>
<thead>
<tr>
<th>Members</th>
<th>311</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>314</td>
</tr>
<tr>
<td>CachedUpdates Property</td>
<td>315</td>
</tr>
<tr>
<td>IndexFieldNames Property</td>
<td>316</td>
</tr>
<tr>
<td>KeyExclusive Property</td>
<td>317</td>
</tr>
<tr>
<td>LocalConstraints Property</td>
<td>318</td>
</tr>
<tr>
<td>LocalUpdate Property</td>
<td>319</td>
</tr>
<tr>
<td>Prepared Property</td>
<td>319</td>
</tr>
<tr>
<td>Ranged Property</td>
<td>319</td>
</tr>
<tr>
<td>UpdateRecordTypes Property</td>
<td>320</td>
</tr>
<tr>
<td>UpdatesPending Property</td>
<td>321</td>
</tr>
<tr>
<td>Methods</td>
<td>321</td>
</tr>
<tr>
<td>ApplyRange Method</td>
<td>323</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>324</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>324</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>326</td>
</tr>
<tr>
<td>CancelRange Method</td>
<td>326</td>
</tr>
<tr>
<td>CancelUpdates Method</td>
<td>326</td>
</tr>
<tr>
<td>CommitUpdates Method</td>
<td>327</td>
</tr>
<tr>
<td>DeferredPost Method</td>
<td>328</td>
</tr>
<tr>
<td>EditRangeEnd Method</td>
<td>329</td>
</tr>
<tr>
<td>EditRangeStart Method</td>
<td>329</td>
</tr>
<tr>
<td>GetBlob Method</td>
<td>330</td>
</tr>
<tr>
<td>GetBlob Method</td>
<td>330</td>
</tr>
<tr>
<td>GetBlob Method</td>
<td>331</td>
</tr>
<tr>
<td>GetBlob Method</td>
<td>331</td>
</tr>
<tr>
<td>Locate Method</td>
<td>332</td>
</tr>
<tr>
<td>Locate Method</td>
<td>332</td>
</tr>
<tr>
<td>Locate Method</td>
<td>333</td>
</tr>
<tr>
<td>LocateEx Method</td>
<td>334</td>
</tr>
<tr>
<td>LocateEx Method</td>
<td>335</td>
</tr>
<tr>
<td>LocateEx Method</td>
<td>335</td>
</tr>
<tr>
<td>Prepare Method</td>
<td>336</td>
</tr>
<tr>
<td>RestoreUpdates Method</td>
<td>337</td>
</tr>
<tr>
<td>RevertRecord Method</td>
<td>338</td>
</tr>
<tr>
<td>SaveToXML Method</td>
<td>338</td>
</tr>
<tr>
<td>SaveToXML Method</td>
<td>339</td>
</tr>
<tr>
<td>SaveToXML Method</td>
<td>339</td>
</tr>
<tr>
<td>SetRange Method</td>
<td>340</td>
</tr>
<tr>
<td>SetRangeEnd Method</td>
<td>341</td>
</tr>
<tr>
<td>SetRangeStart Method</td>
<td>342</td>
</tr>
<tr>
<td>UnPrepare Method</td>
<td>342</td>
</tr>
<tr>
<td>UpdateResult Method</td>
<td>343</td>
</tr>
<tr>
<td>UpdateStatus Method</td>
<td>344</td>
</tr>
<tr>
<td>Events</td>
<td>344</td>
</tr>
<tr>
<td>OnUpdateError Event</td>
<td>345</td>
</tr>
<tr>
<td>OnUpdateRecord Event</td>
<td>346</td>
</tr>
<tr>
<td>Variables</td>
<td>346</td>
</tr>
<tr>
<td>DoNotRaiseExceptionOnUsFail Variable</td>
<td>347</td>
</tr>
<tr>
<td>SendDataSetChangeEventAfterOpen Variable</td>
<td>347</td>
</tr>
</tbody>
</table>

© 2019 Devart
Members ......................................................................................................................................................... 352
Events ......................................................................................................................................................... 355
  OnDeleteRecord Event ................................................................................................................................. 356
  OnGetField Value Event ............................................................................................................................... 357
  OnGetRecordCount Event .............................................................................................................................. 358
  OnInsertRecord Event ................................................................................................................................. 358
  OnModifyRecord Event ............................................................................................................................... 359

Types .......................................................................................................................................................... 359
  TOnDeleteRecordEvent Procedure Reference ............................................................................................ 360
  TOnGetField Value Event Procedure Reference .......................................................................................... 360
  TOnGetRecordCountEvent Procedure Reference ......................................................................................... 361
  TOnModifyRecordEvent Procedure Reference ............................................................................................ 361

7 VirtualQuery ........................................................................................................................................ 362

Classes ......................................................................................................................................................... 362
  TCustomVirtualQuery Class .......................................................................................................................... 362
    Members .................................................................................................................................................. 363
    Properties ................................................................................................................................................ 371
      Options Property ................................................................................................................................ 375
      SourceDataSets Property ....................................................................................................................... 375
    Events ...................................................................................................................................................... 376
      OnRegisterCollations Event .................................................................................................................. 377
      OnRegisterFunctions Event .................................................................................................................. 377
  TDataSetLink Class .................................................................................................................................. 378
    Members .................................................................................................................................................. 379
    Properties ................................................................................................................................................ 379
      DataSet Property ................................................................................................................................ 380
      SchemaName Property .......................................................................................................................... 380
      TableName Property ............................................................................................................................. 381
  TDataSetLinks Class .................................................................................................................................. 381
    Members .................................................................................................................................................. 382
    Methods ................................................................................................................................................... 382
      Add Method ......................................................................................................................................... 383
      Add Method ......................................................................................................................................... 383
      Add Method ......................................................................................................................................... 384
  TVirtualCollationManager Class .................................................................................................................. 385
    Members .................................................................................................................................................. 385
  TVirtualFunctionManager Class ................................................................................................................... 385
    Members .................................................................................................................................................. 386
  TVirtualQuery Class ................................................................................................................................... 386
    Members .................................................................................................................................................. 387
    Properties ................................................................................................................................................ 387
      FetchAll Property ................................................................................................................................. 395
      UpdatingTable Property ....................................................................................................................... 400
  TVirtualQueryOptions Class .......................................................................................................................... 401
    Members .................................................................................................................................................. 401
    Properties ................................................................................................................................................ 404
      AutoOpenSources Property .................................................................................................................. 407
      FullRefresh Property ............................................................................................................................ 408
      SetEmptyStrToNull Property .................................................................................................................. 408
      TrimVarChar Property ............................................................................................................................ 409

Types .......................................................................................................................................................... 409
  TRegisterFunctionsEvent Procedure Reference .......................................................................................... 409

8 VirtualTable .......................................................................................................................................... 410
Classes .......................................................................................................................................................... 410
TVirtualAutoIncField Class ........................................................................................................................ 410
   Members .................................................................................................................................................. 411
   Properties ............................................................................................................................................. 411
      AutoGenerateValue Property ............................................................................................................. 412
      Increment Property ....................................................................................................................... 413
      InitialValue Property ..................................................................................................................... 413
TVirtualTable Class ..................................................................................................................................... 414
   Members ............................................................................................................................................... 415
   Properties ........................................................................................................................................... 418
      DefaultSortType Property ............................................................................................................. 420
      Options Property .......................................................................................................................... 420
   Methods ............................................................................................................................................. 420
      AddField Method .......................................................................................................................... 423
      Assign Method .............................................................................................................................. 424
      Clear Method ............................................................................................................................... 425
      DeleteField Method ...................................................................................................................... 425
      DeleteFields Method .................................................................................................................... 426
      LoadFromFile Method .................................................................................................................. 426
      LoadFromStream Method .............................................................................................................. 427
      SaveToFile Method ....................................................................................................................... 428
      SaveToStream Method .................................................................................................................... 428
Types ......................................................................................................................................................... 429
TVirtualTableOptions Set ............................................................................................................................ 429
Enumerations ............................................................................................................................................... 429
TVirtualTableOption Enumeration ........................................................................................................... 430
1 What's New

26-Nov-19 New Features in VirtualTable 11.1
• Android 64-bit is supported
• Lazarus 2.0.6 is supported
• Now Trial edition for macOS and Linux is fully functional

22-Jul-19 New Features in VirtualTable 11.0
• 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 VirtualTable 10.6
• Lazarus 2.0.2 is supported
• The DefaultSortType property for TVirtualTable is added
• Performance of the SaveToFile/LoadFromFile methods of TVirtualTable is significantly increased

26-Nov-18 New Features in VirtualTable 10.5
• RAD Studio 10.3 Rio is supported
• Support of UPPER and LOWER functions for Unified SQL is added

09-Jul-18 New Features in VirtualTable 10.4
• Lazarus 1.8.4 is supported
• The LocalUpdate property for TVirtualQuery is added
• The AutoOpenSources option for TVirtualQuery is added

18-Jan-18 New Features in VirtualTable 10.3
• Lazarus is supported

19-Sep-17 New Features in VirtualDAC 10.2
• The performance of TVirtualQuery is significantly improved
• Application-defined functions in TVirtualQuery are supported
• Application-defined collations in TVirtualQuery are supported
• AutoInc fields in TVirtualTable are supported

05-Apr-17 New Features in VirtualDAC 10.1
• RAD Studio 10.2 Tokyo is supported
• Linux in RAD Studio 10.2 Tokyo is supported

08-Sep-16 Release of VirtualDAC 10.0
• VirtualTable is renamed to VirtualDAC
• TVirtualDataSet component is added
• TVirtualQuery component is added

25-Apr-16 New Features in VirtualTable 9.7
• RAD Studio 10.1 Berlin is supported
• Lazarus 1.6 and FPC 3.0.0 is supported

09-Sep-16 New Features in VirtualTable 9.6
• RAD Studio 10 Seattle is supported

14-Apr-15 New Features in VirtualTable 9.5
• RAD Studio XE8 is supported
• AppMethod is supported

15-Sep-14 New Features in VirtualTable 9.4
• RAD Studio XE7 is supported
• Lazarus 1.2.2 and FPC 2.6.4 is supported

29-Apr-14 New Features in VirtualTable 9.3
• RAD Studio XE6 is supported
• Android in C++Builder XE6 is supported
• Lazarus 1.2.2 and FPC 2.6.4 is supported
• An option for saving all data avoiding filter is added

25-Dec-13 New Features in VirtualTable 9.2
• 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

19-Sep-13 New Features in VirtualTable 9.1
• 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
• The ReadOnly property is added
• The voSetEmptyStrToNull option 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

25-Apr-13 New Features in VirtualTable 9.0
• RAD Studio XE4 is supported
• NEXTGEN compiler is supported
• Application development for iOS is supported
• Support of TimeStamp, Single and Extended fields is added

14-Feb-13 New Features in VirtualTable 8.6
• RAD Studio XE3 Update 2 is now required
• C++Builder 64-bit for Windows is supported

07-Nov-12 New Features in VirtualTable 8.5
• RAD Studio XE3 is supported
• Windows 8 is supported

21-Jun-12 New Features in VirtualTable 8.2
• Update 4 Hotfix 1 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required

02-Apr-12 New Features in VirtualTable 8.1
• Lazarus 0.9.30.4 and FPC 2.6.0 are supported
• Bug with saving BCD fields to a Stream is fixed
• Update 4 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
• macOS in RAD Studio XE2 is supported
• FireMonkey support is improved

15-Sep-11 New Features in VirtualTable 8.0
• 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 is added
• OnProgress event is added

15-Jun-11 New Features in VirtualTable 7.20
• Possibility to load data without recreating persistent fields is added
• Working in design time editor is improved
• Lazarus 0.9.30 and FPC 2.4.2 is supported

10-Mar-11 New Features in VirtualTable 7.10
• Delphi & C++Builder XE Starter Edition supported

13-Sep-10 New Features in VirtualTable 7.00
• Embarcadero RAD Studio XE supported

10-Sep-09 New Features in VirtualTable 6.90
• Embarcadero RAD Studio 2010 supported

27-May-09 New Features in VirtualTable 6.80
• Added distinction between empty string and null value when saving/loading string fields
• Added support for Free Pascal under Linux

26-Feb-09 New Features in VirtualTable 6.70
• Delphi 2009 and C++Builder 2009 supported
• Extended Unicode support for Delphi 2007 added (special Unicode build)
• Free Pascal 2.2 supported

14-Nov-07 New Features in VirtualTable 6.25
• CodeGear RAD Studio 2007 supported
• Added support for TVariantField

02-Aug-07 New Features in VirtualTable 6.10
• C++Builder 2007 supported

20-Apr-07 New Features in VirtualTable 6.00
• Delphi 2007 for Win32 supported
• LargInt fields supported
• WideMemo field type in Delphi 2006 supported
• General performance improved

16-Feb-07 New Features in VirtualTable 5.80
• Added support for Professional editions of Turbo Delphi, Turbo Delphi for .NET, Turbo C++

07-Jul-06 New Features in VirtualTable 5.70
• Support for Delphi 2006 added
• Performance of SaveToFile and LoadFromFile functions improved (12435)
• Modifying FieldDefs is accelerated

29-Apr-05 New Features in VirtualTable 5.50
• Support for Delphi 2005 added
• Update Pack 3 is required for Delphi 8

09-Apr-04 New Features in VirtualTable 5.00
• Support for Delphi 8 added

08-Oct-02 New Features in VirtualTable 4.00
• Delphi 7 support added

03-Sep-02 New Features in VirtualTable 3.90
• Kylix3 support

14-Dec-01 New Features in VirtualTable 3.50
• Support for Kylix 2 is added

20-Feb-01 New Features in VirtualTable 3.00
• Support for Kylix is added

30-Mar-00 New Features in VirtualTable 2.20
• Support for C++ Builder 5 is added

14-Oct-99 New Features in VirtualTable 2.00
• Smart refreshing is added

01-Mar-99 First release of VirtualTable 1.00
2 General Information

This section contains general information about Virtual Data Access Components

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

© 1997-2019
Devart. All Rights Reserved.

2.1 Overview

Virtual Data Access Components Overview

Virtual Data Access Components (VirtualDAC) is a library of components that extends data processing capabilities in Delphi, C++Builder and Lazarus (FPC) on Windows, macOS, Linux, iOS, Android, for both 32-bit and 64-bit platforms. VirtualDAC consists of components that provides such capabilities as working with non-tabular data, performing SQL-queries to data in memory or to data from sources in different RDBMS'es etc., that are not available for standard data access components.

VirtualQuery

The TVirtualQuery component is used to retrieve data with SQL queries from sources, that are any TDataSet descendants. Due to this, you can build complex SQL queries to in-memory data (data sources are TVirtualTable, TVirtualDataSet or another TVirtualQuery) or to data stored in different RDBMS'es (data sources can be TUniQuery, TOraQuery, TMSQuery, TMyQuery, etc. at the same time).

VirtualTable

The TVirtualTable component represents an in-memory data storage that does not have
linked data files. TVirtualTable allows to load data from a XML file, or from an existing dataset, and then process the data in the usual way, as in any TDataSet descendant.

VirtualDataSet

The TVirtualDataSet component is data wrapper that doesn't store data in memory and interacts with data using event handlers. It allows to represent arbitrary data (arrays, lists, objects, etc.) as TDataSet descendants and link it with any DB-aware components to display data or modify it.

Description of VirtualDAC package

- The TVirtualTable component represents an in-memory data storage that does not have linked data files. TVirtualTable allows to load data from a XML file, or from an existing dataset, and then process the data in the usual way, as in any TDataSet descendant.
- The TVirtualDataSet component does not store data in memory and interacts with data using event handlers. It can be used to represent of arbitrary data (arrays, lists etc.) in tabular form, and for modifying the data using usual TDataSet methods.
- The TVirtualQuery component is used to retrieve data simultaneously from several different RDBMS'es. Instead of a database connection, it use a collection of TDataSet descendants as the data source, to which a SQL-query can be performed.

Key Features

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

- Performing SQL-queries simultaneously to non-RDBMS data sources
- Working with non-tabular data using familiar data-access components
- Easy visualizing and editing arbitrary data with standard data-aware controls without writing extra code
- 32-bit and 64-bit OS support
- macOS support
- iOS support
- Android support
- FireMonkey support
- All types of local sorting and filtering, including by calculated and lookup fields
- Automatic data updating
- Unicode and national charset support
- Support for using macros and parameters in SQL
- Absolutely free Express edition that consists of TVirtualTable and TVirtualDataSet
components
- Includes annual VirtualDAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

© 1997-2019
Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

2.2 Features
In this topic you will find the complete VirtualDAC feature list sorted by categories

General usability
- Interface compatible with standard data access methods, such as BDE and ADO
- VCL, FMX, LCL development platforms are available
- Separated run-time and GUI specific parts allow you to create pure console applications such as CGI
- Unicode and national charset support

TVirtualQuery
- Building complex SQL queries to data from different TDataSet descendants
- Database-independent SQL engine with TMemoryTable as a source
- Ability for querying non-database data using TVirtualDataSet as a source
- Almost full supporting of the SQLite SQL dialect

TVirtualTable
- Database-independent in-memory data storage
- Can save and load the table structure and record data into/from the file at run-time
- Can store the table structure and record data in dfm at design-time
- Allows to add and remove fields to the existing data
- Optimized string and row data storing

TVirtualDataSet
- Represents arbitrary data (arrays, lists, objects, etc.) as a TDataSet descendant
- Allows to display data using any DB-aware controls
- Ability for modifying data using trivial TDataSet methods

Data storage operations
- Local sorting and filtering, including by calculated and lookup fields
• Local master/detail relationship
• Extended capabilities for record locating

Data exchange
• Data export and import to/from XML (ADO format)

Design-time enhancements
• Advanced design-time component and property editors

Product clarity
• Complete documentation sets
• Helpful demo projects

Licensing and support
• Included annual VirtualDAC Subscription with Priority Support
• Licensed royalty-free per developer, per team, or per site

2.3 Requirements
An application based on VirtualDAC does not require any additional files or client libraries. If the TVirtualQuery component is used in an application, the need for additional libraries depends on data-access components which are used as source datasets for TVirtualQuery.

2.4 Compatibility

IDE Compatibility
VirtualDAC is compatible with the following IDEs:
• Embarcadero RAD Studio 10.3 Rio (Requires Release 2)
  o Embarcadero Delphi 10.3 Rio for Windows 32-bit & 64-bit
  o Embarcadero Delphi 10.3 Rio for macOS 32-bit & 64-bit
  o Embarcadero Delphi 10.3 Rio for Linux 64-bit
  o Embarcadero Delphi 10.3 Rio for iOS 32-bit & 64-bit
- Embarcadero Delphi 10.3 Rio for Android 32-bit & 64-bit
- Embarcadero C++Builder 10.3 Rio for Windows 32-bit & 64-bit
- Embarcadero C++Builder 10.3 Rio for macOS
- Embarcadero C++Builder 10.3 Rio for iOS 32-bit & 64-bit
- Embarcadero C++Builder 10.3 Rio for Android

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

- Embarcadero RAD Studio 10.1 Berlin
  - Embarcadero Delphi 10.1 Berlin for Windows 32-bit & 64-bit
  - Embarcadero Delphi 10.1 Berlin for macOS
  - Embarcadero Delphi 10.1 Berlin for iOS 32-bit & 64-bit
  - Embarcadero Delphi 10.1 Berlin for Android
  - Embarcadero C++Builder 10.1 Berlin for Windows 32-bit & 64-bit
  - Embarcadero C++Builder 10.1 Berlin for macOS
  - Embarcadero C++Builder 10.1 Berlin for iOS 32-bit & 64-bit
  - Embarcadero C++Builder 10.1 Berlin for Android

- Embarcadero RAD Studio 10 Seattle
  - Embarcadero Delphi 10 Seattle for Windows 32-bit & 64-bit
  - Embarcadero Delphi 10 Seattle for macOS
  - Embarcadero Delphi 10 Seattle for iOS 32-bit & 64-bit
  - Embarcadero Delphi 10 Seattle for Android
  - Embarcadero C++Builder 10 Seattle for Windows 32-bit & 64-bit
  - Embarcadero C++Builder 10 Seattle for macOS
  - Embarcadero C++Builder 10 Seattle for iOS 32-bit & 64-bit
  - Embarcadero C++Builder 10 Seattle for Android

- Embarcadero RAD Studio XE8
  - Embarcadero Delphi XE8 for Windows 32-bit & 64-bit
  - Embarcadero Delphi XE8 for macOS
  - Embarcadero Delphi XE8 for iOS 32-bit & 64-bit
- Embarcadero Delphi XE8 for Android
- Embarcadero C++Builder XE8 for Windows 32-bit & 64-bit
- Embarcadero C++Builder XE8 for macOS
- Embarcadero C++Builder XE8 for iOS 32-bit & 64-bit
- Embarcadero C++Builder XE8 for Android

- Embarcadero RAD Studio XE7
  - Embarcadero Delphi XE7 for Windows 32-bit & 64-bit
  - Embarcadero Delphi XE7 for macOS
  - Embarcadero Delphi XE7 for iOS
  - Embarcadero Delphi XE7 for Android
  - Embarcadero C++Builder XE7 for Windows 32-bit & 64-bit
  - Embarcadero C++Builder XE7 for macOS
  - Embarcadero C++Builder XE7 for iOS
  - Embarcadero C++Builder XE7 for Android

- Embarcadero RAD Studio XE6
  - Embarcadero Delphi XE6 for Windows 32-bit & 64-bit
  - Embarcadero Delphi XE6 for macOS
  - Embarcadero Delphi XE6 for iOS
  - Embarcadero Delphi XE6 for Android
  - Embarcadero C++Builder XE6 for Windows 32-bit & 64-bit
  - Embarcadero C++Builder XE6 for macOS
  - Embarcadero C++Builder XE6 for iOS
  - Embarcadero C++Builder XE6 for Android

- Embarcadero RAD Studio XE5 (Requires Update 2)
  - Embarcadero Delphi XE5 for Windows 32-bit & 64-bit
  - Embarcadero Delphi XE5 for macOS
  - Embarcadero Delphi XE5 for iOS
  - Embarcadero Delphi XE5 for Android
  - Embarcadero C++Builder XE5 for Windows 32-bit & 64-bit
  - Embarcadero C++Builder XE5 for macOS
  - Embarcadero C++Builder XE5 for iOS

- Embarcadero RAD Studio XE4
  - Embarcadero Delphi XE4 for Windows 32-bit & 64-bit
  - Embarcadero Delphi XE4 for macOS
  - Embarcadero Delphi XE4 for iOS
  - Embarcadero C++Builder XE4 for Windows 32-bit & 64-bit
  - Embarcadero C++Builder XE4 for macOS
- Embarrasser RAD Studio XE3 (Requires Update 2)
  - Embarrasser Delphi XE3 for Windows 32-bit & 64-bit
  - Embarrasser Delphi XE3 for macOS
  - Embarrasser C++Builder XE3 for Windows 32-bit & 64-bit
  - Embarrasser C++Builder XE3 for macOS
- Embarrasser RAD Studio XE2 (Requires Update 4 Hotfix 1)
  - Embarrasser Delphi XE2 for Windows 32-bit & 64-bit
  - Embarrasser Delphi XE2 for macOS
  - Embarrasser C++Builder XE2 for Windows 32-bit
  - Embarrasser C++Builder XE2 for macOS
- Embarrasser RAD Studio XE
  - Embarrasser Delphi XE
  - Embarrasser C++Builder XE
- Embarrasser RAD Studio 2010
  - Embarrasser Delphi 2010
  - Embarrasser C++Builder 2010
- CodeGear RAD Studio 2009 (Requires Update 3)
  - CodeGear Delphi 2009
  - CodeGear C++Builder 2009
- CodeGear RAD Studio 2007
  - CodeGear Delphi 2007
  - CodeGear C++Builder 2007
- CodeGear RAD Studio 2006
  - CodeGear Delphi 2006
  - CodeGear C++Builder 2006
- Borland Delphi 7
- Borland Delphi 6 (Requires Update Pack 2 – Delphi 6 Build 6.240)
- Borland C++Builder 6 (Requires Update Pack 4 – C++Builder 6 Build 10.166)
- Lazarus 2.0.6 and Free Pascal 3.0.4 for Windows, Linux, macOS, FreeBSD for 32-bit and 64-bit platforms

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

Note that support for 64-bit Windows and macOS was introduced in RAD Studio XE2, and is not available in older versions of RAD Studio. Support for iOS is available since RAD Studio XE4, but support for iOS 64-bit is available since RAD Studio XE8. Support for Android is available since RAD Studio XE5. Support for Linux 64-bit is available since RAD Studio 10.2 Tokyo. 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.

DAC products 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.

Note that VirtualDAC Express Edition is not compatible with the following Data Access Components, as they already contain TVirtualTable and TVirtualDataSet components:

- Universal Data Access Components
- Oracle Data Access Components
- SQL Server Data Access Components
- MySQL Data Access Components
- InterBase Data Access Components
- PostgreSQL Data Access Components
- SQLite Data Access Components

VirtualDAC Standard Edition is not compatible with Universal Data Access Components, as it already contains TVirtualTable, TVirtualDataSet and TVirtualQuery components.
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
- dacvcXX.bpl
- dcdacXX.bpl

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

2.6 Component List

This topic presents a brief description of the components included in the Virtual Data Access Components library. Click on the name of each component for more information. These components does not have its own page, and are added to the Data Access page of the Component palette. TVirtualTable component is included in all VirtualDAC editions.

VirtualDAC Express Edition includes only the TVirtualTable component and is absolutely free. TVirtualTable and TVirtualDataSet components are also included in all editions of other Devart Data Access Components.

**VirtualDAC Express Edition components**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVirtualTable</td>
<td>Dataset that stores data in memory.</td>
</tr>
</tbody>
</table>

**VirtualDAC Standard Edition components**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVirtualDataSet</td>
<td>Dataset that processes arbitrary non-tabular data.</td>
</tr>
<tr>
<td>TVirtualQuery</td>
<td>Dataset that allows to use SQL statements to retrieve data from in-memory datasets or simultaneously from several different RDBMS’es.</td>
</tr>
</tbody>
</table>
See Also
- Hierarchy chart

© 1997-2019 Devart. All Rights Reserved.

2.7 Hierarchy Chart

Many LiteDAC classes are inherited from standard VCL/LCL classes. The inheritance hierarchy chart for LiteDAC is shown below. The LiteDAC 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
    - TDataSet
      - TMemDataSet
        - TCustomDADataSet
          - TCustomVirtualQuery
            - TVirtualQuery
          - TCustomVirtualDataSet
            - TVirtualDataSet
            - TVirtualTable

© 1997-2019 Devart. All Rights Reserved.

2.8 Editions


VirtualDAC Express Edition includes only the TVirtualTable component and is absolutely free.

VirtualDAC Standard Edition includes all the VirtualDAC components.

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

You can get Source Access to the VirtualDAC Standard Edition by purchasing the special
**VirtualDAC Standard Edition with Source Code.** The Standard edition include the source code for all component classes. **FreePascal** support is available in the **Standard Edition with Source Code** and **Trial Edition**.

### VirtualDAC Edition Matrix

<table>
<thead>
<tr>
<th>Feature</th>
<th>Express</th>
<th>Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Virtual Data Access Components</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TVirtualTable</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TVirtualDataSet</td>
<td>☑️</td>
<td>✓</td>
</tr>
<tr>
<td>TVirtualQuery</td>
<td>☑️</td>
<td>✓</td>
</tr>
<tr>
<td>Desktop Application Development</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>macOS</td>
<td>☑️</td>
<td>✓</td>
</tr>
<tr>
<td>Linux</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Mobile Application Development</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iOS</td>
<td>☑️</td>
<td>✓</td>
</tr>
<tr>
<td>Android</td>
<td>☑️</td>
<td>✓</td>
</tr>
<tr>
<td>Design-Time Features</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enhanced component and property editors</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
2.9 Licensing

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

INTRODUCTION

This Devart end-user license agreement ("Agreement") is a legal agreement between you (either an individual person or a single legal entity) and Devart, for the use of VirtualDAC 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;

© 2019 Devart
• create libraries, components, and frameworks derived from the Software for personal use only;
• deploy and register run-time libraries and packages of the Software, subject to the Redistribution policy defined below.

1.3. You are allowed to use evaluation versions of the Software as specified in the Evaluation section.
No other rights or privileges are granted in this Agreement.

2. LIMITATIONS
Only legally registered users are licensed to use the Software, subject to all of the conditions of this Agreement. Usage of the Software is subject to the following restrictions.
2.1. You may not reverse engineer, decompile, or disassemble the Software.
2.2. You may not build any other components through inheritance for public distribution or commercial sale.
2.3. You may not use any part of the source code of the Software (original or modified) to build any other components for public distribution or commercial sale.
2.4. You may not reproduce or distribute any Software documentation without express written permission from Devart.
2.5. You may not distribute and sell any portion of the Software without integrating it into your Applications as Executable Code, except Trial edition that can be distributed for free as original Devart's VirtualDAC 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 VirtualDAC. You can distribute VirtualDAC only in compiled Executable Programs or Dynamic-Link Libraries with required run-time libraries and packages.
All Devart's units, source code, and other files remain Devart's exclusive property.

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

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

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

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

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

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

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

© 1997-2019
Devart. All Rights Reserved.

2.10 Getting Support

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

Support Options

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

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

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

VirtualDAC Priority Support

VirtualDAC Priority Support is an advanced product support service for getting expedited individual assistance with VirtualDAC-related questions from the VirtualDAC 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 VirtualDAC Subscription. To get help through the VirtualDAC 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 VirtualDAC Registration number.
- Full VirtualDAC edition name and version number. You can find both of these from the
Getting Started

This section contains installation and deployment instructions for Virtual Data Access Components.

3.1 Installation

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

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

Before installing VirtualDAC ...

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

So before installing a new version of VirtualDAC, uninstall all previous versions of VirtualDAC 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 virtualdac@devart.com

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

### Installed packages

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

<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>dcddacXX.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>virtualdacXX.bpl</td>
<td>VirtualDAC run-time package</td>
<td>Windows\System32</td>
</tr>
<tr>
<td>dclvirtualdacXX.bpl</td>
<td>VirtualDAC design-time package</td>
<td>Delphi\Bin</td>
</tr>
</tbody>
</table>

### Environment Changes

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

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

**Delphi**

- %VirtualDAC% Li b should be included in the Library Path accessible from Tools | Enviroment options | Library.

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

**C++Builder**

C++Builder 6:

- $(BCB)\Vi r t u a l DAC\ Li b should be included in the Library Path of the Default Project
Options accessible from Project | Options | Directories/Conditionals.

- $(BCB)\ V r t ual DAC\ I nclude should be included in the Include Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.

C++Builder 2006, 2007:

- $(BCB)\ V r t ual DAC\ Li b should be included in the Library search path of the Default Project Options accessible from Project | Default Options | C++Builder | Linker | Paths and Defines.

- $(BCB)\ V r t ual DAC\ I nclude 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 VirtualDAC Installer performs C++Builder environment changes automatically for compiled versions of VirtualDAC.

**Lazarus**

The VirtualDAC installation program only copies VirtualDAC files. You need to install VirtualDAC packages to the Lazarus IDE manually. Open %VirtualDAC%\Source\Lazarus1\virtualdac10.lpk (for Trial version %VirtualDAC%\Packages\virtualdac10.lpk) file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilded with VirtualDAC packages.

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

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

© 1997-2019 Devar. All Rights Reserved.

3.2 Deployment

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

**Deploying Windows applications built without run-time packages**

You do not need to deploy any files with VirtualDAC-based applications built without run-time
packages, provided you are using a registered version of VirtualDAC.
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 VirtualDAC Trial Edition, you will
need to deploy the following DAC BPL files:

- dacXX.bpl  always
- virtualdacXX.bpl  always

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

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

**Deploying Windows applications built with run-time packages**
You can set your application to be built with run-time packages by selecting the "Build with
runtime packages" check box in the Project Options dialog box before compiling your
application.
In this case, you will also need to deploy the following BPL files with your Windows
application:

- dacXX.bpl  always
- virtualdacXX.bpl  always

© 1997-2019
Devart. All Rights Reserved.

## 4 Reference

This page shortly describes units that exist in VirtualDAC.

### Units

<table>
<thead>
<tr>
<th>Unit Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 2019 Devart
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dac.DASQLMonitor</td>
<td>This unit contains the base class for the TVirtualSQLMonitor component.</td>
</tr>
<tr>
<td>Dac.DBAccess</td>
<td>This unit contains base classes for most of the components.</td>
</tr>
<tr>
<td>Dac.MemData</td>
<td>This unit contains classes for storing data in memory.</td>
</tr>
<tr>
<td>Dac.MemDS</td>
<td>This unit contains implementation of the TMemDataSet class.</td>
</tr>
<tr>
<td>Dac.VirtualDataSet</td>
<td>This unit contains implementation of the TVirtualDataSet component.</td>
</tr>
<tr>
<td>Dac.VirtualTable</td>
<td>This unit contains implementation of the TVirtualTable component.</td>
</tr>
<tr>
<td>DALoader</td>
<td>Description is not available at the moment.</td>
</tr>
<tr>
<td>DASQLMonitor</td>
<td>Description is not available at the moment.</td>
</tr>
<tr>
<td>DBAccess</td>
<td>Description is not available at the moment.</td>
</tr>
<tr>
<td>MemData</td>
<td>Description is not available at the moment.</td>
</tr>
<tr>
<td>MemDS</td>
<td>Description is not available at the moment.</td>
</tr>
<tr>
<td>VirtualDataSet</td>
<td>Description is not available at the moment.</td>
</tr>
<tr>
<td>VirtualQuery</td>
<td>Description is not available at the moment.</td>
</tr>
<tr>
<td>VirtualTable</td>
<td>Description is not available at the moment.</td>
</tr>
</tbody>
</table>
4.1 DALoader

4.1.1 Classes

Classes in the DALoader unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDALoaderOptions</td>
<td>Allows loading external data into database.</td>
</tr>
</tbody>
</table>

4.1.1.1 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);

4.1.1.1.1 Members

TDALoaderOptions class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseBlankValues</td>
<td>Forces VirtualDAC to fill the buffer with null values after loading a row to the database.</td>
</tr>
</tbody>
</table>
4.1.1.2.1 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 VirtualDAC 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**

© 1997-2019 Devart. All Rights Reserved.

4.1.1.2.1.1 UseBlankValues Property

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

Class

**TDALoaderOptions**

Syntax

```typescript
property UseBlankValues: boolean default True;
```

Remarks

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

© 1997-2019 Devart. All Rights Reserved.
4.2 DASQLMonitor

4.2.1 Classes

Classes in the DASQLMonitor unit.

<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>

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.
4.2.1.1.1 Members

**TCustomDASQLMonitor** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to activate monitoring of SQL.</td>
</tr>
<tr>
<td>DBMonitorOptions</td>
<td>Used to set options for dbMonitor.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to include the desired properties for TCustomDASQLMonitor.</td>
</tr>
<tr>
<td>TraceFlags</td>
<td>Used to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSQL</td>
<td>Occurs when tracing of SQL activity on database components is needed.</td>
</tr>
</tbody>
</table>

4.2.1.1.2 Properties

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

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

### Active Property

Used to activate monitoring of SQL.

#### Class

**TCustomDASQLMonitor**

#### Syntax

```pascal
property Active: boolean default True;
```

#### Remarks

Set the Active property to True to activate monitoring of SQL.

### DBMonitorOptions Property

Used to set options for dbMonitor.

#### Class

**TCustomDASQLMonitor**

#### Syntax
### Options Property

Used to include the desired properties for TCustomDASQLMonitor.

**Class**

TCustomDASQLMonitor

**Syntax**

```objectivec
property Options: TMonitorOptions default [moDialog, moSQLMonitor, moDBMonitor, moCustom];
```

**Remarks**

Set Options to include the desired properties for TCustomDASQLMonitor.

**See Also**

- [OnSQL](#)

### TraceFlags Property

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

**Class**

TCustomDASQLMonitor

**Syntax**

```objectivec
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

© 1997-2019 Devart. All Rights Reserved.

### 4.2.1.1.3 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`

© 1997-2019 Devart. All Rights Reserved.

### 4.2.1.1.3.1 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

### 4.2.1.2 TDBMonitorOptions Class

This class holds options for dbMonitor. For a list of all members of this type, see TDBMonitorOptions members.

#### Unit
DASQLMonitor

#### Syntax

```plaintext
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</td>
</tr>
</tbody>
</table>
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`

4.2.1.2.2.1 Host Property

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

Class

`TDBMonitorOptions`
Syntax

```plaintext
property Host: string;
```

Remarks

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

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

Class

`TDBMonitorOptions`

Syntax

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

Remarks

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

Class

`TDBMonitorOptions`

Syntax

```plaintext
property ReconnectTimeout: integer default
```

4.2.1.2.2.3 ReconnectTimeout Property

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

Class

`TDBMonitorOptions`

Syntax

```plaintext
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.

4.2.1.2.2.4 SendTimeout Property

Used to set timeout for sending events to dbMonitor.

Class
TDBMonitorOptions

Syntax
property SendTimeout: integer default DefaultSendTimeout;

Remarks
Use the SendTimeout property to set timeout (in milliseconds) for sending events to dbMonitor. If dbMonitor does not respond in the specified timeout, event is ignored.

4.2.2 Types
Types in the DASQLMonitor unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlags</td>
<td>Represents the set of TDATraceFlag.</td>
</tr>
<tr>
<td>TMonitorOptions</td>
<td>Represents the set of TMonitorOption.</td>
</tr>
<tr>
<td>TOnSQLEvent</td>
<td>This type is used for the TCustomDASQLMonitor.On</td>
</tr>
</tbody>
</table>
4.2.2.1 TDATraceFlags Set

Represents the set of TDATraceFlag.

Unit
DASQLMonitor

Syntax

TDATraceFlags = set of TDATraceFlag;

4.2.2.2 TMonitorOptions Set

Represents the set of TMonitorOption.

Unit
DASQLMonitor

Syntax

TMonitorOptions = set of TMonitorOption;

4.2.2.3 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.

4.2.3 Enumerations

Enumerations in the DASQLMonitor unit.

<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>

4.2.3.1 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);
```
### 4.2.3.2 `TMonitorOption` Enumeration

Used to define where information from SQLMonitor will be displayed.

#### Unit

`DASQLMonitor`

#### Syntax

```pascal
TMonitorOption = (moDialog, moSQLMonitor, moDBMonitor, moCustom, moHandled);
```

#### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tfBlob</code></td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td><code>tfConnect</code></td>
<td>Establishing a connection.</td>
</tr>
<tr>
<td><code>tfError</code></td>
<td>Errors of query execution.</td>
</tr>
<tr>
<td><code>tfMisc</code></td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td><code>tfObjDestroy</code></td>
<td>Destroying of components.</td>
</tr>
<tr>
<td><code>tfParams</code></td>
<td>Representing parameter values for <code>tQPrepare</code> and <code>tQExecute</code>.</td>
</tr>
<tr>
<td><code>tfPool</code></td>
<td>Connection pool operations.</td>
</tr>
<tr>
<td><code>tfQExecute</code></td>
<td>Execution of the queries.</td>
</tr>
<tr>
<td><code>tfQFetch</code></td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td><code>tfQPrepare</code></td>
<td>Queries preparation.</td>
</tr>
<tr>
<td><code>tfService</code></td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td><code>tfStmt</code></td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td><code>tfTransact</code></td>
<td>Processing transactions.</td>
</tr>
</tbody>
</table>

---

```pascal
© 1997-2019
Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>moCustom</code></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 <code>moDialog</code> is included.</td>
</tr>
<tr>
<td><code>moDBMonitor</code></td>
<td>Debug information is displayed in A:Using_DBMonitor.</td>
</tr>
<tr>
<td><code>moDialog</code></td>
<td>Debug information is displayed in debug window.</td>
</tr>
</tbody>
</table>
Classes in the **DBAccess** unit.

### 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>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 <strong>TDAConditions</strong> list.</td>
</tr>
<tr>
<td>TDAConditions</td>
<td>Holds a collection of <strong>TDACondition</strong> objects.</td>
</tr>
<tr>
<td>TDACConnectionOptions</td>
<td>This class allows setting up the behaviour of the <strong>TDACConnection</strong> class.</td>
</tr>
<tr>
<td>Class/Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TDADatasetOptions</td>
<td>This class allows setting up the behaviour of the TDADataset class.</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>TMacro</td>
<td>Object that represents the value of a macro.</td>
</tr>
<tr>
<td>TMacros</td>
<td>Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataset components.</td>
</tr>
<tr>
<td>TPoolingOptions</td>
<td>This class allows setting up the behaviour of the connection pool.</td>
</tr>
<tr>
<td>TSmartFetchOptions</td>
<td>Smart fetch options are used to set up the behavior of the SmartFetch mode.</td>
</tr>
</tbody>
</table>

### 4.3.1.1 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.
EDAError is a base class for exceptions that are raised when an error occurs on the server side.

Properties

- **Component**: Contains the component that caused the error.
- **ErrorCode**: Determines the error code returned by the server.

Public

- **Component**: Contains the component that caused the error.
- **ErrorCode**: Determines the error code returned by the server.

See Also

- [EDAError Class](#)
4.3.1.1.2.1 Component Property

Contains the component that caused the error.

Class
EDAError

Syntax

```plaintext
property Component: TObject;
```

Remarks
The Component property contains the component that caused the error.

4.3.1.2 ErrorCode Property

Determines the error code returned by the server.

Class
EDAError

Syntax

```plaintext
property ErrorCode: integer;
```

Remarks
Use the ErrorCode property to determine the error code returned by SQLite. This value is always positive.

4.3.1.2 TCustomConnectDialog Class

A base class for the connect dialog components.
For a list of all members of this type, see TCustomConnectDialog members.
Unit

DBAccess

Syntax

```
TCustomConnectDialog = class(TComponent);
```

Remarks

TCustomConnectDialog is a base class for the connect dialog components. It provides functionality to show a dialog box where user can edit username, password and server name before connecting to a database. You can customize captions of buttons and labels by their properties.

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>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.</td>
</tr>
</tbody>
</table>
ServerLabel

Used to specify a prompt for the server name edit.

StoreLogInfo

Used to specify whether the login information should be kept in system registry after a connection was established.

UsernameLabel

Used to specify a prompt for username edit.

Methods

<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

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

### 4.3.1.2.2.1 CancelButton Property

Used to specify the label for the Cancel button.

**Class**

TCustomConnectDialog

**Syntax**

```plaintext
property CancelButton: string;
```

**Remarks**

Use the CancelButton property to specify the label for the Cancel button.
4.3.1.2.2.2 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.

4.3.1.2.2.3 ConnectButton Property

Used to specify the label for the Connect button.

Class

TCustomConnectDialog

Syntax

```
property ConnectButton: string;
```

Remarks

Use the ConnectButton property to specify the label for the Connect button.

4.3.1.2.2.4 DialogClass Property

Used to specify the class of the form that will be displayed to enter login information.

Class

TCustomConnectDialog

Syntax
property DialogClass: string;

Remarks
Use the DialogClass property to specify the class of the form that will be displayed to enter login information. When this property is blank, TCustomConnectDialog uses the default form - TConnectForm. You can write your own login form to enter login information and assign its class name to the DialogClass property. Each login form must have ConnectDialog: TCustomConnectDialog published property to access connection information. For details see the implementation of the connect form which sources are in the Lib subdirectory of the VirtualDAC installation directory.

See Also
• GetServerList

4.3.1.2.2.5 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 IsEnglish.

4.3.1.2.2.6 PasswordLabel Property

Used to specify a prompt for password edit.

Class
TCustomConnectDialog
4.3.1.2.2.7  Retries Property

Used to indicate the number of retries of failed connections.

Class

TCustomConnectDialog

Syntax

```plaintext
property Retries: word default 3;
```

Remarks

Use the Retries property to determine the number of retries of failed connections.

4.3.1.2.8  SavePassword Property

Used for the password to be displayed in ConnectDialog in asterisks.

Class

TCustomConnectDialog

Syntax

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

Remarks

If True, and the Password property of the connection instance is assigned, the password in ConnectDialog is displayed in asterisks.
4.3.1.2.2.9 ServerLabel Property

Used to specify a prompt for the server name edit.

Class
TCustomConnectDialog

Syntax

```property ServerLabel: string;```

Remarks

Use the ServerLabel property to specify a prompt for the server name edit.

4.3.1.2.2.10 StoreLogInfo Property

Used to specify whether the login information should be kept in system registry after a connection was established.

Class
TCustomConnectDialog

Syntax

```property StoreLogInfo: boolean default True;```

Remarks

Use the StoreLogInfo property to specify whether to keep login information in system registry after a connection was established using provided username, password and servername. Set this property to True to store login information. The default value is True.
4.3.1.2.11 UsernameLabel Property

Used to specify a prompt for username edit.

Class

TCustomConnectDialog

Syntax

```
property UsernameLabel: string;
```

Remarks

Use the UsernameLabel property to specify a prompt for username edit.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.</td>
</tr>
<tr>
<td>GetServerList</td>
<td>Retrieves a list of available server names.</td>
</tr>
</tbody>
</table>

See Also

- TCustomConnectDialog Class
- TCustomConnectDialog Class Members

4.3.1.3.1 Execute Method

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.
### Class

**TCustomConnectDialog**

### Syntax

```pascal
function Execute: boolean; virtual;
```

### Return Value

True, if connected.

### Remarks

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button. Returns True if connected. If user clicks Cancel, Execute returns False. In the case of failed connection Execute offers to connect repeat **Retries** times.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.2.3.2 GetServerList Method

Retrieves a list of available server names.

### Class

**TCustomConnectDialog**

### Syntax

```pascal
procedure GetServerList(List: TStrings); virtual;
```

### Parameters

- **List**
  
  Holds a list of available server names.

### Remarks

Call the GetServerList method to retrieve a list of available server names. It is particularly relevant for writing custom login form.

### See Also

- **DialogClass**

© 1997-2019 Devart. All Rights Reserved.
4.3.1.3  **TCustomDAConnection Class**

A base class for components used to establish connections. For a list of all members of this type, see **TCustomDAConnection** members.

**Unit**

DBAccess

**Syntax**

```pascal
TCustomDAConnection = class(TCustomConnection);
```

**Remarks**

TCustomDAConnection is a base class for components that establish connection with database, provide customised login support, and perform transaction control. Do not create instances of TCustomDAConnection. To add a component that represents a connection to a source of data, use descendants of the TCustomDAConnection class.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.3.1 Members

**TCustomDAConnection** 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>--------------</td>
<td>------------------------------------</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>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>Get Stored Proc Names</td>
<td>Returns a list of stored procedures from the server.</td>
</tr>
</tbody>
</table>
GetTableNames | Provides a list of available tables names.
---|---
MonitorMessage | Sends a specified message through the TCustomDASQLMonitor component.
Ping | Used to check state of connection to the server.
RemoveFromPool | Marks the connection that should not be returned to the pool after disconnect.
Rollback | Discards all current data changes and ends transaction.
StartTransaction | Begins a new user transaction.

**Events**

<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>

© 1997-2019 Devart. All Rights Reserved.

### 4.3.1.3.2 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</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UserName, Password, Server, etc.</td>
<td>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>Username</td>
<td>Used to supply a user name for login.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDAConnection Class
- TCustomDAConnection Class Members

© 1997-2019 Devart. All Rights Reserved.

4.3.1.3.2.1 ConnectDialog Property

Allows to link a TCustomConnectDialog component.

Class
TCustomDAConnection

Syntax

```property`` ConnectDialog: TCustomConnectDialog;

Remarks
Use the ConnectDialog property to assign to connection a TCustomConnectDialog component.

See Also
• TCustomConnectDialog

4.3.1.3.2.2 ConnectString Property

Used to specify the connection information, such as: UserName, Password, Server, etc.

Class
TCustomDAConnection

Syntax

property ConnectString: string stored False;

Remarks
VirtualDAC 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</td>
</tr>
</tbody>
</table>
Validate Connection

Used for a connection to be validated when it is returned from the pool.

See Also
- Password
- Username
- Server
- Connect

4.3.1.3.2.3 ConvertEOL Property

Allows customizing line breaks in string fields and parameters.

Class
TCustomDAConnection

Syntax

```property` ConvertEOL: boolean `default` False;```

Remarks

Affects the line break behavior in string fields and parameters. When fetching strings (including the TEXT fields) with ConvertEOL = True, dataset converts their line breaks from the LF to CRLF form. And when posting strings to server with ConvertEOL turned on, their line breaks are converted from CRLF to LF form. By default, strings are not converted.

4.3.1.3.2.4 InTransaction Property

Indicates whether the transaction is active.

Class
TCustomDAConnection
**Syntax**

```property``` InTransaction: boolean;

**Remarks**

Examine the InTransaction property at runtime to determine whether user transaction is currently in progress. In other words InTransaction is set to True when user explicitly calls `StartTransaction`. Calling `Commit` or `Rollback` sets InTransaction to False. The value of the InTransaction property cannot be changed directly.

**See Also**

- `StartTransaction`
- `Commit`
- `Rollback`

© 1997-2019 Devart. All Rights Reserved.

### 4.3.1.3.2.5 LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

**Class**

`TCustomDACConnection`

**Syntax**

```property``` LoginPrompt default DefValLoginPrompt;

**Remarks**

Specifies whether a login dialog appears immediately before opening a new connection. If `ConnectDialog` is not specified, the default connect dialog will be shown. The connect dialog will appear only if the VirtualDACVcl unit appears to the uses clause.

© 1997-2019 Devart. All Rights Reserved.

### 4.3.1.3.2.6 Options Property

Specifies the connection behavior.

**Class**
**TCustomDAConnection**

**Syntax**

```
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. 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 OnConnectionLost event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
</tbody>
</table>

See Also

- A:Work_DisconnectMode
- A:Unstable_Network

© 1997-2019 Devart. All Rights Reserved.

4.3.1.3.2.7 Password Property

Serves to supply a password for login.

**Class**

TCustomDAConnection
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

Enables or disables using connection pool.

Class
TCustomDAConnection

Syntax
property Pooling: boolean default DefValPooling;

Remarks
Normally, when TCustomDAConnection establishes connection with the server it takes server memory and time resources for allocating new server connection. For example, pooling can be very useful when using disconnect mode. If an application has wide user activity that forces many connect/disconnect operations, it may spend a lot of time on creating connection and sending requests to the server. TCustomDAConnection has software pool which stores open connections with identical parameters. Connection pool uses separate thread that validates the pool every 30 seconds. Pool validation consists of checking each connection in the pool. If a connection is broken due to a network problem or another reason, it is deleted from the pool. The validation procedure removes also connections that are not used for a long time even if they are valid from the pool.
Set Pooling to True to enable pooling. Specify correct values for PoolingOptions. Two
connections belong to the same pool if they have identical values for the parameters: `MinPoolSize`, `MaxPoolSize`, `Validate`, `ConnectionLifeTime`

**Note:** Using Pooling := True can cause errors with working with temporary tables.

**See Also**
- `Username`
- `Password`
- `PoolingOptions`
- `A:work_pooling`

**Remarks**
Set the properties of `PoolingOptions` to specify the behaviour of connection pool.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ConnectionLifetime</code></td>
<td>Used to specify the maximum time during which an opened connection can be used by connection pool.</td>
</tr>
<tr>
<td><code>MaxPoolSize</code></td>
<td>Used to specify the maximum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td><code>MinPoolSize</code></td>
<td>Used to specify the minimum number of connections that can be opened in the connection pool.</td>
</tr>
<tr>
<td><code>Validate</code></td>
<td>Used for a connection to be validated when it is returned from the pool.</td>
</tr>
</tbody>
</table>

**See Also**
4.3.1.3.2.10 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

Warning: Storing hard-coded user name and password entries as property values or in code for the OnLogin event handler can compromise server security.

4.3.1.3.2.11 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, VirtualDAC tries to connect with the user name.
4.3.1.3.3 Methods

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td>Commit</td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td>Connect</td>
<td>Establishes a connection to the server.</td>
</tr>
<tr>
<td>CreateSQL</td>
<td>Creates a component for queries execution.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Performs disconnect.</td>
</tr>
<tr>
<td>ExecProc</td>
<td>Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td>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>
</tbody>
</table>
### 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
- TCustomDACConnection Class
- TCustomDACConnection Class Members

© 1997-2019 Devart. All Rights Reserved.

4.3.1.3.3.1 ApplyUpdates Method

Applies changes in datasets.

#### Class
TCustomDACConnection

#### 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`
- `TMemDataSet.ApplyUpdates`

Applies changes from the specified datasets.

Class

TCustomDAConnection

Syntax

```plaintext
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.

4.3.1.3.3.2 Commit Method

Commits current transaction.

Class

TCustomDAConnection

Syntax

procedure Commit; virtual;

Remarks

Call the Commit method to commit current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database and then ends the transaction. The current transaction is the last transaction started by calling StartTransaction.

See Also

• Rollback
• StartTransaction
• P:Devart.VirtualDac.TCustomVirtualDataSet.FetchAll

4.3.1.3.3.3 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

4.3.1.3.3.4 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.
4.3.1.3.5 Disconnect Method

Performs disconnect.

Class

TCustomDAConnection

Syntax

procedure Disconnect;

Remarks

Call the Disconnect method to drop a connection to database. Before the connection component is deactivated, all associated datasets are closed. Calling Disconnect is similar to setting the Connected property to False.

In most cases, closing a connection frees system resources allocated to the connection. If user transaction is active, e.g. the InTransaction flag is set, calling to Disconnect the current user transaction.

Note: If a previously active connection is closed and then reopened, any associated datasets must be individually reopened; reopening the connection does not automatically reopen associated datasets.

See Also

• Connect

4.3.1.3.6 ExecProc Method

Allows to execute stored procedure or function providing its name and parameters.

Class

TCustomDAConnection

Syntax

function ExecProc(const Name: string; const Params: array of variant): variant; virtual;

Parameters

Name

Holds the name of the stored procedure or function.
**Params**

Holds the parameters of the stored procedure or function.

**Return Value**

the result of the stored procedure.

**Remarks**

Allows to execute stored procedure or function providing its name and parameters. Use the following Name value syntax for executing specific overloaded routine: "StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.

Assign parameters' values to the Params array in exactly the same order and number as they appear in the stored procedure declaration. Out parameters of the procedure can be accessed with the ParamByName procedure.

If the value of an input parameter was not included to the Params array, parameter default value is taken. Only parameters at the end of the list can be unincluded to the Params array. If the parameter has no default value, the NULL value is sent.

**Note:** Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. The stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For further examples of parameter usage see ExecSQL, ExecSQLEx.

**Example**

For example, having stored function declaration presented in Example 1), you may execute it and retrieve its result with commands presented in Example 2):

```
Example 1)
CREATE  procedure  MY_SUM (      A INTEGER,      B INTEGER)
RETURNS (      RESULT INTEGER)
as
begin
    Result = a + b;
end;
Example 2)
Label2.Caption:= MyVirtualConnection1.ParamByName('Result').AsString;
```

See Also

- ExecProcEx
4.3.1.3.3.7 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

© 1997-2019
Devart. All Rights
Reserved.

4.3.1.3.3.8 ExecSQL Method

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.

### See Also
- ExecSQLEx
- ExecProc

### Class
**TCustomDAConnection**

### Syntax
```pascal
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

```pascal
VirtualConnection.ExecSQLEx('begin :A:= :B + :C; end;',
['A', 0, 'B', 5, 'C', 3]);
A:= VirtualConnection.ParamByName('A').AsInteger;
```

See Also
- ExecSQL

4.3.1.3.3.10 GetDatabaseNames Method

Returns a database list from the server.

Class

TCustomDAConnection

Syntax

```pascal
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
### 4.3.1.3.3.11 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`

© 1997-2019 Devart. All Rights Reserved.

### 4.3.1.3.3.12 GetStoredProcNames Method

Returns a list of stored procedures from the server.

**Class**

`TCustomDAConnection`

**Syntax**

```pascal
procedure GetStoredProcNames(List: TStrings; AllProcs: boolean = true);
```
Parameters

List
A TStrings descendant that will be filled with the names of stored procedures in the database.

AllProcs
True, if stored procedures from all schemas or including system procedures (depending on the server) are returned. False otherwise.

Remarks

Call the GetStoredProcNames method to get the names of available stored procedures and functions. GetStoredProcNames populates a string list with the names of stored procs in the database. If AllProcs = True, the procedure returns to the List parameter the names of the stored procedures that belong to all schemas; otherwise, List will contain the names of functions that belong to the current schema.

Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetStoredProcNames.

See Also
- GetDatabaseNames
- GetTableNames

© 1997-2019 Devart. All Rights Reserved.

4.3.1.3.3.13 GetTableNames Method

Provides a list of available tables names.

Class

TCustomDAConnection

Syntax

```delphi
procedure GetTableNames(List: TStrings; AllTables: boolean = False; OnlyTables: boolean = False); virtual;
```

Parameters

List
A TStrings descendant that will be filled with table names.

AllTables
True, if procedure returns all table names including the names of system tables to the List parameter.

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

4.3.1.3.3.14  MonitorMessage Method

Sends a specified message through the TCustomDASQLMonitor component.

Class

TCustomDACConnection

Syntax

```plaintext
procedure MonitorMessage(const Msg: string);
```

Parameters

- `Msg`
  Message text that will be sent.

Remarks

Call the MonitorMessage method to output specified message via the TCustomDASQLMonitor component.

See Also
4.3.1.3.3.15 Ping Method

Used to check state of connection to the server.

Class
TCustomDAConnection

Syntax

```pascal
procedure Ping;
```

Remarks
The method is used for checking server connection state.

4.3.1.3.3.16 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
4.3.1.3.3.17 Rollback Method

Discards all current data changes and ends transaction.

Class

TCustomDAConnection

Syntax

```
procedure Rollback; virtual;
```

Remarks

Call the Rollback method to discard all updates, insertions, and deletions of data associated with the current transaction to the database server and then end the transaction. The current transaction is the last transaction started by calling StartTransaction.

See Also

- Commit
- StartTransaction
- P:Devart.VirtualDac.TCustomVirtualDataSet.FetchAll

© 1997-2019
Devart. All Rights Reserved.

4.3.1.3.3.18 StartTransaction Method

Begins a new user transaction.

Class

TCustomDAConnection

Syntax

```
procedure StartTransaction; virtual;
```

Remarks

Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction property. If InTransaction is True, indicating that a transaction is already in progress, a subsequent call to StartTransaction without first calling Commit or Rollback to end the current transaction raises EDatabaseError. Calling StartTransaction when
connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes, or Rollback to cancel them.

See Also
- Commit
- Rollback
- InTransaction

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnectionLost</td>
<td>This event occurs when connection was lost.</td>
</tr>
<tr>
<td>OnError</td>
<td>This event occurs when an error has arisen in the connection.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDAConnection Class
- TCustomDAConnection Class Members

© 1997-2019 Devart. All Rights Reserved.
**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.

4.3.1.3.4.2 OnError Event

This event occurs when an error has arisen in the connection.

Class
TCustomDAConnection

Syntax

**property** OnError: TDACConnectionErrorEvent;

Remarks
Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

4.3.1.4 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
TCustomDADataSet = \textbf{class}(TMemDataSet);\\

Remarks

TCustomDADataSet encapsulates general set of properties, events, and methods for working with data accessed through various database engines. All database-specific features are supported by descendants of TCustomDADataSet. Applications should not use TCustomDADataSet objects directly.

Inheritance Hierarchy

\texttt{TMemDataSet} \rightarrow \texttt{TCustomDADataSet}

Properties

\begin{tabular}{|l|l|}
\hline
\textbf{Name} & \textbf{Description} \\
\hline
\texttt{BaseSQL} & Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL. \\
\hline
\texttt{CachedUpdates} (inherited from \texttt{TMemDataSet}) & Used to enable or disable the use of cached updates for a dataset. \\
\hline
\texttt{Conditions} & Used to add WHERE conditions to a query \\
\hline
\texttt{Connection} & Used to specify a connection object to use to connect to a data store. \\
\hline
\texttt{DataTypeMap} & Used to set data type mapping rules \\
\hline
\texttt{Debug} & Used to display executing statement, all its parameters' values, and the type of parameters. \\
\hline
\texttt{DetailFields} & Used to specify the fields that correspond to the foreign key fields from \\
\hline
\end{tabular}
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MasterFields</td>
<td>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>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>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from TMemDataSet) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>(inherited from TMemDataSet) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>(inherited from TMemDataSet) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>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>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MasterFields</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of TCustomDADataSet object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td>Prepared</td>
<td>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>SQL</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ApplyRange</strong></td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong></td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong></td>
<td>Used to learn whether TCustomDADataset has already fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong></td>
<td>Used to learn whether TCustomDADataset is still</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</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>Description is not available at the moment.</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 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</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</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 <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 name passed in <code>Name</code>.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Sets or uses 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 <code>AddWhere</code> and <code>SetOrderBy</code>.</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></td>
<td>Saves the SQL property value to <code>BaseSQL</code>.</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></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>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SetRangeStart (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>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 (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>UpdateResult (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>UpdateStatus (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>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 <strong>TMemDataSet</strong>)</td>
<td>Occurs when an exception is generated while cached updates are applied to a</td>
</tr>
</tbody>
</table>
### Properties

Properties of the `TCustomDADataset` class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BaseSQL</code></td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><code>CachedUpdates</code></td>
<td>(inherited from <code>TMemDataSet</code>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><code>Conditions</code></td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><code>Connection</code></td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><code>DataTypeMap</code></td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td><code>Debug</code></td>
<td>Used to display executing statement, all its parameters' values, and the type of parameters.</td>
</tr>
<tr>
<td><code>DetailFields</code></td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><code>Disconnected</code></td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</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>IndexFieldNames</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 whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>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>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</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of TCustomDADataSet object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td>Prepared (inherited from TMemDataSet)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged (inherited from TMemDataSet)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>ReadOnly</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>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</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLLock</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>Used if an application does not need bidirectional access to records in the result set.</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>
</tbody>
</table>

See Also
- TCustomDADataSet Class
- TCustomDADataSet Class Members

4.3.1.4.2.1 BaseSQL Property

Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.

Class
**TCustomDADataSet**

**Syntax**

```
property BaseSQL: string;
```

**Remarks**

Use the BaseSQL property to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL, only macros are expanded. SQL text with all these changes can be returned by **FinalSQL**.

**See Also**

- **FinalSQL**
- **AddWhere**
- **SaveSQL**
- **SQLSaved**
- **RestoreSQL**

© 1997-2019 Devart. All Rights Reserved.

4.3.1.4.2.2 Conditions Property

Used to add WHERE conditions to a query

**Class**

**TCustomDADataSet**

**Syntax**

```
property Conditions: TDAConditions stored False;
```

**See Also**

- **TDAConditions**

© 1997-2019 Devart. All Rights Reserved.
4.3.1.4.2.3 Connection Property

Used to specify a connection object to use to connect to a data store.

Class
TCustomDADataSet

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.
Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.
At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

© 1997-2019
Devart. All Rights Reserved.

4.3.1.4.2.4 DataTypeMap Property

Used to set data type mapping rules

Class
TCustomDADataSet

Syntax

```
property DataTypeMap: TDAMapRules stored IsMapRulesStored;
```

See Also

• TDAMapRules

© 1997-2019
Devart. All Rights Reserved.

4.3.1.4.2.5 Debug Property

Used to display executing statement, all its parameters' values, and the type of parameters.

Class
**TCustomDADataSet**

**Syntax**

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

**Remarks**

Set the Debug property to True to display executing statement and all its parameters' values. Also displays the type of parameters.

**See Also**

- [TCustomDASQL.Debug](#)

© 1997-2019 Devart. All Rights Reserved.

---

**4.3.1.4.2.6 DetailFields Property**

Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

**Class**

`TCustomDADataSet`

**Syntax**

```plaintext
property DetailFields: string;
```

**Remarks**

Use the DetailFields property to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship. DetailFields is a string containing one or more field names in the detail table. Separate field names with semicolons. Use Field Link Designer to set the value in design time.

**See Also**

- [MasterFields](#)
- [MasterSource](#)

© 1997-2019 Devart. All Rights Reserved.
4.3.1.4.2.7 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.

4.3.1.4.2.8 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.

4.3.1.4.2.9 FilterSQL Property

Used to change the WHERE clause of SELECT statement and reopen a query.

Class
TCPromptDAC98

© 2019 Devart

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

4.3.1.4.2.10  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

© 1997-2019
Devart. All Rights Reserved.
4.3.1.4.2.11  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.

4.3.1.4.2.12  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 key from source dataset.

See Also
- SQLDelete
- SQLInsert
- SQLRefresh
- SQLUpdate

4.3.1.4.2.13 MacroCount Property

Used to get the number of macros associated with the Macros property.

Class
TCustomDADataset

Syntax

```pascal
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also
- Macros

4.3.1.4.2.14 Macros Property

Makes it possible to change SQL queries easily.

Class
TCustomDADataset
**Syntax**

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

**Remarks**

With the help of macros you can easily change SQL query text at design- or runtime. Macros extend abilities of parameters and allow to change conditions in a WHERE clause or sort order in an ORDER BY clause. You just insert &MacroName in the SQL query text and change value of macro in the Macro property editor at design time or call the MacroByName function at run time. At the time of opening the query macro is replaced by its value.

**Example**

```VirtualQuery.SQL := 'SELECT * FROM Dept ORDER BY &Order';
VirtualQuery_MACROByName('Order').Value := 'DeptNo';
VirtualQuery.Open;```

**See Also**

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

© 1997-2019 Devart. All Rights Reserved.

4.3.1.4.2.15 MasterFields Property

Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

**Class**

[TCustomDADataset](#)

**Syntax**

```property MasterFields: string;```

**Remarks**

Use the MasterFields property after setting the [MasterSource](#) property to specify the names of one or more fields that are used as foreign keys for this dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons. Each time the current record in the master table changes, the new values in these fields are used to select corresponding records in this table for display. Use Field Link Designer to set the values at design time after setting the MasterSource property.

See Also
- MasterFields
- MasterSource
- A:Work_MD

Class
TCustomDADataset

Syntax

```property```

MasterSource: TDataSource;

Remarks
The MasterSource property specifies the data source component which binds current dataset to the master one. TCustomDADataset uses MasterSource to extract foreign key fields values from the master dataset when building master/detail relationship between two datasets. MasterSource must point to another dataset; it cannot point to this dataset component. When MasterSource is not `nil` dataset fills parameter values with corresponding field values from the current record of the master dataset. **Note:** Do not set the DataSource property when building master/detail relationships. Although it points to the same object as the MasterSource property, it may lead to undesirable results.

See Also
- MasterFields
- DetailFields
4.3.1.4.2.17 Options Property

Used to specify the behaviour of TCustomDADataset object.

Class

TCustomDADataset

Syntax

\begin{verbatim}
property Options: TDADataSetOptions;
\end{verbatim}

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 \texttt{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>DefaultValue</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</td>
</tr>
<tr>
<td>Property</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 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>
<tr>
<td><strong>UpdateBatchSize</strong></td>
<td>Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.</td>
</tr>
</tbody>
</table>
4.3.1.4.2.18 ParamCheck Property

Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Class

TCustomDAODataSet

Syntax

```
property ParamCheck: boolean default True;
```

Remarks

Use the ParamCheck property to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
Set ParamCheck to True to let dataset automatically generate the Params property for the dataset based on a SQL statement.
Setting ParamCheck to False can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of stored procedures which themselves will accept parameterized values. The default value is True.

See Also

• Params

4.3.1.4.2.19 ParamCount Property

Used to indicate how many parameters are there in the Params property.

Class

TCustomDAODataSet

Syntax
property ParamCount: word;

Remarks
Use the ParamCount property to determine how many parameters are there in the Params property.

See Also
• Params

4.3.1.4.2.20  Params Property

Used to view and set parameter names, values, and data types dynamically.

Class
TCustomDADataset

Syntax
property Params: TDAParams stored False;

Remarks
Contains the parameters for a query's SQL statement. Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access. An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

See Also
• ParamByName
• Macros

© 1997-2019 Devart. All Rights Reserved.
4.3.1.4.2.21 ReadOnly Property

Used to prevent users from updating, inserting, or deleting data in the dataset.

Class

TCustomDADataSet

Syntax

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

Remarks

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is False, meaning that users can potentially alter data stored in the dataset.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to True. When ReadOnly is True, the dataset's CanModify property is False.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.4.2.22 RefreshOptions Property

Used to indicate when the editing record is refreshed.

Class

TCustomDADataSet

Syntax

```plaintext
property RefreshOptions: TRefreshOptions default [];
```

Remarks

Use the RefreshOptions property to determine when the editing record is refreshed. Refresh is performed by the RefreshRecord method.

It queries the current record and replaces one in the dataset. Refresh record is useful when the table has triggers or the table fields have default values. Use roBeforeEdit to get actual data before editing.

The default value is [].

See Also

- RefreshRecord
4.3.1.4.2.23 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

TCustomDADataSet

Syntax

property RowsAffected: integer;

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

4.3.1.4.2.24 SQL Property

Used to provide a SQL statement that a query component executes when its Open method is called.

Class

TCustomDADataSet

Syntax

property SQL: TStrings;

Remarks

Use the SQL property to provide a SQL statement that a query component executes when its Open method is called. At the design time the SQL property can be edited by invoking the String List editor in Object Inspector. When SQL is changed, TCustomDADataSet calls Close and UnPrepare.

See Also
4.3.1.4.2.25 SQLDelete Property

Used to specify a SQL statement that will be used when applying a deletion to a record.

Class

TCustomDADataSet

Syntax

```
property SQLDelete: TStrings;
```

Remarks

Use the SQLDelete property to specify the SQL statement that will be used when applying a deletion to a record. Statements can be parameterized queries.

To create a SQLDelete statement at design-time, use the query statements editor.

Example

```
DELETE FROM Orders
WHERE
    OrderID = :Old_OrderID
```

See Also

- SQL
- SQLInsert
- SQLUpdate
- SQLRefresh

© 1997-2019
Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
4.3.1.4.2.26 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

4.3.1.4.2.27 SQLLock Property

Used to specify a SQL statement that will be used to perform a record lock.

Class
TCustomDADataSet

Syntax

```
property SQLLock: TStrings;
```

Remarks
Use the SQLLock property to specify a SQL statement that will be used to perform a record lock.
lock. Statements can be parameterized queries. Names of the parameters should be the
same as field names. The parameters prefixed with OLD_ allow to use current values of
fields prior to the actual operation.
To create a SQLLock statement at design-time, the use query statement editor.

See Also
- SQL
- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh

© 1997-2019
Devart. All Rights
Reserved.

4.3.1.4.2.28 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
TDDataSetOptions.QueryRecCount property is True, and the TCustomDADataSet.FetchAll
property is False. Is not used if the FetchAll property is True.
To create a SQLRecCount statement at design-time, use the query statements editor.

See Also
- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh
- TDDataSetOptions

© 2019 Devart
4.3.1.4.29 SQLRefresh Property

Used to specify a SQL statement that will be used to refresh current record by calling the RefreshRecord procedure.

Class

TCustomDADataset

Syntax

```plaintext
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

```plaintext
SELECT Shipname FROM Orders
WHERE
  OrderID = :OrderID
```

See Also

- RefreshRecord
- SQL
- SQLInsert
- SQLUpdate
- SQLDelete
4.3.1.4.2.30 SQLUpdate Property

Used to specify a SQL statement that will be used when applying an update to a dataset.

Class

TCustomDADataSet

Syntax

```plaintext
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

© 1997-2019 Devart. All Rights Reserved.

4.3.1.4.2.31 UniDirectional Property

Used if an application does not need bidirectional access to records in the result set.

Class

TCustomDADataSet
Syntax

```plaintext
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

- TVirtualQuery.FetchAll

© 1997-2019 Devart. All Rights Reserved.

4.3.1.4.3 Methods

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange</td>
<td>(inherited from TMemDataSet) Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>(inherited from TMemDataSet) Overloaded. Writes dataset's pending cached</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</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>Fetched</td>
<td>Used to learn whether TCustomDADataset has already 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>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FindKey</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Description is not available at the moment.</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 (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>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 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</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TMemDataSet.Locate</td>
<td>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 name passed in Name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Sets or uses parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td>RestoreUpdates (inherited from TMemDataSet)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord (inherited from TMemDataSet)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveSQL</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td>SaveToXML (inherited from TMemDataSet)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetOrderBy</td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td>SetRange (inherited from TMemDataSet)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
</tbody>
</table>
### SQLSaved
Determines if the SQL property value was saved to the BaseSQL property.

### UnLock
Releases a record lock.

### UnPrepare (inherited from TMemDataSet)
Frees the resources allocated for a previously prepared query on the server and client sides.

### UpdateResult (inherited from TMemDataSet)
Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

### UpdateStatus (inherited from TMemDataSet)
Indicates the current update status for the dataset when cached updates are enabled.

---

See Also
- TCustomDADataset Class
- TCustomDADataset Class Members

---

#### 4.3.1.4.3.1 AddWhere Method

Adds condition to the WHERE clause of SELECT statement in the SQL property.

**Class**

TCustomDADataset

**Syntax**

```delphi
procedure AddWhere(const Condition: string);
```

**Parameters**

- **Condition**
  
  Holds the condition that will be added to the WHERE clause.

---

© 2019 Devart
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**

4.3.1.4.3.2  **BreakExec Method**

Breaks execution of the SQL statement on the server.

Class

**TCustomDADataSet**

Syntax

```pascal
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to break execution of the SQL statement on the server. It makes sense to call BreakExec only from another thread.

4.3.1.4.3.3  **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.

Class
TCustomDADataSet

Syntax

procedure DeleteWhere;

Remarks
Call the DeleteWhere method to remove WHERE clause from the SQL property and assign BaseSQL property.

See Also
- AddWhere
- BaseSQL

© 1997-2019 Devart. All Rights Reserved.
4.3.1.4.3.5 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 the specified SQL query.</td>
</tr>
</tbody>
</table>

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.

The Execute implicitly prepares SQL statement by calling the TCustomDADataSet.Prepare method if the TCustomDADataSet.Options option is set to True and the statement has not been prepared yet. To speed up the performance in case of multiple Execute calls, an application should call Prepare before calling the Execute method for the first time.

See Also

- TCustomDADataSet.AfterExecute
- TCustomDADataSet.Executing
- TCustomDADataSet.Prepare

© 1997-2019
Devart. All Rights Reserved.
Used to perform the specified SQL query.

Class

`TCustomDADataSet`

Syntax

```pascal
procedure Execute(Iter: integer; Offset: integer = 0); overload;
virtual;
```

Parameters

- **Iter**
  - Specifies the number of inserted rows.
- **Offset**
  - Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The `Execute` method executes the specified SQL query.

© 1997-2019
Devar. All Rights Reserved.

4.3.1.4.3.6 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.

© 1997-2019
Devar. All Rights Reserved.
4.3.1.4.3.7 Fetched Method

Used to learn whether TCustomDADataSet has already fetched all rows.

Class
TCustomDADataSet

Syntax

```pascal
function Fetched: boolean; virtual;
```

Return Value
True, if all rows are fetched.

Remarks
Check Fetched to learn whether TCustomDADataSet has already fetched all rows.

See Also
• Fetching

© 1997-2019 Devart. All Rights Reserved.
Request Support  DAC Forum  Provide Feedback

4.3.1.4.3.8 Fetching Method

Used to learn whether TCustomDADataSet is still fetching rows.

Class
TCustomDADataSet

Syntax

```pascal
function Fetching: boolean;
```

Return Value
True, if TCustomDADataSet is still fetching rows.

Remarks
Check Fetching to learn whether TCustomDADataSet is still fetching rows. Use the Fetching method if NonBlocking is True.

See Also
• Executing
4.3.1.4.3.9 FetchingAll Method

Used to learn whether TCustomDADataSet is fetching all rows to the end.

Class

TCustomDADataSet

Syntax

```delphi
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

4.3.1.4.3.10 FindKey Method

Searches for a record which contains specified field values.

Class

TCustomDADataSet

Syntax

```delphi
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.

4.3.1.4.3.11 FindMacro Method

**Class**

*TCustomDADataSet*

**Syntax**

```pascal
function FindMacro(const Value: string): TMacro;
```

**Parameters**

*Value*

**See Also**

- *TMacro*
- *Macros*
- *MacroByName*

4.3.1.4.3.12 FindNearest Method

Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.

**Class**

*TCustomDADataSet*

**Syntax**

```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
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
4.3.1.4.3.14 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.

4.3.1.4.3.15 GetFieldObject Method

Returns a multireference shared object from field.

Class

TCustomDADataSet

Syntax

function GetFieldObject(Field: TField): TSharedObject; overload;

overload: function GetFieldObject(Field: TField; RecBuf: TRecordBuffer): TSharedObject; overload;
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.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.4.3.16 GetFieldPrecision Method

Retrieves the precision of a number field.

Class

TCustomDADataSet

Syntax

function GetFieldPrecision(const FieldName: string): integer;

Parameters

FieldName
    Holds the existing field name.

Return Value
    precision of number field.

Remarks

Call the GetFieldPrecision method to retrieve the precision of a number field. FieldName is the
name of an existing field.

See Also

• GetFieldScale

© 1997-2019 Devart
4.3.1.4.3.17 GetFieldScale Method

Retrieves the scale of a number field.

Class

TCustomDADataSet

Syntax

```function GetFieldScale(const FieldName: string): integer;```

Parameters

FieldName

Holds the existing field name.

Return Value

the scale of the number field.

Remarks

Call the GetFieldScale method to retrieve the scale of a number field. FieldName is the name of an existing field.

See Also

- GetFieldPrecision

© 1997-2019
Devart. All Rights Reserved.

4.3.1.4.3.18 GetKeyFieldNames Method

Provides a list of available key field names.

Class

TCustomDADataSet

Syntax

```procedure GetKeyFieldNames(List: TStrings);```

Parameters

List

The list of available key field names
### GetKeyFieldNames Method

**Return Value**
- Key field name

**Remarks**
Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

**See Also**
- `TCustomDAConnection.GetTableNames`
- `TCustomDAConnection.GetStoredProcNames`

---

### GetOrderBy Method

**Remarks**
Call the GetOrderBy method to retrieve an ORDER BY clause from a SQL statement.

**Class**
- `TCustomDADataset`

**Syntax**
```pascal
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`

© 1997-2019 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
4.3.1.4.3.20 GotoCurrent Method

Sets the current record in this dataset similar to the current record in another dataset.

**Class**

*TCustomDADataSet*

**Syntax**

```plaintext
procedure GotoCurrent(DataSet: TCustomDADataSet);
```

**Parameters**

`DataSet`

Holds the TCustomDADataSet descendant to synchronize the record position with.

**Remarks**

Call the GotoCurrent method to set the current record in this dataset similar to the current record in another dataset. The key fields in both these DataSets must be coincident.

**See Also**

- `TMemDataSet.Locate`
- `TMemDataSet.LocateEx`

© 1997-2019 Devart. All Rights Reserved.

4.3.1.4.3.21 Lock Method

Locks the current record.

**Class**

*TCustomDADataSet*

**Syntax**

```plaintext
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>`. 
4.3.1.4.3.22 MacroByName Method

Finds a Macro with the name passed in Name.

Class

TCustomDADataSet

Syntax

```pascal
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds the name of the Macro to search for.

Return Value

The Macro, if a match was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Name. If a match was found, MacroByName returns the Macro. Otherwise, an exception is raised. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

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

To assign the value of macro use the `TMacro.Value` property.

Example

```pascal
VirtualQuery.SQL := 'SELECT * FROM Scott.Dept ORDER BY &Order';
VirtualQuery.MacroByName('Order').Value := 'DeptNo';
VirtualQuery.Open;
```

See Also

- `TMacro`
- `Macros`
- `FindMacro`
4.3.1.4.3.23  ParamByName Method

Sets or uses parameter information for a specific parameter based on its name.

Class

TCustomDADataSet

Syntax

function ParamByName(const Value: string): TDAParam;

Parameters

Value

Holds the name of the parameter for which to retrieve information.

Return Value

a TDAParam object.

Remarks

Call the ParamByName method to set or use parameter information for a specific parameter based on its name. Name is the name of the parameter for which to retrieve information. ParamByName is used to set a parameter's value at runtime and returns a TDAParam object.

Example

The following statement retrieves the current value of a parameter called "Contact" into an edit box:

```
Edit1.Text := Query1.ParamsByName('Contact').AsString;
```

See Also

- Params
- FindParam

© 1997-2019
Devar. All Rights Reserved.
4.3.1.4.3.24  Prepare Method

Allocates, opens, and parses cursor for a query.

Class

TCustomDADataSet

Syntax

procedure Prepare; override;

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance. The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- TMemDataSet.Prepared
- TMemDataSet.UnPrepare
- Options

4.3.1.4.3.25  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.
4.3.1.4.3.26 RestoreSQL Method

Restores the SQL property modified by AddWhere and SetOrderBy.

Class

TCustomDADataSet

Syntax

```pascal
procedure RestoreSQL;
```

Remarks

Call the RestoreSQL method to restore the SQL property modified by AddWhere and SetOrderBy.

See Also

- AddWhere
- SetOrderBy
- SaveSQL
- SQLSaved

4.3.1.4.3.27 SaveSQL Method

Saves the SQL property value to BaseSQL.

Class

TCustomDADataSet

Syntax

```pascal
procedure SaveSQL;
```
SetOrderBy Method

Builds an ORDER BY clause of a SELECT statement.

Class

TCustomDADataset

Syntax

procedure Set.OrderBy(const Fields: string);

Parameters

Fields

Holds the names of the fields which will be added to the ORDER BY clause.

Remarks

Call the SetOrderBy method to build an ORDER BY clause of a SELECT statement. The fields are identified by the comma-delimited field names. 

Note: The GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

Example

Query1.Set.OrderBy('DeptNo;DName');

See Also

• GetOrderBy
4.3.1.4.3.29 SQLSaved Method

Determines if the SQL property value was saved to the BaseSQL property.

Class
TCustomDADataSet

Syntax

```pascal
function SQLSaved: boolean;
```

Return Value
True, if the SQL property value was saved to the BaseSQL property.

Remarks
Call the SQLSaved method to know whether the SQL property value was saved to the BaseSQL property.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.4.3.30 UnLock Method

Releases a record lock.

Class
TCustomDADataSet

Syntax

```pascal
procedure UnLock;
```

Remarks
Call the Unlock method to release the record lock made by the Lock method before. Unlock is performed by rolling back to the savepoint set by the Lock method.

See Also
- Lock

© 1997-2019 Devart. All Rights Reserved.
4.3.1.4.4 Events

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td>OnUpdateError</td>
<td>(inherited from TMemDataSet) Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord</td>
<td>(inherited from TMemDataSet) Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDADataset Class
- TCustomDADataset Class Members

© 1997-2019 Devart. All Rights Reserved.
Request Support DAC Forum Provide Feedback

4.3.1.4.4.1 AfterExecute Event

Occurs after a component has executed a query to database.
**TCustomDADataSet**

**Syntax**

```property` AfterExecute: `TAfterExecuteEvent;```

**Remarks**

Occurs after a component has executed a query to database.

**See Also**

- `TCustomDADataSet.Execute`

© 1997-2019
Devart. All Rights Reserved.

4.3.1.4.4.2  **AfterFetch Event**

Occurs after dataset finishes fetching data from server.

**Class**

`TCustomDADataSet`

**Syntax**

```property` AfterFetch: `TAfterFetchEvent;```

**Remarks**

The AfterFetch event occurs after dataset finishes fetching data from server.

**See Also**

- `BeforeFetch`

© 1997-2019
Devart. All Rights Reserved.

4.3.1.4.4.3  **AfterUpdateExecute Event**

Occurs after executing insert, delete, update, lock and refresh operations.

**Class**

`TCustomDADataSet`
4.3.1.4.4.4 BeforeFetch Event

Occurs before dataset is going to fetch block of records from the server.

Class

TCustomDADataset

Syntax

property BeforeFetch: TBeforeFetchEvent;

Remarks

The BeforeFetch event occurs every time before dataset is going to fetch a block of records from the server. Set Cancel to True to abort current fetch operation.

See Also

• AfterFetch

4.3.1.4.4.5 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

4.3.1.5 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.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>Enables or disables</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</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 executing statement, all its parameters' values, and the type of 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 property.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to contain parameters for a SQL statement.</td>
</tr>
<tr>
<td>ParamValues</td>
<td>Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
</tbody>
</table>
| SQL            | Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>Searches for 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 name passed in Name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Finds a parameter with the specified name.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>WaitExecuting</td>
<td>Waits until TCustomDASQL executes a SQL statement.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a SQL statement has been executed.</td>
</tr>
</tbody>
</table>

© 1997-2019 Devart. All Rights Reserved.

4.3.1.5.2 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 executing statement, all its parameters' values, and the type of 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 property.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to contain parameters for a SQL statement.</td>
</tr>
<tr>
<td>ParamValues</td>
<td>Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
</tbody>
</table>
SQL

Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

See Also
- TCustomDASQL Class
- TCustomDASQL Class Members

4.3.1.5.2.1 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.

4.3.1.5.2.2 Connection Property

Used to specify a connection object to use to connect to a data store.

Class
TCustomDASQL

Syntax
**property** Connection: **TCustomDAConnection**;

**Remarks**
Use the Connection property to specify a connection object that will be used to connect to a data store.
Set at design-time by selecting from the list of provided TCustomDACConnection or its descendant class objects.
At runtime, link an instance of a TCustomDACConnection descendant to the Connection property.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.5.2.3 **Debug Property**
Used to display executing statement, all its parameters' values, and the type of parameters.

**Class**
**TCustomDASQL**

**Syntax**

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

**Remarks**
Set the Debug property to True to display executing statement and all its parameters' values. Also displays the type of parameters.

**See Also**
- **TCustomDADataset.Debug**

© 1997-2019 Devart. All Rights Reserved.

4.3.1.5.2.4 **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.

4.3.1.5.2.5 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](#)

4.3.1.5.2.6 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
- MacroByName
- Params

© 1997-2019 Devart. All Rights Reserved.

4.3.1.5.2.7 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
4.3.1.5.2.8  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.

4.3.1.5.2.9  Params Property

Used to contain parameters for a SQL statement.

Class

TCustomDASQL

Syntax

| property | Params: TDAParams stored False; |

Remarks

Access the Params property at runtime to view and set parameter names, values, and data types dynamically (at design-time use the Parameters editor to set parameter properties). Params is a zero-based array of parameter records. Index specifies the array element to access. An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

Example

Setting parameters at runtime:

| procedure TForm1.Button1Click(Sender: TObject); |
begin with virtualSQL do
begin
SQL.Clear;
SQL.Add('INSERT INTO Temp_Table(Id, Name)');
SQL.Add('VALUES (:id, :Name)');
ParamByName('Id').AsInteger := 55;
Params[1].AsString := ' Green';
Execute;
end;
end;

See Also
- TDAParam
- FindParam
- Macros

© 1997-2019 Devart. All Rights Reserved.

4.3.1.5.2.10 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.

4.3.1.5.2.11  Prepared Property

Used to indicate whether a query is prepared for execution.

Class

**TCustomDASQL**

**Syntax**

```plaintext
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

4.3.1.5.2.12  RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

**TCustomDASQL**

**Syntax**

```plaintext
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.

© 1997-2019
Devart. All Rights Reserved.

4.3.1.5.2.13 SQL Property

Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

Class

TCustomDASQL

Syntax

property SQL: TStrings;

Remarks

Use the SQL property to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called. At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

See Also

• FinalSQL
• TCustomDASQL.Execute

© 1997-2019
Devart. All Rights Reserved.

4.3.1.5.3 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>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
</tbody>
</table>
Executing | Checks whether TCustomDASQL still executes a SQL statement.
FindMacro | Searches for a macro with the specified name.
FindParam | Finds a parameter with the specified name.
MacroByName | Finds a Macro with the name passed in Name.
ParamByName | Finds a parameter with the specified name.
Prepare | Allocates, opens, and parses cursor for a query.
UnPrepare | Frees the resources allocated for a previously prepared query on the server and client sides.
WaitExecuting | Waits until TCustomDASQL executes a SQL statement.

See Also
- TCustomDASQL Class
- TCustomDASQL Class Members

Executes a SQL statement on the server.

Class
TCustomDASQL

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Execute(Iters: integer; Offset: integer)</td>
<td>Used to perform the specified SQL query.</td>
</tr>
</tbody>
</table>
Executes a SQL statement on the server.

Class

TCustomDASQL

Syntax

```pascal
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.

© 1997-2019 Devart. All Rights Reserved.

Used to perform the specified SQL query.

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 SQL query.

© 1997-2019 Devart. All Rights Reserved.
4.3.1.5.3.2 Executing Method

Checks whether TCustomDASQL still executes a SQL statement.

Class

TCustomDASQL

Syntax

function Executing: boolean;

Return Value

True, if a SQL statement is still being executed by TCustomDASQL.

Remarks

Check Executing to find out whether TCustomDASQL still executes a SQL statement.

© 1997-2019

Devart. All Rights Reserved.

4.3.1.5.3.3 FindMacro Method

Searches for a macro with the specified name.

Class

TCustomDASQL

Syntax

function FindMacro(const Value: string): TMacro;

Parameters

Value

Holds the name of a macro to search for.

Return Value

the TMacro object, if a macro with the specified name has been found. If it has not, returns nil.

Remarks

Call the FindMacro method to find a macro with the specified name in a dataset.

See Also

• TMacro
4.3.1.5.3.4 FindParam Method

Finds a parameter with the specified name.

Class

TCustomDASQL

Syntax

```funcion 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**

4.3.1.5.3.5 MacroByName Method

Finds a Macro with the name passed in Name.

Class

TCustomDASQL

Syntax

```funcion MacroByName(const Value: string): TMacro;```
Parameters

Value
Holds the name of the Macro to search for.

Return Value
the Macro, if a match was found.

Remarks
Call the MacroByName method to find a Macro with the name passed in Name. If a match was found, MacroByName returns the Macro. Otherwise, an exception is raised. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.
To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.
To assign the value of macro use the TMacro.Value property.

See Also
- TMacro
- Macros
- FindMacro

© 1997-2019 Devart. All Rights Reserved.
Request Support DAC Forum Provide Feedback

4.3.1.5.3.6 ParamByName Method

Finds a parameter with the specified name.

Class
TCustomDASQL

Syntax

function ParamByName(const Value: string): TDAParam;

Parameters

Value
Holds the name of the parameter to search for.

Return Value
a TDAParam object, if a match was found. Otherwise, an exception is raised.

Remarks
Use the ParamByName method to find a parameter with the specified name. If no parameter with the specified name found, an exception is raised.

Example

```pascal
VirtualSQL.Execute;
Edit1.Text := VirtualSQL.ParamsByName('Contact').AsString;
```

See Also
- `FindParam`

4.3.1.5.3.7 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

`TCustomDASQL`

Syntax

```pascal
procedure Prepare; virtual;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

The UnPrepare method unprepares a query.

**Note:** When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also
- `Prepared`
- `UnPrepare`
4.3.1.5.3.8 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class
TCustomDASQL

Syntax
procedure UnPrepare; virtual;

Remarks
Call the UnPrepare method to free resources allocated for a previously prepared query on the server and client sides.

See Also
• Prepare

4.3.1.5.3.9 WaitExecuting Method

Waits until TCustomDASQL executes a SQL statement.

Class
TCustomDASQL

Syntax
function WaitExecuting(TimeOut: integer = 0): boolean;

Parameters

TimeOut
Holds the time in seconds to wait while TCustomDASQL executes the SQL statement. Zero means infinite time.

Return Value
True, if the execution of a SQL statement was completed in the preset time.

Remarks
Call the WaitExecuting method to wait until TCustomDASQL executes a SQL statement.

See Also
4.3.1.5.4 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

4.3.1.5.4.1 AfterExecute Event

Occurs after a SQL statement has been executed.

Class

TCustomDASQL

Syntax

```pascal
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
4.3.1.6  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

- P:Devart.VirtualDac.TCustomVirtualDataSet.UpdateObject

4.3.1.6.1 Members

TCustomDAUpdateSQL class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSet</td>
<td>Used to hold a reference to the TCustomDADataSet object that is being updated.</td>
</tr>
<tr>
<td>DeleteObject</td>
<td>Provides ability to perform advanced adjustment of the delete operations.</td>
</tr>
<tr>
<td>DeleteSQL</td>
<td>Used when deleting a record.</td>
</tr>
<tr>
<td><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><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apply</strong></td>
<td>Sets parameters for a SQL statement and executes it to update a record.</td>
</tr>
<tr>
<td><strong>ExecSQL</strong></td>
<td>Executes a SQL statement.</td>
</tr>
</tbody>
</table>
For a complete list of the **TCustomDAUpdateSQL** class members, see the [TCustomDAUpdateSQL Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSet</td>
<td>Used to hold a reference to the TCustomDADataset object that is being updated.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteObject</td>
<td>Provides ability to perform advanced adjustment of the delete operations.</td>
</tr>
<tr>
<td>DeleteSQL</td>
<td>Used when deleting a record.</td>
</tr>
<tr>
<td>InsertObject</td>
<td>Provides ability to perform advanced adjustment of insert operations.</td>
</tr>
<tr>
<td>InsertSQL</td>
<td>Used when inserting a record.</td>
</tr>
<tr>
<td>LockObject</td>
<td>Provides ability to perform advanced adjustment of lock operations.</td>
</tr>
<tr>
<td>LockSQL</td>
<td>Used to lock the current record.</td>
</tr>
<tr>
<td>ModifyObject</td>
<td>Provides ability to perform advanced adjustment of modify operations.</td>
</tr>
<tr>
<td>ModifySQL</td>
<td>Used when updating a record.</td>
</tr>
<tr>
<td>RefreshObject</td>
<td>Provides ability to perform advanced adjustment of refresh operations.</td>
</tr>
<tr>
<td>RefreshSQL</td>
<td>Used to specify an SQL statement that will be used for refreshing the current record by</td>
</tr>
</tbody>
</table>
4.3.1.6.2.1  DataSet Property

Used to hold a reference to the TCustomDADataset object that is being updated.

Class
TCustomDAUpdateSQL

Syntax

property  DataSet:  TCustomDADataset;

Remarks
The DataSet property holds a reference to the TCustomDADataset object that is being updated. Generally it is not used directly.

4.3.1.6.2.2  DeleteObject Property

Provides ability to perform advanced adjustment of the delete operations.

Class
TCustomDAUpdateSQL

Syntax

property  DeleteObject:  TComponent;

Remarks
Assign SQL component or a TCustomVirtualQuery 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`

**4.3.1.6.2.3 DeleteSQL Property**

Used when deleting a record.

**Class**

`TCustomDAUpdateSQL`

**Syntax**

```pascal
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.

**4.3.1.6.2.4 InsertObject Property**

Provides ability to perform advanced adjustment of insert operations.

**Class**

`TCustomDAUpdateSQL`

**Syntax**

```pascal
property InsertObject: TComponent;
```

**Remarks**

Assign SQL component or `TCustomVirtualQuery` descendant to this property to perform advanced adjustment of insert operations. In some cases this can give some additional
performance. Set the SQL property of the object in the same way as used for the InsertSQL property.

See Also
- InsertSQL

Class
TCustomDAUpdateSQL

Syntax

property InsertSQL: TStrings;

Remarks
Set the InsertSQL property to an INSERT INTO statement to use when inserting a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

Class
TCustomDAUpdateSQL

Syntax

property LockObject: TComponent;

Remarks
Assign a SQL component or TCustomVirtualQuery 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

4.3.1.6.2.7 LockSQL Property

Used to lock the current record.

Class
TCustomDAUpdateSQL

Syntax

```property``
LockSQL: TStrings;
```

Remarks
Use the LockSQL property to lock the current record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

4.3.1.6.2.8 ModifyObject Property

Provides ability to perform advanced adjustment of modify operations.

Class
TCustomDAUpdateSQL

Syntax

```property``
ModifyObject: TComponent;
```

Remarks
Assign a SQL component or TCustomVirtualQuery 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

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.

Class
TCustomDAUpdateSQL

Syntax

```property RefreshObject: TComponent;```

Remarks
Assign a SQL component or TCustomVirtualQuery 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.
4.3.1.6.2.11 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 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

- RefreshSQL

4.3.1.6.2.12 SQL Property(Indexer)

Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Class

TCustomDAUpdateSQL

Syntax

```
property SQL[UpdateKind: TUpdateKind]: TStrings;
```
**Parameters**

*UpdateKind*

Specifies which of update SQL statements to return.

**Remarks**

Returns a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties, depending on the value of the UpdateKind index.

```
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

Executes a SQL statement.

Class

TCustomDAUpdateSQL

Syntax

```pascal
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.
4.3.1.7  TDACondition Class

Represents a condition from the TDAConditions list.
For a list of all members of this type, see TDACondition members.

Unit
DBAccess

Syntax

TDACondition = class(TCollectionItem);

Remarks
Manipulate conditions using TDAConditions.

See Also
- TDAConditions

4.3.1.7.1  Members

TDACondition class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Indicates whether the condition is enabled or not</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the condition</td>
</tr>
<tr>
<td>Value</td>
<td>The value of the condition</td>
</tr>
</tbody>
</table>

Methods
4.3.1.7.2 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>Enable</td>
<td>Enables the condition</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables the condition</td>
</tr>
</tbody>
</table>

See Also
- TDACondition Class
- TDACondition Class Members

4.3.1.7.2.1 Enabled Property

Indicates whether the condition is enabled or not

Class

TDACondition

Syntax

```plaintext
property Enabled: Boolean default True;
```
4.3.1.7.2.2 Name Property

The name of the condition

Class

TDAClassCondition

Syntax

```pascal
property Name: string;
```

4.3.1.7.2.3 Value Property

The value of the condition

Class

TDAClassCondition

Syntax

```pascal
property Value: string;
```

4.3.1.7.3 Methods

Methods of the TDAClassCondition class.
For a complete list of the TDAClassCondition class members, see the TDAClassCondition 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

- TDAClassCondition Class
- TDAClassCondition Class Members
4.3.1.7.3.1 Disable Method

Disables the condition

Class

TDACCondition

Syntax

```pascal
procedure Disable;
```

4.3.1.7.3.2 Enable Method

Enables the condition

Class

TDACCondition

Syntax

```pascal
procedure Enable;
```

4.3.1.8 TDACConditions Class

Holds a collection of TDACCondition objects.
For a list of all members of this type, see TDACConditions members.

Unit

DBAccess

Syntax

```pascal
TDACConditions = class(TCollection);
```

Remarks
The given example code

```csharp
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
(SAL<2500)
```

### 4.3.1.8.1 Members

**TDACConditions** 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=CONDITON</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>
</tbody>
</table>
### 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><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Condition</strong></td>
<td>Used to iterate through all the conditions.</td>
</tr>
<tr>
<td><strong>Enabled</strong></td>
<td>Indicates whether the condition is enabled.</td>
</tr>
<tr>
<td><strong>Items</strong></td>
<td>Used to iterate through all conditions.</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The property returns condition names and values as <code>CONDITION_NAME=CONDITION</code></td>
</tr>
<tr>
<td><strong>WhereSQL</strong></td>
<td>Returns the SQL WHERE condition added in the</td>
</tr>
</tbody>
</table>
4.3.1.8.2.1  Condition Property (Indexer)

Used to iterate through all the conditions.

Class

TDAConditions

Syntax

```pascal
property Condition[Index: Integer]: TDACondition;
```

Parameters

Index

4.3.1.8.2.2  Enabled Property

Indicates whether the condition is enabled

Class

TDAConditions

Syntax

```pascal
property Enabled: Boolean;
```

© 1997-2019
Devart. All Rights Reserved.
4.3.1.8.2.3 Items Property (Indexer)

Used to iterate through all conditions.

Class
TDAConditions

Syntax

```powershell
property Items[Index: Integer]: TDACondition; default;
```

Parameters

INDEX
Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all conditions. Index identifies the index in the range 0..Count - 1. Items can reference a particular condition by its index, but the Condition property is preferred in order to avoid depending on the order of the conditions.

© 1997-2019
Devart. All Rights Reserved.

4.3.1.8.2.4 Text Property

The property returns condition names and values as CONDITION_NAME=CONDITION

Class
TDAConditions

Syntax

```powershell
property Text: string;
```

© 1997-2019
Devart. All Rights Reserved.

4.3.1.8.2.5 WhereSQL Property

Returns the SQL WHERE condition added in the Conditions property.

Class
TDAConditions
4.3.1.8.3 Methods

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Overloaded. Adds a condition to the WHERE clause of the query.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the condition</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables the condition</td>
</tr>
<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](#)
### 4.3.1.8.3.1 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

```plaintext
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:

\[
\text{SELECT * FROM EMP} \\
\text{WHERE (JOB="MANAGER")} \\
\text{and} \\
\text{(SAL<2500)}
\]

Adds a condition to the WHERE clause of the query.

Class

**TDAConditions**

Syntax

```pascal
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:

\[
\text{SELECT * FROM EMP} \\
\text{WHERE (JOB="MANAGER")} \\
\text{and} \\
\text{(SAL<2500)}
\]

4.3.1.8.3.2 Delete Method

Deletes the condition

Class

**TDAConditions**

Syntax
procedure Delete(Index: integer);

Parameters

Index
   Index of the condition

4.3.1.8.3.3 Disable Method

Disables the condition

Class
TDAConditions

Syntax

procedure Disable;

4.3.1.8.3.4 Enable Method

Enables the condition

Class
TDAConditions

Syntax

procedure Enable;

4.3.1.8.3.5 Find Method

Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.

Class
TDAConditions
4.3.1.8.3.6 Get Method

Retrieving a TDACondition object by its name. If found, the TDACondition object is returned, otherwise - an exception is raised.

Class

TDAConditions

Syntax

```plaintext
function Find(const Name: string): TDACondition;
```

Parameters

Name

© 1997-2019 Devart. All Rights Reserved.

4.3.1.8.3.7 IndexOf Method

Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.

Class

TDAConditions

Syntax

```plaintext
function IndexOf(const Name: string): Integer;
```

Parameters

Name

© 1997-2019 Devart. All Rights Reserved.
4.3.1.8.3.8 Remove Method

Removes the condition

Class
TDAConditions

Syntax

```
procedure Remove(const Name: string);
```

Parameters

Name
Specifies the name of the removed condition

4.3.1.9 TDACConnectionOptions Class

This class allows setting up the behaviour of the TDACConnection class.
For a list of all members of this type, see TDACConnectionOptions members.

Unit
DBAccess

Syntax

```
TDACConnectionOptions = class(TPersistent);
```

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowImplicitConnect</td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
</tbody>
</table>
### Properties

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>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the <code>TMemDataSet.IndexFieldNames</code> property of a dataset.</td>
</tr>
<tr>
<td>DisconnectedMode</td>
<td>Used to open a connection only when needed for performing a server call and closes after performing the operation.</td>
</tr>
<tr>
<td>KeepDesignConnected</td>
<td>Used to prevent an application from establishing a connection at the time of startup.</td>
</tr>
<tr>
<td>LocalFailover</td>
<td>If True, the <code>TCustomDAConnection.OnConnectionLost</code> event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
</tbody>
</table>
### TDAConnectionOptions Class Members

#### AllowImplicitConnect Property

4.3.1.9.2.1  AllowImplicitConnect Property

Specifies whether to allow or not implicit connection opening.

**Class**

TDAConnectionOptions

**Syntax**

```delphi
property AllowImplicitConnect: boolean default True;
```

**Remarks**

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>
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.

4.3.1.9.2.2 DefaultSortType Property

Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.

Class
TDACConnectionOptions

Syntax

```
property DefaultSortType: TSortType default stCaseSensitive;
```

Remarks
Use the DefaultSortType property to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.

4.3.1.9.2.3 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 A:Work_DisconnectMode topic for more information.

Class
TDACallbackOptions

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.

Class
TDACallbackOptions

Syntax

property LocalFailover: boolean default False;

Remarks
If True, the TCustomDACConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

Class
TDACallbackOptions

Syntax
**property** LocalFailover: boolean **default** False;

Remarks

If True, the `TCustomDACConnection.OnConnectionLost` event occurs and a failover operation can be performed after connection breaks. Read the A:Unstable_Network topic for more information about using failover.

4.3.1.10 **TDADDataSetOptions** Class

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

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

Unit

**DBAccess**

Syntax

```plaintext
TDADDataSetOptions = class(TPersistent);
```

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoPrepare</strong></td>
<td>Used to execute automatic <code>TCustomDADataset.Prepare</code> on the query execution.</td>
</tr>
<tr>
<td><strong>CacheCalcFields</strong></td>
<td>Used to enable caching of the 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</td>
</tr>
<tr>
<td><strong>DetailDelay</strong></td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td><strong>FieldsOrigin</strong></td>
<td>Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td><strong>FlatBuffers</strong></td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td><strong>InsertAllSetFields</strong></td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td><strong>LocalMasterDetail</strong></td>
<td>Used for TCustomDADataset to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.</td>
</tr>
<tr>
<td><strong>LongStrings</strong></td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<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...</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>QuoteNames</code></td>
<td>Used for <code>TCustomDADataset</code> to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td><code>RemoveOnRefresh</code></td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td><code>RequiredFields</code></td>
<td>Used for <code>TCustomDADataset</code> to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td><code>ReturnParams</code></td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td><code>SetFieldsReadOnly</code></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><code>StrictUpdate</code></td>
<td>Used for <code>TCustomDADataset</code> to raise an exception when the number of updated or deleted records is not equal 1.</td>
</tr>
<tr>
<td><code>TrimFixedChar</code></td>
<td>Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
<tr>
<td><code>UpdateAllFields</code></td>
<td>Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td><code>UpdateBatchSize</code></td>
<td>Used to get or set a value that enables or disables batch processing support, and specifies the number of</td>
</tr>
</tbody>
</table>
4.3.1.10.2 Properties

Properties of the **TDADatasetOptions** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoPrepare</strong></td>
<td>Used to execute automatic 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>DefaultValue</strong></td>
<td>Used to request default values/expressions from the server and assign them to the DefaultExpression property.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</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 additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td>RemoveOnRefresh</td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td>RequiredFields</td>
<td>Used for TCustomDADataSet to set</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ReturnParams</td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td>SetFieldsReadOnly</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>StrictUpdate</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>TrimFixedChar</td>
<td>Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
<tr>
<td>UpdateAllFields</td>
<td>Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td>UpdateBatchSize</td>
<td>Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.</td>
</tr>
</tbody>
</table>

See Also
- **TDADatasetOptions Class**
- **TDADatasetOptions Class Members**

Used to execute automatic TCustomDADataSet.Prepare on the query execution.

Class

**TDADatasetOptions**
Syntax

```
property AutoPrepare: boolean default False;
```

Remarks
Use the AutoPrepare property to execute automatic `TCustomDADataset.Prepare` on
the query execution. Makes sense for cases when a query will be executed several
times, for example, in Master/Detail relationships.

Class
`TDADataSetOptions`

Syntax

```
property CacheCalcFields: boolean default False;
```

Remarks
Use the CacheCalcFields property to enable caching of the `TField.Calculated` and
`TField.Lookup` fields. It can be useful for reducing CPU usage for calculated fields.
Using caching of calculated and lookup fields increases memory usage on the client side.

Class
`TDADataSetOptions`

Syntax

```
property CompressBlobMode: TCompressBlobMode default cbNone;
```

Remarks
Used to store values of the BLOB fields in compressed form.
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.

4.3.1.10.2.4 DefaultValues Property

Used to request default values/expressions from the server and assign them to the DefaultExpression property.

Class
TDADatasetOptions

Syntax

```pascal
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.

4.3.1.10.2.5 DetailDelay Property

Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.

Class
TDADatasetOptions

Syntax

```pascal
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.

4.3.1.10.2.6 FieldsOrigin Property

Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.

Class
TDADatasetOptions

Syntax

```delphi
property FieldsOrigin: boolean;
```

Remarks
If True, TCustomDADataSet fills the Origin property of the TField objects by appropriate value when opening a dataset.

4.3.1.10.2.7 FlatBuffers Property

Used to control how a dataset treats data of the ftString and ftVarBytes fields.

Class
TDADatasetOptions

Syntax

```delphi
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.
4.3.1.10.2.8  InsertAllSetFields Property

Used to include all set dataset fields in the generated INSERT statement.

Class

TDADatasetOptions

Syntax

property InsertAllSetFields: boolean default False;

Remarks

If True, all set dataset fields, including those set to NULL explicitly, will be included in the generated INSERT statements. Otherwise, only set fields containing not NULL values will be included to the generated INSERT statement.

4.3.1.10.2.9  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.
### 4.3.1.10.2.10 LongStrings Property

Used to represent string fields with the length that is greater than 255 as TStringField.

**Class**

`TDADatasetOptions`

**Syntax**

```delphi
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.

### 4.3.1.10.2.11 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**

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

### 4.3.1.10.2.12 NumberRange Property

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

**Class**

`TDADatasetOptions`
NumberRange Property

Syntax

```pascal
property NumberRange: boolean default False;
```

Remarks

Use the NumberRange property to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Request Support    DAC Forum    Provide Feedback

QueryRecCount Property

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.

QuoteNames Property

Syntax

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

Remarks

Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
Remarks
If True, TCustomDADataSet quotes all database object names in autogenerated SQL statements such as update SQL.

Class
TCustomDADataSet

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.

Class
TCustomDADataSet

Syntax

property RequiredFields: boolean default True;

Remarks
Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
If True, TCustomDADeveloper will set the Required property of the TField objects for the NOT NULL fields. It is useful when the table has a trigger which updates the NOT NULL fields.

**4.3.1.10.2.17 ReturnParams Property**

Used to return the new value of fields to the dataset after insert or update.

**Class**

`TDADeveloperOptions`

**Syntax**

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

**Remarks**

Use the ReturnParams property to return the new value of fields to the dataset after insert or update. The actual value of a 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 are assigned to the corresponding fields.

**4.3.1.10.2.18 SetFieldsReadOnly Property**

Used for a dataset to set the ReadOnly property to True for all fields that do not belong to the `UpdatingTable` or can not be updated.

**Class**

`TDADeveloperOptions`

**Syntax**

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

**Remarks**

If True, the dataset will set the ReadOnly property to True for all fields that do not belong to the `UpdatingTable` or can not be updated. This option is set for datasets that use automatic generation of the update SQL statements only.
4.3.1.10.2.19  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.

4.3.1.10.2.20  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.
4.3.1.10.2.21 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.

4.3.1.10.2.22 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.
4.3.1.11 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.

4.3.1.11.1 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</td>
</tr>
</tbody>
</table>
### Properties of the TDAMapRule class.

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

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>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>
</tbody>
</table>
### FieldType

Delphi field type, that the specified DB type or DataSet field will be mapped to.

### IgnoreErrors

Ignoring errors when converting data from DB to Delphi type.

---

#### 4.3.1.11.2.1 DBLengthMax Property

Maximum DB field length, until which the rule is applied.

**Class**

TDAMapRule

**Syntax**

```delphi
property DBLengthMax default rlAny;
```

**Remarks**

Setting maximum DB field length, until which the rule is applied to the specified DB field.

---

#### 4.3.1.11.2.2 DBLengthMin Property

Minimum DB field length, starting from which the rule is applied.

**Class**

TDAMapRule

**Syntax**

```delphi
property DBLengthMin default rlAny;
```
Remarks
Setting minimum DB field length, starting from which the rule is applied to the specified DB field.

Class
TDAMapRule

Syntax

property DBScaleMax default rlAny;

Remarks
Setting maximum DB field scale, until which the rule is applied to the specified DB field.

Class
TDAMapRule

Syntax

property DBScaleMin default rlAny;

Remarks
Setting minimum DB field Scale, starting from which the rule is applied to the specified DB field.
4.3.1.11.2.5 DbType Property

DB field type, that the rule is applied to.

Class
TDAMapRule

Syntax

```delphi
property DbType default dtUnknown;
```

Remarks
Setting DB field type, that the rule is applied to. If the current rule is set for Connection, the
rule will be applied to all fields of the specified type in all DataSets related to this Connection.

4.3.1.11.2.6 FieldLength Property

The resultant field length in Delphi.

Class
TDAMapRule

Syntax

```delphi
property FieldLength default rlAny;
```

Remarks
Setting the Delphi field length after conversion.

4.3.1.11.2.7 FieldName Property

DataSet field name, for which the rule is applied.

Class
TDAMapRule

Syntax
### FieldName Property

**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.

### FieldScale Property

**Class**

`TDAMapRule`

**Syntax**

```delphi
property FieldScale default rlAny;
```

**Remarks**

Setting the Delphi field Scale after conversion.

### FieldType Property

**Class**

`TDAMapRule`

**Syntax**

```delphi
property FieldType: TFieldType stored IsFieldTypeStored default ftUnknown;
```

**Remarks**

Setting Delphi field type, that the specified DB type or DataSet field will be mapped to.
4.3.1.11.2.10 IgnoreErrors Property

Ignoring errors when converting data from DB to Delphi type.

Class

TDAMapRule

Syntax

```property IgnoreErrors 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.

4.3.1.12 TDAMapRules Class

Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types. For a list of all members of this type, see TDAMapRules members.

Unit

DBAccess

Syntax

```TDAMapRules = class(TMapRules);```
4.3.1.12.2 Properties

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

4.3.1.12.2.1 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 `TDAMapRule.IgnoreErrors` property.

See Also
- `TDAMapRule.IgnoreErrors`

4.3.1.13 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

```delphi
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:

```delphi
MetaData.Connection := Connection;
MetaData.MetaDataKind := 'Columns';
MetaData.Restrictions.Values['TABLE_NAME'] := 'Emp';
MetaData.Open;
```

Inheritance Hierarchy

`TMemDataSet`
TDAMetaData

See Also
- TDAMetaData.MetaDataKind
- TDAMetaData.Restrictions
- TDAMetaData.GetMetaDataKinds
- TDAMetaData.GetRestrictions

4.3.1.13.1 Members

**TDAMetaData** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from <strong>TMemDataSet</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MetaDataKind</td>
<td>Used to specify which kind of metainformation to show.</td>
</tr>
<tr>
<td>Prepared</td>
<td>(inherited from <strong>TMemDataSet</strong>) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</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>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>GetMetaDataKinds</strong></td>
<td>Used to get values acceptable in the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetRestrictions</td>
<td>MetadataKind property. 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>
<tr>
<td>UnPrepare (inherited from TMemDataSet)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
</tbody>
</table>
### Properties 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>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>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> (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</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>lower boundaries for a range.</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>------------------------------</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>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**

4.3.1.13.2.1 **Connection Property**

Used to specify a connection object to use to connect to a data store.

**Class**

**TDAMetaData**

**Syntax**
property Connection: TCustomDAConnection;

Remarks
Use the Connection property to specify a connection object to use to connect to a data store. Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects. At runtime, set the Connection property to reference an instanciated TCustomDAConnection object.

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 MetaDataKind property is an empty string</td>
</tr>
<tr>
<td>ProcedureParameters</td>
<td>show metainformation about parameters of existing procedures</td>
</tr>
<tr>
<td>Procedures</td>
<td>show metainformation about existing procedures</td>
</tr>
<tr>
<td>Restrictions</td>
<td>generates a dataset that describes which restrictions are applicable to each MetaDataKind</td>
</tr>
</tbody>
</table>
### 4.3.1.13.2.3 Restrictions Property

#### Used to provide one or more conditions restricting the list of objects to be described.

#### Class

**TDAMetaData**

#### Syntax

```delphi
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**

---

### 4.3.1.13.3 Methods

Methods of the **TDAMetaData** class.

For a complete list of the **TDAMetaData** class members, see the **TDAMetaData Members** topic.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>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>DeferredPost (inherited from TMemDataSet)</td>
<td>Makes permanent changes to the database server.</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>GetBlob (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>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</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>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
- TDAMetaData Class
- TDAMetaData Class Members
4.3.1.13.3.1 GetMetaDataKinds Method

Used to get values acceptable in the MetaDataKind property.

Class

**TDAMetaData**

Syntax

```delphi
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**

4.3.1.13.3.2 GetRestrictions Method

Used to find out which restrictions are applicable to a certain MetaDataKind.

Class

**TDAMetaData**

Syntax

```delphi
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`

---

### 4.3.1.14 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**

```plaintext
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`
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsBlob</td>
<td>Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td>AsBlobRef</td>
<td>Used to set and read the value of the BLOB parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to assign the value for a float field to a parameter.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to assign the value for an integer field to the parameter.</td>
</tr>
<tr>
<td>AsLargeInt</td>
<td>Used to assign the value for a LargeInteger field to the parameter.</td>
</tr>
<tr>
<td>AsMemo</td>
<td>Used to assign the value for a memo field to the parameter.</td>
</tr>
<tr>
<td>AsMemoRef</td>
<td>Used to set and read the value of the memo parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsSQLTimeStamp</td>
<td>Used to specify the value of the parameter when it represents a SQL 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 parameter is null.</td>
</tr>
</tbody>
</table>
value assigned to a parameter is NULL.

**ParamType**

Used to indicate the type of use for a parameter.

**Size**

Specifies the size of a string type parameter.

**Value**

Used to represent the value of the parameter as Variant.

### Methods

<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>

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsBlob</strong></td>
<td>Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td><strong>AsBlobRef</strong></td>
<td>Used to set and read the value of the BLOB</td>
</tr>
</tbody>
</table>

© 1997-2019 Devart. All Rights Reserved.  
© 2019 Devart
### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td><strong>DataType</strong></td>
<td>Indicates the data type of the parameter.</td>
</tr>
</tbody>
</table>

### See Also
4.3.1.14.2.1 AsBlob Property

Used to set and read the value of the BLOB parameter as string.

Class
TDAParam

Syntax

```delphi
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.

4.3.1.14.2.2 AsBlobRef Property

Used to set and read the value of the BLOB parameter as a TBlob object.

Class
TDAParam

Syntax

```delphi
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.
4.3.1.14.2.3 AsFloat Property

Used to assign the value for a float field to a parameter.

Class

TDAParam

Syntax

property AsFloat: double;

Remarks

Use the AsFloat property to assign the value for a float field to the parameter. Setting AsFloat will set the DataType property to dtFloat.

Read the AsFloat property to determine the value that was assigned to an output parameter, represented as Double. The value of the parameter will be converted to the Double value if possible.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.14.2.4 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.

© 1997-2019 Devart. All Rights Reserved.
4.3.1.14.2.5  AsLargeInt Property

Used to assign the value for a LargeInteger field to the parameter.

Class

`TDAParam`

Syntax

```delphi
property AsLargeInt: Int64;
```

Remarks

Set the `AsLargeInt` property to assign the value for an `Int64` field to the parameter. Setting `AsLargeInt` will set the `DataType` property to `dtLargeint`.

Read the `AsLargeInt` property to determine the value that was assigned to an output parameter, represented as a 64-bit integer. The value of the parameter will be converted to the `Int64` value if possible.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.14.2.6  AsMemo Property

Used to assign the value for a memo field to the parameter.

Class

`TDAParam`

Syntax

```delphi
property AsMemo: string;
```

Remarks

Use the `AsMemo` property to assign the value for a memo field to the parameter. Setting `AsMemo` will set the `DataType` property to `ftMemo`.

© 1997-2019 Devart. All Rights Reserved.
4.3.1.14.2.7 AsMemoRef Property

Used to set and read the value of the memo parameter as a TBlob object.

Class

TDAParam

Syntax

property AsMemoRef: TBlob;

Remarks

Use the AsMemoRef property to set and read the value of the memo parameter as a TBlob object. Setting AsMemoRef will set the DataType property to ftMemo.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.14.2.8 AsSQLTimeStamp Property

Used to specify the value of the parameter when it represents a SQL timestamp field.

Class

TDAParam

Syntax

property AsSQLTimeStamp: TSQLTimeStamp;

Remarks

Set the AsSQLTimeStamp property to assign the value for a SQL timestamp field to the parameter. Setting AsSQLTimeStamp sets the DataType property to ftTimeStamp.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.14.2.9 AsString Property

Used to assign the string value to the parameter.

Class

TDAParam
4.3.1.14.2.10  AsWideString Property

Used to assign the Unicode string value to the parameter.

Class

TDAParam

Syntax

```delphi
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.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.14.2.11  DataType Property

Indicates the data type of the parameter.

Class

TDAParam

Syntax

```delphi
property DataType: TDAParam.DataType;
```
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.

4.3.1.14.2.12  IsNull Property

Used to indicate whether the value assigned to a parameter is NULL.

Class
TDAParam

Syntax
property IsNull: boolean;

Remarks
Use the IsNull property to indicate whether the value assigned to a parameter is NULL.

4.3.1.14.2.13  ParamType Property

Used to indicate the type of use for a parameter.

Class
TDAParam

Syntax
property ParamType default DB . ptUnknown;

Remarks
Objects that use TDAParam objects to represent field parameters set ParamType to indicate the type of use for a parameter.
To learn the description of TParamType refer to Delphi Help.

4.3.1.14.2.14  Size Property

Specifies the size of a string type parameter.

Class

TDAParam

Syntax

(property size: integer default 0;

Remarks

Use the Size property to indicate the maximum number of characters the parameter may contain. Use the Size property only for Output parameters of the ftString, ftFixedChar, ftBytes, ftVarBytes, or ftWideString type.

4.3.1.14.2.15  Value Property

Used to represent the value of the parameter as Variant.

Class

TDAParam

Syntax

(property value: variant stored IsValueStored;

Remarks

The Value property represents the value of the parameter as Variant.

Use Value in generic code that manipulates the values of parameters without the need to know the field type the parameter represent.
4.3.1.14.3 Methods

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignField</td>
<td>Assigns field name and field value to a param.</td>
</tr>
<tr>
<td>AssignFieldValue</td>
<td>Assigns the specified field properties and value to a parameter.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Places the content of a specified file into a TDAParam object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Places the content from a stream into a TDAParam object.</td>
</tr>
<tr>
<td>SetBlobData</td>
<td>Overloaded. Writes the data from a specified buffer to BLOB.</td>
</tr>
</tbody>
</table>

See Also
- TDAParam Class
- TDAParam Class Members

© 1997-2019 Devart. All Rights Reserved.

4.3.1.14.3.1 AssignField Method

Assigns field name and field value to a param.

Class

TDAParam

Syntax

```
procedure AssignField(Field: TField);
```

Parameters

- **Field**
  
  Holds the field which name and value should be assigned to the param.
Remarks
Call the AssignField method to assign field name and field value to a param.

4.3.1.14.3.2 AssignFieldValue Method

Assigns the specified field properties and value to a parameter.

Class
TDAParam

Syntax

```plaintext
procedure AssignFieldValue(Field: TField; const Value: Variant);
```

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.

4.3.1.14.3.3 LoadFromFile Method

Places the content of a specified file into a TDAParam object.

Class
TDAParam

Syntax

```plaintext
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
See Also
- `LoadFromFile`

© 1997-2019 Devart. All Rights Reserved.

4.3.14.3.5 SetBlobData Method

Writes the data from a specified buffer to BLOB.

Class

`TDAParam`

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SetBlobData(Buffer: TValueBuffer)</code></td>
<td>Writes the data from a specified buffer to BLOB.</td>
</tr>
<tr>
<td><code>SetBlobData(Buffer: IntPtr; Size: Integer)</code></td>
<td>Writes the data from a specified buffer to BLOB.</td>
</tr>
</tbody>
</table>

© 1997-2019 Devart. All Rights Reserved.

Writes the data from a specified buffer to BLOB.

Class

`TDAParam`

Syntax

```delphi
procedure SetBlobData(Buffer: TValueBuffer); overload;
```

Parameters

`Buffer`

Holds the pointer to the data.

© 1997-2019 Devart. All Rights Reserved.

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.

---

**4.3.1.15 TDAPrams Class**

This class is used to manage a list of TDAPram objects for an object that uses field parameters.

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

**Unit**

`DBAccess`

**Syntax**

```plaintext
TDAPrams = class(TParams);
```

**Remarks**

Use TDAPrams to manage a list of TDAPram objects for an object that uses field parameters. For example, TCustomDADataSet objects and TCustomDASQL objects use TDAPrams objects to create and access their parameters.

**See Also**

- [TCustomDADataSet.Params](#)
- [TCustomDASQL.Params](#)
- [TDAPram](#)
4.3.1.15.1 Members

**TDAParams** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all parameters.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindParam</td>
<td>Searches for a parameter with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Searches for a parameter with the specified name.</td>
</tr>
</tbody>
</table>

4.3.1.15.2 Properties

Properties of the **TDAParams** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all parameters.</td>
</tr>
</tbody>
</table>

See Also

- **TDAParams Class**
- **TDAParams Class Members**
4.3.1.15.2.1 Items Property(Indexer)

Used to iterate through all parameters.

**Class**

`TDAParams`

**Syntax**

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

**Parameters**

- **Index**
  
  Holds an index in the range 0..Count - 1.

**Remarks**

Use the `Items` property to iterate through all parameters. Index identifies the index in the range 0..Count - 1. `Items` can reference a particular parameter by its index, but the `ParamByName` method is preferred in order to avoid depending on the order of the parameters.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.15.3 Methods

Methods of the `TDAParams` class.

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

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FindParam</code></td>
<td>Searches for a parameter with the specified name.</td>
</tr>
<tr>
<td><code>ParamByName</code></td>
<td>Searches for a parameter with the specified name.</td>
</tr>
</tbody>
</table>

**See Also**

- `TDAParams Class`
- `TDAParams Class Members`
4.3.1.15.3.1 FindParam Method

Searches for a parameter with the specified name.

Class

TDAParams

Syntax

function FindParam(const Value: string): TDAParam;

Parameters

Value
Holds the parameter name.

Return Value
a parameter, if a match was found. Nil otherwise.

Remarks

Use the FindParam method to find a parameter with the name passed in Value. If a match is found, FindParam returns the parameter. Otherwise, it returns nil. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate more than one parameter at a time by name, use the GetParamList method instead. To get only the value of a named parameter, use the ParamValues property.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.15.3.2 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.

© 2019 Devart
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.

4.3.1.16 TMacro Class
Object that represents the value of a macro.
For a list of all members of this type, see TMacro members.

Unit
DBAccess

Syntax
TMacro = class(TCollectionItem);

Remarks
TMacro object represents the value of a macro. Macro is a variable that holds string value. You just insert & MacroName in a SQL query text and change the value of macro by the Macro property editor at design time or the Value property at run time. At the time of opening query macro is replaced by its value.
If by any reason it is not convenient for you to use the ' & ' symbol as a character of macro replacement, change the value of the MacroChar variable.

See Also
• TMacros
4.3.1.16.1 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>

© 1997-2019 Devart. All Rights Reserved.

4.3.1.16.2 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>
<tr>
<td>AsString</td>
<td>Used to assign the string value to a macro.</td>
</tr>
</tbody>
</table>

Published
### Active Property

Used to determine if the macro should be expanded.

**Class**

**TMacro**

**Syntax**

```delphi
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**

```delphi
VirtualQuery.SQL.Text := 'SELECT * FROM Dept WHERE DeptNo > 20 &Cond1';
VirtualQuery.Macros[0].Value := 'and DName is NULL';
VirtualQuery.Macros[0].Active := False;
```
4.3.1.16.2.2 AsDateTime Property

Used to set the TDateTime value to a macro.

Class

TMacro

Syntax

```pascal
property AsDateTime: TDateTime;
```

Remarks

Use the AsDateTime property to set the TDateTime value to a macro.

4.3.1.16.2.3 AsFloat Property

Used to set the float value to a macro.

Class

TMacro

Syntax

```pascal
property AsFloat: double;
```

Remarks

Use the AsFloat property to set the float value to a macro.

4.3.1.16.2.4 AsInteger Property

Used to set the integer value to a macro.

Class

TMacro

Syntax

```pascal
property AsInteger: integer;
```
Remarks
Use the AsInteger property to set the integer 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.

Class
TMacro

Syntax

```
property Name: string;
```

Remarks
Use the Name property to identify a particular macro.
4.3.1.16.2.7 Value Property

Used to set the value to a macro.

Class

TMacro

Syntax

```property value: string;```

Remarks

Use the Value property to set the value to a macro.

4.3.1.17 TMacros Class

Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataset components.

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

Unit

DBAccess

Syntax

```TMacros = class(TCollection);```

Remarks

Use TMacros to manage a list of TMacro objects for the TCustomDASQL or TCustomDADataset components.

See Also

• TMacro
4.3.1.17.1 Members

**TMacros** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all the macros parameters.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignValues</td>
<td>Copies the macros values and properties from the specified source.</td>
</tr>
<tr>
<td>Expand</td>
<td>Changes the macros in the passed SQL statement to their values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Searches for a TMacro object by its name.</td>
</tr>
<tr>
<td>IsEqual</td>
<td>Compares itself with another TMacro object.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Used to search for a macro with the specified name.</td>
</tr>
<tr>
<td>Scan</td>
<td>Creates a macros from the passed SQL statement.</td>
</tr>
</tbody>
</table>

4.3.1.17.2 Properties

Properties of the **TMacros** class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all the macros parameters.</td>
</tr>
</tbody>
</table>

See Also
• **TMacros Class**

• **TMacros Class Members**

© 1997-2019
Devart. All Rights Reserved.

### 4.3.1.17.2.1 Items Property(Indexer)

Used to iterate through all the macros parameters.

**Class**

**TMacros**

**Syntax**

```plaintext
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.

© 1997-2019
Devart. All Rights Reserved.

### 4.3.1.17.3 Methods

Methods of the **TMacros** class.

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

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignValues</td>
<td>Copies the macros values and properties from the specified source.</td>
</tr>
<tr>
<td>Expand</td>
<td>Changes the macros in the passed SQL statement to their values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Searches for a TMacro object by its name.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsEqual</td>
<td>Compares itself with another TMacro object.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Used to search for a macro with the specified name.</td>
</tr>
<tr>
<td>Scan</td>
<td>Creates a macros from the passed SQL statement.</td>
</tr>
</tbody>
</table>

See Also
- TMacros Class
- TMacros Class Members

### AssignValues Method

Copies the macros values and properties from the specified source.

**Class**

TMacros

**Syntax**

```pascal
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.

### Expand Method

Changes the macros in the passed SQL statement to their values.

**Class**

TMacros
Syntax

```pascal
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.

4.3.1.17.3.3 FindMacro Method

Searches for a TMacro object by its name.

Class

**TMacros**

Syntax

```pascal
function FindMacro(const Value: string): TMacro;
```

Parameters

**Value**

Holds the value of a macro to search for.

Return Value

TMacro object if a match was found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the name passed in Value. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.
4.3.1.17.3.4 IsEqual Method

Compares itself with another TMacro object.

Class

TMacros

Syntax

function IsEqual(Value: TMacros): boolean;

Parameters

Value

Holds the values of TMacro objects.

Return Value

True, if the number of TMacro objects and the values of all TMacro objects are equal.

Remarks

Call the IsEqual method to compare itself with another TMacro object. Returns True if the number of TMacro objects and the values of all TMacro objects are equal.

© 1997-2019 Devart. All Rights Reserved.

4.3.1.17.3.5 MacroByName Method

Used to search for a macro with the specified name.

Class

TMacros

Syntax

function MacroByName(const Value: string): TMacro;

Parameters

Value

Holds a name of the macro to search for.

Return Value

TMacro object, if a macro with specified name was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Value. If a match is
found, MacroByName returns the Macro. Otherwise, an exception is raised. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

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

4.3.1.17.3.6 Scan Method

Creates a macros from the passed SQL statement.

Class

\texttt{TMacros}

Syntax

\texttt{procedure Scan(const SQL: string);}

Parameters

\texttt{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.

4.3.1.18 TPoolingOptions Class

This class allows setting up the behaviour of the connection pool.

For a list of all members of this type, see \texttt{TPoolingOptions} members.

Unit

\texttt{DBAccess}

Syntax

\texttt{TPoolingOptions = class(TPersistent);}
4.3.1.18.1 Members

**TPoolingOptions** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionLifetime</td>
<td>Used to specify the maximum time during which an opened connection can be used by connection pool.</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>Used to specify the maximum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td>MinPoolSize</td>
<td>Used to specify the minimum number of 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>

4.3.1.18.2 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>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>
</tbody>
</table>
4.3.1.18.2.1 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.
4.3.1.18.2.2 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.

4.3.1.18.2.3 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.

4.3.1.18.2.4 Validate Property

Used for a connection to be validated when it is returned from the pool.

Class

TPoolingOptions
**Syntax**

```plaintext
property Validate: boolean default DefValValidate;
```

**Remarks**

If the Validate property is set to True, connection will be validated when it is returned from the pool. By default this option is set to False and pool does not validate connection when it is returned to be used by a TCustomDACConnection component.

---

### 4.3.1.19 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**

```plaintext
TSmartFetchOptions = class(TPersistent);
```

---

#### 4.3.1.19.1 Members

[TSmartFetchOptions](#) class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Sets SmartFetch mode enabled or not.</td>
</tr>
<tr>
<td>LiveBlock</td>
<td>Used to minimize memory consumption.</td>
</tr>
<tr>
<td>PrefetchedFields</td>
<td>List of fields additional to key fields that will be read from the database on dataset open.</td>
</tr>
<tr>
<td>SQLGetKeyValues</td>
<td>SQL query for the read key</td>
</tr>
</tbody>
</table>
4.3.1.19.2  Properties

Properties of the **TSmartFetchOptions** class.

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

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enabled</strong></td>
<td>Sets SmartFetch mode enabled or not.</td>
</tr>
<tr>
<td><strong>LiveBlock</strong></td>
<td>Used to minimize memory consumption.</td>
</tr>
<tr>
<td><strong>PrefetchedFields</strong></td>
<td>List of fields additional to key fields that will be read from the database on dataset open.</td>
</tr>
<tr>
<td><strong>SQLGetKeyValues</strong></td>
<td>SQL query for the read key and prefetched fields from the database.</td>
</tr>
</tbody>
</table>

### See Also

- **TSmartFetchOptions Class**
- **TSmartFetchOptions Class Members**

© 1997-2019 Devart. All Rights Reserved.

4.3.1.19.2.1  Enabled Property

Sets SmartFetch mode enabled or not.

**Class**

**TSmartFetchOptions**

**Syntax**
4.3.1.19.2.2 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.

4.3.1.19.2.3 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.
4.3.1.19.2.4 SQLGetKeyValues Property

SQL query for the read key and prefetched fields from the database.

Class

TSmartFetchOptions

Syntax

```pascal
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.

4.3.2 Types

Types in the DBAccess unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAfterExecuteEvent</td>
<td>This type is used for the TCustomDADataset.AfterExecute and TCustomDASQL.AfterExecute events.</td>
</tr>
<tr>
<td>TAfterFetchEvent</td>
<td>This type is used for the TCustomDADataset.AfterFetch event.</td>
</tr>
<tr>
<td>TBeforeFetchEvent</td>
<td>This type is used for the TCustomDADataset.Before Fetch event.</td>
</tr>
<tr>
<td>TConnectionLostEvent</td>
<td>This type is used for the TCustomDACConnection.OnConnectionLost event.</td>
</tr>
<tr>
<td>TDACConnectionErrorEvent</td>
<td>This type is used for the TCustomDACConnection.OnConnectionError event.</td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TDATransactionErrorEvent</td>
<td>This type is used for the E:Devart.Dac.TDATransaction.OnError 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 TCustomDADataset.AfterUpdateExecute and TCustomDADataset.BeforeUpdateExecute events.</td>
</tr>
</tbody>
</table>

### 4.3.2.1 TAfterExecuteEvent Procedure Reference

This type is used for the TCustomDADataset.AfterExecute and TCustomDASQL.AfterExecute events.

**Unit**

DBAccess

**Syntax**

```plaintext
TAfterExecuteEvent = procedure (Sender: TObject; Result: boolean) of object;
```

**Parameters**

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

- **Result**
  - The result is True if SQL statement is executed successfully. False otherwise.

© 1997-2019 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

### 4.3.2.2 TAfterFetchEvent Procedure Reference

This type is used for the TCustomDADataset.AfterFetch event.

**Unit**

DBAccess

© 2019 Devart
Syntax

TAfterFetchEvent = procedure (DataSet: TCustomDADataSet) of object;

Parameters

*DataSet*
Holds the TCustomDADataSet descendant to synchronize the record position with.

4.3.2.3 TBeforeFetchEvent Procedure Reference

This type is used for the TCustomDADataSet.BeforeFetch event.

Unit
DBAccess

Syntax

TBeforeFetchEvent = procedure (DataSet: TCustomDADataSet; var Cancel: boolean) of object;

Parameters

*DataSet*
Holds the TCustomDADataSet descendant to synchronize the record position with.

*Cancel*
True, if the current fetch operation should be aborted.

4.3.2.4 TConnectionLostEvent Procedure Reference

This type is used for the TCustomDACConnection.OnConnectionLost event.

Unit
DBAccess

Syntax

TConnectionLostEvent = procedure (Sender: TObject; Component: TComponent; ConnLostCause: TConnLostCause; var RetryMode:
4.3.2.5 TDAConnectionErrorEvent Procedure Reference

This type is used for the TCustomDAConnection.OnError event.

```
TDAConnectionErrorEvent = procedure (Sender: TObject; E: EDAError;
var Fail: boolean) of object;
```

**Parameters**

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

**E**
The error information.

**Fail**
False, if an error dialog should be prevented from being displayed and EAbort exception should be raised to cancel current operation.

© 1997-2019 Devart. All Rights Reserved.

4.3.2.6 TDATransactionErrorEvent Procedure Reference

This type is used for the E:Devart.Dac.TDATransaction.OnError event.

Unit

```
DBAccess
```

© 2019 Devart
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.

TRefreshOptions Set

Represents the set of TRefreshOption.

TUpdateExecuteEvent Procedure Reference

This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.
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.

4.3.3 Enumerations

Enumerations in the DBAccess unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCheckMode</td>
<td>Specifies the action to take when another user makes modifications to a record.</td>
</tr>
<tr>
<td>TLabelSet</td>
<td>Sets the language of labels in the connect dialog.</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>

4.3.3.1 TCheckMode Enumeration

Specifies the action to take when another user makes modifications to a record.

Unit
DBAccess

Syntax
TCheckMode = (cmNone, cmException, cmRefresh);
4.3.3.2 TLabelSet Enumeration

Sets the language of labels in the connect dialog.

Unit

DBAccess

Syntax

TLabelSet = (lsCustom, lsEnglish, lsFrench, lsGerman, lsItalian, lsPolish, lsPortuguese, lsRussian, lsSpanish);

Values

<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>
4.3.3.3 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>

4.3.3.4 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>rmReconnectExecute</td>
<td>Reconnect is performed and abortive operation is reexecuted. Exception is not raised.</td>
</tr>
</tbody>
</table>
4.3.4 Variables

Variables in the DBAccess unit.

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQLOldBehavior</td>
<td>After assigning SQL text and modifying it by AddWhere, DeleteWhere, and SetOrderBy, all subsequent changes of the SQL property will not be reflected in the BaseSQL property.</td>
</tr>
<tr>
<td>ChangeCursor</td>
<td>When set to True allows data access components to change screen cursor for the execution time.</td>
</tr>
<tr>
<td>SQLGeneratorCompatibility</td>
<td>The value of the TCustomDADataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause.</td>
</tr>
</tbody>
</table>

4.3.4.1 BaseSQLOldBehavior Variable

After assigning SQL text and modifying it by AddWhere, DeleteWhere, and SetOrderBy, all subsequent changes of the SQL property will not be reflected in the BaseSQL property.

Unit

DBAccess

Syntax

BaseSQLOldBehavior: boolean = False;

Remarks
The `BaseSQL` property is similar to the SQL property, but it does not store changes made by the `AddWhere`, `DeleteWhere`, and `SetOrderBy` methods. After assigning SQL text and modifying it by one of these methods, all subsequent changes of the SQL property will not be reflected in the `BaseSQL` property. This behavior was changed in VirtualDAC. To restore old behavior, set the `BaseSQLOldBehavior` variable to True.

### 4.3.4.2 ChangeCursor Variable

When set to True allows data access components to change screen cursor for the execution time.

**Unit**

`DBAccess`

**Syntax**

```pascal
ChangeCursor: boolean = True;
```

### 4.3.4.3 SQLGeneratorCompatibility Variable

The value of the `TCustomDADataSet.BaseSQL` property is used to complete the refresh SQL statement, if the manually assigned `TCustomDAUpdateSQL.RefreshSQL` property contains only WHERE clause.

**Unit**

`DBAccess`

**Syntax**

```pascal
SQLGeneratorCompatibility: boolean = False;
```

**Remarks**

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

© 1997-2019 Devart. All Rights Reserved.

4.4 MemData

4.4.1 Classes

Classes in the MemData unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAttribute</td>
<td>TAttribute is not used in VirtualDAC.</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 storing data in memory.</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>

© 1997-2019 Devart. All Rights Reserved.

4.4.1.1 TAttribute Class

TAttribute is not used in VirtualDAC.

For a list of all members of this type, see TAttribute members.
Unit

MemData

Syntax

```pascal
TAttribute = class(System TObject);
```

<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>
<tr>
<td>Scale</td>
<td>Returns the scale of dtFloat and dtInteger attributes.</td>
</tr>
<tr>
<td>Size</td>
<td>Returns the size of an attribute value in external representation.</td>
</tr>
</tbody>
</table>

© 1997-2019 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
### 4.4.1.1.2 Properties

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> attributes.</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Returns a <code>TObjectType</code> 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 <code>TObjectClass</code> 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>

**See Also**
- `TAttribute Class`
- `TAttribute Class Members`
4.4.1.1.2.1 AttributeNo Property

Returns an attribute's ordinal position in object.

Class

TAttribute

Syntax

```delphi
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

© 1997-2019 Devart. All Rights Reserved.

4.4.1.1.2.2 DataSize Property

Returns the size of an attribute value in internal representation.

Class

TAttribute

Syntax

```delphi
property DataSize: Integer;
```

Remarks

Use the DataSize property to learn the size of an attribute value in internal representation.

© 1997-2019 Devart. All Rights Reserved.
4.4.1.1.2.3  **DataType Property**

Returns the type of data that was assigned to the Attribute.

**Class**

**TAttribute**

**Syntax**

```pascal
property DataType: Word;
```

**Remarks**

Use the DataType property to discover the type of data that was assigned to the Attribute.
Possible values: dtDate, dtFloat, dtInteger, dtString, dtObject.

4.4.1.1.2.4 **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](#)
4.4.1.1.2.5  ObjectType Property

Returns a TObjectType object for an object attribute.

Class

TAttribute

Syntax

```property ObjectType: TObjectType;```

Remarks

Use the ObjectType property to return a TObjectType object for an object attribute.

4.4.1.1.2.6  Offset Property

Returns an offset of the attribute value in internal representation.

Class

TAttribute

Syntax

```property Offset: Integer;```

Remarks

Use the DataSize property to learn an offset of the attribute value in internal representation.

4.4.1.1.2.7  Owner Property

Indicates TObjectType that uses the attribute to represent one of its attributes.

Class

TAttribute

Syntax

```property Owner: TObjectType;```

© 1997-2019 Devart. All Rights Reserved.
Remarks
Check the value of the Owner property to determine TObjectType that uses the attribute to represent one of its attributes. Applications should not assign the Owner property directly.

Class
TAttribute

Syntax

```plaintext
property Scale: Word;
```

Remarks
Use the Scale property to learn the scale of dtFloat and dtInteger attributes.

See Also
• Length

4.4.1.1.2.9 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:
### TBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types. For a list of all members of this type, see TBlob members.

**Unit**

MemData

**Syntax**

```
TBlob = class(TSharedObject);
```

**Remarks**

Object TBlob holds large object value for the field and parameter dtBlob, dtMemo, dtWideMemo data types.

**Inheritance Hierarchy**

TSharedObject

TBlob

**See Also**

- [TMemDataSet.GetBlob](#)
4.4.1.2.1 Members

**TBlob** class overview.

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

<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>Assign</td>
<td>Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Copies the contents of a stream into the TBlob object.</td>
</tr>
<tr>
<td>Read</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlob.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the 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</td>
</tr>
</tbody>
</table>
4.4.1.2.2 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>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>

See Also
- TBlob Class
- TBlob Class Members

© 1997-2019 Devart. All Rights Reserved.
**TBlob**

**Syntax**

```delphi
property AsString: string;
```

**Remarks**

Use the AsString property to manipulate BLOB value as string.

**See Also**

- `Assign`
- `AsWideString`

---

4.4.1.2.2.2  **AsWideString Property**

Used to manipulate BLOB value as Unicode string.

**Class**

**TBlob**

**Syntax**

```delphi
property AsWideString: string;
```

**Remarks**

Use the AsWideString property to manipulate BLOB value as Unicode string.

**See Also**

- `Assign`
- `AsString`

---

4.4.1.2.2.3  **IsUnicode Property**

Gives choice of making TBlob store and process data in Unicode format or not.
**TBlob**

### Syntax

**property** IsUnicode: boolean;

### Remarks

Set IsUnicode to True if you want TBlob to store and process data in Unicode format. **Note:** changing this property raises an exception if TBlob is not empty.

© 1997-2019 Devart. All Rights Reserved.

4.4.1.2.2.4 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.

© 1997-2019 Devart. All Rights Reserved.

4.4.1.2.3 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><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>Assign</strong></td>
<td>Sets BLOB value from</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes the current value in TBlog object.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Loads the contents of a file into a TBlog object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Copies the contents of a stream into the TBlog object.</td>
</tr>
<tr>
<td>Read</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlog.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
<tr>
<td>SaveToFile</td>
<td>Saves the contents of the TBlog object to a file.</td>
</tr>
<tr>
<td>SaveToStream</td>
<td>Copies the contents of a TBlog object to a stream.</td>
</tr>
<tr>
<td>Truncate</td>
<td>Sets new TBlog size and discards all data over it.</td>
</tr>
<tr>
<td>Write</td>
<td>Stores a raw sequence of bytes into a TBlog object.</td>
</tr>
</tbody>
</table>

See Also
- TBlog Class
- TBlog Class Members

4.4.1.2.3.1 Assign Method

Sets BLOB value from another TBlog object.

Class

TBlog

Syntax

```pascal
procedure Assign(Source: TBlog);
```

Parameters

Source

Holds the BLOB from which the value to the current object will be assigned.
Remarks
Call the Assign method to set BLOB value from another TBlob object.

See Also
- LoadFromStream
-AsString
-AsWideString

4.4.1.2.3.2 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.

4.4.1.2.3.3 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

4.4.1.2.3.4 LoadFromStream Method
Copies the contents of a stream into the TBlob object.

Class
TBlob

Syntax

```plaintext
procedure LoadFromStream(Stream: TStream); virtual;
```

Parameters
Stream
Holds the specified stream from which the field's value is copied.

Remarks
Call the LoadFromStream method to copy the contents of a stream into the TBlob object. Specify the stream from which the field's value is copied as the value of the Stream parameter.

See Also
- SaveToStream

4.4.1.2.3.5 Read Method
Acquires a raw sequence of bytes from the data stored in TBlob.

Class
**TBlob**

**Syntax**

```pascal
function Read(Position: Cardinal; Count: Cardinal; Dest: IntPtr): Cardinal; virtual;
```

**Parameters**

- **Position**
  - Holds the starting point of the byte sequence.
- **Count**
  - Holds the size of the sequence in bytes.
- **Dest**
  - Holds a pointer to the memory area where to store the sequence.

**Return Value**

Actually read byte count if the sequence crosses object size limit.

**Remarks**

Call the Read method to acquire a raw sequence of bytes from the data stored in TBlob.
The Position parameter is the starting point of byte sequence which lasts Count number of bytes. The Dest parameter is a pointer to the memory area where to store the sequence. If the sequence crosses object size limit, function will return actually read byte count.

**See Also**

- Write

© 1997-2019 Devart. All Rights Reserved.

4.4.1.2.3.6 SaveToFile Method

Saves the contents of the TBlob object to a file.

**Class**

**TBlob**

**Syntax**

```pascal
procedure SaveToFile(const FileName: string);
```

**Parameters**

- **FileName**
Holds a string that contains the name of the file.

Remarks
Call the SaveToFile method to save the contents of the TBlob object to a file. Specify the name of the file as the value of the FileName parameter.

See Also
- LoadFromFile

4.4.1.2.3.7 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

4.4.1.2.3.8 Truncate Method
Sets new TBlob size and discards all data over it.

Class
**TBlob**

Syntax

```pascal
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.

---

**Write Method**

Stores a raw sequence of bytes into a TBlob object.

**Class**

**TBlob**

Syntax

```pascal
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

4.4.1.3 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; sourceLen: longint): longint;

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

Inheritance Hierarchy

TSharedObject
  TBlob
**TCompressedBlob**

See Also

- **TBlob**
- **TMemDataSet.GetBlob**
- **TCustomDADataset.Options**

© 1997-2019 Devart. All Rights Reserved.

### Members

**TCompressedBlob** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsString</strong></td>
<td>(inherited from <strong>TBlob</strong>) Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td><strong>AsWideString</strong></td>
<td>(inherited from <strong>TBlob</strong>) 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></td>
<td>(inherited from <strong>TBlob</strong>) Gives choice of making TBlob store and process data in Unicode format or not.</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>Size</strong></td>
<td>(inherited from <strong>TBlob</strong>) 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><strong>AddRef</strong></td>
<td>(inherited from <strong>TSharedObject</strong>) Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
</tbody>
</table>
### 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>
</tbody>
</table>

### Properties of the **TCompressedBlob** class.

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

© 1997-2019 Devart. All Rights Reserved.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>

See Also
- TCompressedBlob Class
- TCompressedBlob Class Members

4.4.1.3.2.1 Compressed Property

Used to indicate if the Blob is compressed.

Class
TCompressedBlob

Syntax

```delphi
property Compressed: boolean;
```

Remarks

Indicates whether the Blob is compressed. Set this property to True or False to compress or decompress the Blob.

4.4.1.3.2.2 CompressedSize Property

Used to indicate compressed size of the Blob data.

Class
TCompressedBlob
4.4.1.4 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.

Inheritance Hierarchy

TSharedObject
TDBObject

© 1997-2019 Devart. All Rights Reserved.

4.4.1.4.1 Members

TDBObject class overview.

Properties

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

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

4.4.1.5 TMemData Class

Implements storing data in memory.
For a list of all members of this type, see TMemData members.

Unit

MemData

Syntax

```
TMemData = class(TData);
```

Inheritance Hierarchy

TData
  TMemData

© 1997-2019
Devart. All Rights Reserved.

Request Support   DAC Forum   Provide Feedback
4.4.1.6 TObjectType Class

This class is not used.
For a list of all members of this type, see TObjectType members.

Unit
MemData

Syntax

TObjectType = class(TSharedObject);

Inheritance Hierarchy

TSharedObject
    TObjectType

4.4.1.6.1 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 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</td>
</tr>
</tbody>
</table>
4.4.1.6.2 Properties

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeCount</td>
<td>Used to indicate the number of attributes of type.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Used to access separate attributes.</td>
</tr>
<tr>
<td>DataType</td>
<td>Used to indicate the type of object dtObject, dtArray or dtTable.</td>
</tr>
<tr>
<td>RefCount (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
4.4.1.6.2.1  AttributeCount Property

Used to indicate the number of attributes of type.

Class

TObjectType

Syntax

```delphi
property AttributeCount: Integer;
```

Remarks

Use the AttributeCount property to determine the number of attributes of type.

4.4.1.6.2.2  Attributes Property(Indexer)

Used to access separate attributes.

Class

TObjectType

Syntax

```delphi
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
4.4.1.6.3 DataType Property

Used to indicate the type of object dtObject, dtArray or dtTable.

Class
TObjectType

Syntax

property DataType: Word;

Remarks
Use the DataType property to determine the type of object dtObject, dtArray or dtTable.

See Also
- T:Devart.Dac.Units.MemData

© 1997-2019
Devart. All Rights Reserved.

4.4.1.6.4 Size Property

Used to learn the size of an object instance.

Class
TObjectType

Syntax

property Size: Integer;

Remarks
Use the Size property to find out the size of an object instance. Size is a sum of all attribute sizes.

See Also
- TAttribute.Size

© 1997-2019
Devart. All Rights Reserved.
4.4.1.6.3 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</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</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

See Also

- TObjectType Class
- TObjectType Class Members

4.4.1.6.3.1 FindAttribute Method

Indicates whether a specified Attribute component is referenced in the TAttributes object.

Class

TObjectType

Syntax

```pascal
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
- MDac.TObject.TObjectType.AttributeByName(System.String)
- Attributes

4.4.1.7 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
4.4.1.7.1 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>

4.4.1.7.2 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**
4.4.1.7.1 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.

4.4.1.7.3 Methods

Methods of the TSharedObject class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

See Also

- TSharedObject Class
- TSharedObject Class Members

4.4.1.7.3.1 AddRef Method

Increments the reference count for the number of references dependent on the TSharedObject object.
Class
TSharedObject

Syntax

```delphi
procedure AddRef;
```

Remarks
Increments the reference count for the number of references dependent on the TSharedObject object.

See Also
- Release

4.4.1.7.3.2 Release Method

Decrements the reference count.

Class
TSharedObject

Syntax

```delphi
procedure Release;
```

Remarks
Call the Release method to decrement the reference count. When RefCount is 1, TSharedObject is deleted from memory.

See Also
- AddRef
4.4.2 Types

Types in the MemData unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLocateExOptions</td>
<td>Represents the set of TLocateExOption.</td>
</tr>
<tr>
<td>TUpdateRecKinds</td>
<td>Represents the set of TUpdateRecKind.</td>
</tr>
</tbody>
</table>

Syntax

```
TLocateExOptions = set of TLocateExOption;
```

4.4.2.1 TLocateExOptions Set

Represents the set of TLocateExOption.

Unit

MemData

Syntax

```
TLocateExOptions = set of TLocateExOption;
```

4.4.2.2 TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

MemData

Syntax

```
TUpdateRecKinds = set of TUpdateRecKind;
```
4.4.3 Enumerations

Enumerations in the MemData unit.

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>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>

4.4.3.1 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>
</table>
4.4.3.2 TConnLostCause Enumeration

Specifies the cause of the connection loss.

Unit
MemData

Syntax

TConnLostCause = (clUnknown, clExecute, clOpen, clRefresh, clApply, clServiceQuery, clTransStart, clConnectionApply, clConnect);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>clApply</td>
<td>Connection loss detected during DataSet.ApplyUpdates (Reconnect/Reexecute possible).</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>clConnect</code></td>
<td>Connection loss detected during connection establishing (Reconnect possible).</td>
</tr>
<tr>
<td><code>clConnectionApply</code></td>
<td>Connection loss detected during <code>Connection.ApplyUpdates</code> (Reconnect/Reexecute possible).</td>
</tr>
<tr>
<td><code>clExecute</code></td>
<td>Connection loss detected during SQL execution (Reconnect with exception is possible).</td>
</tr>
<tr>
<td><code>clOpen</code></td>
<td>Connection loss detected during execution of a SELECT statement (Reconnect with exception possible).</td>
</tr>
<tr>
<td><code>clRefresh</code></td>
<td>Connection loss detected during query opening (Reconnect/Reexecute possible).</td>
</tr>
<tr>
<td><code>clServiceQuery</code></td>
<td>Connection loss detected during service information request (Reconnect/Reexecute possible).</td>
</tr>
<tr>
<td><code>clTransStart</code></td>
<td>Connection loss detected during transaction start (Reconnect/Reexecute possible). <code>clTransStart</code> has less priority then <code>clConnectionApply</code>.</td>
</tr>
<tr>
<td><code>clUnknown</code></td>
<td>The connection loss reason is unknown.</td>
</tr>
</tbody>
</table>

© 1997-2019 Devart. All Rights Reserved.

4.4.3.3 TDANumericType Enumeration

Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.

Unit

MemData

Syntax

```
TDANumericType = (ntFloat, ntBCD, ntFmtBCD);
```

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ntBCD</code></td>
<td>Data is stored on the client side as currency and represented as <code>TBCDField</code>. This format allows storing data with precision up to 0,0001.</td>
</tr>
<tr>
<td><code>ntFloat</code></td>
<td>Data stored on the client side is in double format and represented as <code>TFloatField</code>. The default value.</td>
</tr>
<tr>
<td><code>ntFmtBCD</code></td>
<td>Data is represented as <code>TFMTBCDField</code>. <code>TFMTBCDField</code> gives greater precision and accuracy than <code>TBCDField</code>, but it is slower.</td>
</tr>
</tbody>
</table>

© 1997-2019 Devart. All Rights Reserved.
4.4.3.4 TLocateExOption Enumeration

Allows to set additional search parameters which will be used by the LocateEx method.

Unit
MemData

Syntax

TLocateExOption = (lxCaseInsensitive, lxPartialKey, lxNearest, lxNext, lxUp, lxPartialCompare);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lxCaseInsensitive</td>
<td>Similar to loCaseInsensitive. Key fields and key values are matched without regard to the case.</td>
</tr>
<tr>
<td>lxNearest</td>
<td>LocateEx moves the cursor to a specific record in a dataset or to the first record in the dataset that is greater than the values specified in the KeyValues parameter. For this option to work correctly dataset should be sorted by the fields the search is performed in. If dataset is not sorted, the function may return a line that is not connected with the search condition.</td>
</tr>
<tr>
<td>lxNext</td>
<td>LocateEx searches from the current record.</td>
</tr>
<tr>
<td>lxPartialCompare</td>
<td>Similar to lxPartialKey, but the difference is that it can process value entries in any position. For example, 'HAM' would match both 'HAMM', 'HAMMER.', and also 'MR HAMMER'.</td>
</tr>
<tr>
<td>lxPartialKey</td>
<td>Similar to 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>
## 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>

## 4.4.3.6 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>

## 4.5 MemDS

### 4.5.1 Classes

Classes in the MemDS unit.

### Classes
### 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

```delphi
TMemDataSet = class(TDataSet);
```

#### Remarks

TMemDataSet derives from the TDataSet database-engine independent set of properties, events, and methods for working with data and introduces additional techniques to store and manipulate data in memory.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a</td>
</tr>
</tbody>
</table>
### LocalConstraints
Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

### LocalUpdate
Used to prevent implicit update of rows on database server.

### Prepared
Determines whether a query is prepared for execution or not.

### Ranged
Indicates whether a range is applied to a dataset.

### UpdateRecordTypes
Used to indicate the update status for the current record when cached updates are enabled.

### UpdatesPending
Used to check the status of the cached updates buffer.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyRange</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>CancelRange</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>EditRangeStart</code></td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><code>GetBlob</code></td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><code>Locate</code></td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><code>LocateEx</code></td>
<td>Overloaded. Excludes features that don't need to be included to the <code>TMemDataSet.Locate</code> method of TDataSet.</td>
</tr>
<tr>
<td><code>Prepare</code></td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td><code>RestoreUpdates</code></td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><code>RevertRecord</code></td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><code>SaveToXML</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><code>SetRange</code></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><code>SetRangeEnd</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><code>SetRangeStart</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>
</tbody>
</table>
### UnPrepare

Frees the resources allocated for a previously prepared query on the server and client sides.

### UpdateResult

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

### UpdateStatus

Indicates the current update status for the dataset when cached updates are enabled.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

### Properties

Properties of the **TMemDataSet** class.

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
</tbody>
</table>
### KeyExclusive
Specifies the upper and lower boundaries for a range.

### LocalConstraints
Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

### LocalUpdate
Used to prevent implicit update of rows on database server.

### Prepared
Determines whether a query is prepared for execution or not.

### Ranged
Indicates whether a range is applied to a dataset.

### UpdateRecordTypes
Used to indicate the update status for the current record when cached updates are enabled.

### UpdatesPending
Used to check the status of the cached updates buffer.

#### See Also
- [TMemDataSet Class](#)
- [TMemDataSet Class Members](#)

---

**4.5.1.1.2.1 CachedUpdates Property**

Used to enable or disable the use of cached updates for a dataset.

**Class**

[TMemDataSet](#)

**Syntax**

```pascal
property CachedUpdates: boolean default False;
```

**Remarks**

Use the CachedUpdates property to enable or disable the use of cached updates for a
dataset. Setting CachedUpdates to True enables updates to a dataset (such as posting changes, inserting new records, or deleting records) to be stored in an internal cache on the client side instead of being written directly to the dataset’s underlying database tables. When changes are completed, an application writes all cached changes to the database in the context of a single transaction.

Cached updates are especially useful for client applications working with remote database servers. Enabling cached updates brings up the following benefits:

- Fewer transactions and shorter transaction times.
- Minimized network traffic.

The potential drawbacks of enabling cached updates are:

- Other applications can access and change the actual data on the server while users are editing local copies of data, resulting in an update conflict when cached updates are applied to the database.
- Other applications cannot access data changes made by an application until its cached updates are applied to the database.

The default value is False.

**Note:** When establishing master/detail relationship the CachedUpdates property of detail dataset works properly only when `TDADatasetOptions.LocalMasterDetail` is set to True.

See Also

- `UpdatesPending`
- `TMemDataSet.ApplyUpdates`
- `RestoreUpdates`
- `CommitUpdates`
- `CancelUpdates`
- `UpdateStatus`
- `TCustomDADataset.Options`

© 1997-2019 Devart. All Rights Reserved.  

4.5.1.1.2.2 IndexFieldNames Property

Used to get or set the list of fields on which the recordset is sorted.

Class

`TMemDataSet`

Syntax
property IndexFieldNames: string;

Remarks
Use the IndexFieldNames property to get or set the list of fields on which the recordset is sorted. Specify the name of each column in IndexFieldNames to use as an index for a table. Ordering of column names is significant. Separate names with semicolon. The specified columns don’t need to be indexed. Set IndexFieldNames to an empty string to reset the recordset to the sort order originally used when the recordset's data was first retrieved. Each field may optionally be followed by the keyword ASC / DESC or CIS / CS / BIN. Use ASC, DESC keywords to specify a sort direction for the field. If one of these keywords is not used, the default sort direction for the field is ascending. Use CIS, CS or BIN keywords to specify a sort type for string fields: CIS - compare without case sensitivity; CS - compare with case sensitivity; BIN - compare by character ordinal values (this comparison is also case sensitive). If a dataset uses a TCustomDAConnection component, the default value of sort type depends on the TCustomDAConnection.Options option of the connection. If a dataset does not use a connection (TVirtualTable dataset), the default is CS. Read IndexFieldNames to determine the field (or fields) on which the recordset is sorted. Ordering is processed locally. **Note:** You cannot process ordering by BLOB fields.

Example
The following procedure illustrates how to set IndexFieldNames in response to a button click:

```pascal
Dataset1.IndexFieldNames := 'LastName ASC CIS; DateDue DESC';
```

4.5.1.1.2.3 KeyExclusive Property

Specifies the upper and lower boundaries for a range.

Class
TMemDataSet

Syntax

```pascal
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`

Class

`TMemDataSet`

Syntax

```plaintext
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.
4.5.1.1.2.5 LocalUpdate Property

Used to prevent implicit update of rows on database server.

Class

TMemDataSet

Syntax

```
property LocalUpdate: boolean default False;
```

Remarks

Set the LocalUpdate property to True to prevent implicit update of rows on database server. Data changes are cached locally in client memory.

© 1997-2019 Devart. All Rights Reserved.

4.5.1.1.2.6 Prepared Property

Determines whether a query is prepared for execution or not.

Class

TMemDataSet

Syntax

```
property Prepared: boolean;
```

Remarks

Determines whether a query is prepared for execution or not.

See Also

- Prepare

© 1997-2019 Devart. All Rights Reserved.

4.5.1.1.2.7 Ranged Property

Indicates whether a range is applied to a dataset.

Class
**TMemDataSet**

Syntax

```
property Ranged: Boolean;
```

Remarks

Use the Ranged property to detect whether a range is applied to a dataset.

See Also

- `SetRange`
- `SetRangeEnd`
- `SetRangeStart`

© 1997-2019 Devart. All Rights Reserved.

**4.5.1.2.8 UpdateRecordTypes Property**

Used to indicate the update status for the current record when cached updates are enabled.

Class

**TMemDataSet**

Syntax

```
property UpdateRecordTypes: TUpdateRecordTypes default [rtModified, rtInserted, rtUnmodified];
```

Remarks

Use the UpdateRecordTypes property to determine the update status for the current record when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateRecordTypes offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of records.

See Also

- `CachedUpdates`

© 1997-2019 Devart. All Rights Reserved.
4.5.1.1.2.9  UpdatesPending Property

Used to check the status of the cached updates buffer.

Class

TMemDataSet

Syntax

property UpdatesPending: boolean;

Remarks

Use the UpdatesPending property to check the status of the cached updates buffer. If UpdatesPending is True, then there are edited, deleted, or inserted records remaining in local cache and not yet applied to the database. If UpdatesPending is False, there are no such records in the cache.

See Also

• CachedUpdates

© 1997-2019 Devart. All Rights Reserved.

4.5.1.1.3  Methods

Methods of the TMemDataSet class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyRange</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>CancelRange</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior</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>DeferredPost</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to 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 state.</td>
</tr>
</tbody>
</table>

© 2019 Devart
Assignments to field values specify the end of the range of rows to include in the dataset.

**SetRangeStart**

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

**UnPrepare**

Frees the resources allocated for a previously prepared query on the server and client sides.

**UpdateResult**

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**UpdateStatus**

Indicates the current update status for the dataset when cached updates are enabled.

See Also

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

4.5.1.3.1 ApplyRange Method

Applies a range to the dataset.

**Class**

[TMemDataSet](#)

**Syntax**

```plaintext
procedure ApplyRange;
```

**Remarks**

Call ApplyRange to cause a range established with **SetRangeStart** and **SetRangeEnd**, or
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
- EditRange
- EditRangeEnd
- EditRangeStart
- IndexFieldNames
- SetRange
- SetRangeEnd
- SetRangeStart

---

#### 4.5.1.1.3.2 ApplyUpdates Method

Writes dataset's pending cached updates to a database.

**Class**

**TMemDataSet**

**Overload List**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ApplyUpdates</code></td>
<td>Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><code>ApplyUpdates(const UpdateRecKinds: TUpdateRecKinds)</code></td>
<td>Writes dataset's pending cached updates of specified records to a database.</td>
</tr>
</tbody>
</table>

© 1997-2019 Devart. All Rights Reserved.
Syntax

```plaintext
procedure ApplyUpdates; overload; virtual;
```

Remarks

Call the ApplyUpdates method to write a dataset's pending cached updates to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error. Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.

**Note:** The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

Example

The following procedure illustrates how to apply a dataset's cached updates to a database in response to a button click:

```plaintext
procedure ApplyButtonClick(Sender: TObject);
begin
  with MyQuery do
  begin
    Session.StartTransaction;
    try
      ... <Modify data>
      ApplyUpdates; <try to write the updates to the database>
      Session.Commit; <on success, commit the changes>
    except
      RestoreUpdates; <restore update result for applied records>
      Session.Rollback; <on failure, undo the changes>
      raise; <raise the exception to prevent a call to CommitUpdates!>
    end;
    CommitUpdates; <on success, clear the cache>
  end;
end;
```

See Also

- `TMemDataSet.CachedUpdates`
- `TMemDataSet.CancelUpdates`
- `TMemDataSet.CommitUpdates`
• **TMemDataSet.UpdateStatus**

Writes dataset's pending cached updates of specified records to a database.

**Class**

TMemDataSet

**Syntax**

```delphi
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.

© 1997-2019
Devart. All Rights Reserved.
TMemDataSet

Syntax

```plaintext
procedure CancelRange;
```

Remarks
Call CancelRange to remove a range currently applied to a dataset. Canceling a range reenables access to all records in the dataset.

See Also
- ApplyRange
- EditRangeEnd
- EditRangeStart
- IndexFieldNames
- SetRange
- SetRangeEnd
- SetRangeStart

© 1997-2019 Devart. All Rights Reserved.

4.5.1.1.3.4 CancelUpdates Method

Clears all pending cached updates from cache and restores dataset in its prior state.

Class
TMemDataSet

Syntax

```plaintext
procedure CancelUpdates;
```

Remarks
Call the CancelUpdates method to clear all pending cached updates from cache and restore dataset in its prior state.
It restores the dataset to the state it was in when the table was opened, cached updates were last enabled, or updates were last successfully applied to the database.
When a dataset is closed, or the CachedUpdates property is set to False, CancelUpdates is called automatically.
4.5.1.3.5 CommitUpdates Method

Clears the cached updates buffer.

Class

**TMemDataSet**

Syntax

```delphi
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 the dataset. If there are, it calls ApplyUpdates. Record modifications made after a call to CommitUpdates repopulate the cached update buffer and require a subsequent call to ApplyUpdates to move them to the database.

See Also

- **CachedUpdates**
- **TMemDataSet.ApplyUpdates**
- **UpdateStatus**
4.5.1.3.6 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.

4.5.1.3.7 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
4.5.1.1.3.8 EditRangeStart Method

Enables changing the starting value for an existing range.

Class

TMemDataSet

Syntax

procedure EditRangeStart;

Remarks

Call EditRangeStart to change the starting value for an existing range.

To specify a start range value, call FieldByName after calling EditRangeStart.

After assigning a new ending value, call ApplyRange to activate the modified range.

See Also

• ApplyRange
• CancelRange
• EditRangeEnd
• IndexFieldNames
• SetRange
• SetRangeEnd
• SetRangeStart

4.5.1.1.3.9 GetBlob Method

Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Class
TMemDataSet

Overload List

<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>

Retrieves TBlob object for a field or current record when the field itself is known.

Class

TMemDataSet

Syntax

```pascal
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.
**Syntax**

```pascal
function GetBlob(const FieldName: string): TBlob; overload;
```

**Parameters**

*FieldName*

Holds the name of an existing field.

**Return Value**

TBlob object that was retrieved.

**Example**

```pascal
VirtualQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
```

**See Also**

- `TBlob`

---

**4.5.1.3.10 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: TLocateOptions)</td>
<td>Searches a dataset by the specified fields for a specific record and positions cursor on it.</td>
</tr>
<tr>
<td>Locate(const KeyFields: string; const KeyValues: variant; Options: TLocateOptions)</td>
<td>Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.</td>
</tr>
</tbody>
</table>
TMemDataSet

Syntax

```pascal
function Locate(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateOptions): boolean;
reintroduce; overload;
```

**Parameters**

- **KeyFields**
  - Holds TField objects in which to search.
- **KeyValues**
  - Holds the variant that specifies the values to match in the key fields.
- **Options**
  - Holds additional search latitude when searching in string fields.

**Return Value**

- True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

**Class**

TMemDataSet

Syntax

```pascal
function Locate(const KeyFields: string; const KeyValues: variant; Options: TLocateOptions): boolean; overload; override;
```

**Parameters**

- **KeyFields**
  - Holds a semicolon-delimited list of field names in which to search.
- **KeyValues**
  - Holds the variant that specifies the values to match in the key fields.
- **Options**
  - Holds additional search latitude when searching in string fields.

**Return Value**

- True if it finds a matching record, and makes this record the current one. Otherwise it returns False.
Remarks

Call the Locate method to search a dataset for a specific record and position cursor on it. KeyFields is a string containing a semicolon-delimited list of field names on which to search. KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. An example is provided below.

Options is a set that optionally specifies additional search latitude when searching in string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

Locate returns True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

The Locate function works faster when dataset is locally sorted on the KeyFields fields. Local dataset sorting can be set with the `TMemDataSet.IndexFieldNames` property.

Example

An example of specifying multiple search values:

```pascal
with CustTable do
  Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P', '408-431-1000']), [loPartialKey]);
```

See Also

- `TMemDataSet.IndexFieldNames`
- `TMemDataSet.LocateEx`

© 1997-2019 Devart. All Rights Reserved.

4.5.1.1.3.11 LocateEx Method

Excludes features that don't need to be included to the `TMemDataSet.Locate` method of TDataSet.

Class

`TMemDataSet`

Overload List
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LocateEx</strong>(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateExOptions)</td>
<td>Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.</td>
</tr>
<tr>
<td><strong>LocateEx</strong>(const KeyFields: string; const KeyValues: variant; Options: TLocateExOptions)</td>
<td>Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified field names.</td>
</tr>
</tbody>
</table>

Class
TMemDataSet

Syntax

```delphi
function LocateEx(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateExOptions): boolean; overload;
```

Parameters

- **KeyFields**
  - Holds TField objects to search in.
- **KeyValues**
  - Holds the values of the fields to search for.
- **Options**
  - Holds additional search parameters which will be used by the LocateEx method.

Return Value

- True, if a matching record was found. Otherwise returns False.
Syntax

```pascal
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 TLocExOption enumeration.

See Also

- `TMemDataSet.IndexFieldNames`
- `TMemDataSet.Locate`

© 1997-2019
Devart. All Rights Reserved.

4.5.1.3.12 Prepare Method

Allocates resources and creates field components for a dataset.

Class

`TMemDataSet`

Syntax
**procedure** Prepare; **virtual**;

**Remarks**

Call the Prepare method to allocate resources and create field components for a dataset. To learn whether dataset is prepared or not use the Prepared property. The UnPrepare method unprepares a query.

**Note:** When you change the text of a query at runtime, the query is automatically closed and unprepared.

**See Also**

- Prepared
- UnPrepare

© 1997-2019 Devart. All Rights Reserved.

---

4.5.1.1.3.13 RestoreUpdates Method

Marks all records in the cache of updates as unapplied.

**Class**

**TMemDataSet**

**Syntax**

**procedure** RestoreUpdates;

**Remarks**

Call the RestoreUpdates method to return the cache of updates to its state before calling ApplyUpdates. RestoreUpdates marks all records in the cache of updates as unapplied. It is useful when ApplyUpdates fails.

**See Also**

- CachedUpdates
- **TMemDataSet.ApplyUpdates**
- CancelUpdates
- UpdateStatus

© 1997-2019 Devart. All Rights Reserved.
4.5.1.1.3.14 RevertRecord Method

Cancels changes made to the current record when cached updates are enabled.

Class

**TMemDataSet**

Syntax

```delphi
procedure RevertRecord;
```

Remarks

Call the RevertRecord method to undo changes made to the current record when cached updates are enabled.

See Also

• *CachedUpdates*
• *CancelUpdates*

© 1997-2019 Devart. All Rights Reserved.

4.5.1.1.3.15 SaveToXML Method

Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Class

**TMemDataSet**

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SaveToXML(Destination: TStream)</strong></td>
<td>Saves the current dataset data to a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SaveToXML(const FileName: string)</strong></td>
<td>Saves the current dataset data to a file in the XML format compatible with ADO format.</td>
</tr>
</tbody>
</table>

© 1997-2019 Devart. All Rights Reserved.
Saves the current dataset data to a stream in the XML format compatible with ADO format.

Class

TMemDataSet

Syntax

procedure SaveToXML(Destination: TStream); overload;

Parameters

Destination
Holds a TStream object.

Remarks

Call the SaveToXML method to save the current dataset data to a file or a stream in the XML format compatible with ADO format.

If the destination file already exists, it is overwritten. It remains open from the first call to SaveToXML until the dataset is closed. This file can be read by other applications while it is opened, but they cannot write to the file.

When saving data to a stream, a TStream object must be created and its position must be set in a preferable value.

See Also

- M:Devart.Dac.TVirtualTable.LoadFromFile(System.String,System.Boolean)
- M:Devart.Dac.TVirtualTable.LoadFromStream(Borland.Vcl.TStream,System.Boolean)

© 1997-2019 Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback

Saves the current dataset data to a file in the XML format compatible with ADO format.

Class

TMemDataSet

Syntax

procedure SaveToXML(const FileName: string); overload;

Parameters

FileName
Holds the name of a destination file.
4.5.1.3.16 SetRange Method

Sets the starting and ending values of a range, and applies it.

Class

TMemDataSet

Syntax

procedure SetRange(const StartValues: array of System.TVarRec; const EndValues: array of System.TVarRec; StartExclusive: Boolean = False; EndExclusive: Boolean = False);

Parameters

StartValues
  Indicates the field values that designate the first record in the range. In C++, StartValues_Size is the index of the last value in the StartValues array.

EndValues
  Indicates the field values that designate the last record in the range. In C++, EndValues_Size is the index of the last value in the EndValues array.

StartExclusive
  Indicates the upper and lower boundaries of the start range.

EndExclusive
  Indicates the upper and lower boundaries of the end range.

Remarks

Call SetRange to specify a range and apply it to the dataset. The new range replaces the currently specified range, if any.

SetRange combines the functionality of SetRangeStart, SetRangeEnd, and ApplyRange in a single procedure call. SetRange performs the following functions:

1. Puts the dataset into dsSetKey state.
2. Erases any previously specified starting range values and ending range values.
3. Sets the start and end range values.
4. Applies the range to the dataset.

After a call to SetRange, the cursor is left on the first record in the range.

If either StartValues or EndValues has fewer elements than the number of fields in the current index, then the remaining entries are ignored when performing a search.
4.5.1.1.3.17 SetRangeEnd Method

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

Class

TMemDataSet

Syntax

procedure SetRangeEnd;

Remarks

Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL.
Subsequent field assignments made with FieldByName specify the actual set of ending values for a range.
After assigning end-range values, call ApplyRange to activate the modified range.

See Also

- ApplyRange
- CancelRange
- EditRangeEnd
- EditRangeStart
- IndexFieldNames
- KeyExclusive
- SetRangeEnd
- SetRangeStart

© 1997-2019 Devart. All Rights Reserved.

Request Support        DAC Forum        Provide Feedback
4.5.1.1.3.18  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

```pascal
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

4.5.1.1.3.19  UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class

TMemDataSet
Syntax

```plaintext
procedure UnPrepare; virtual;
```

Remarks

Call the UnPrepare method to free the resources allocated for a previously prepared query on the server and client sides.

**Note:** When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- Prepare

© 1997-2019 Devart. All Rights Reserved.

4.5.1.1.3.20 UpdateResult Method

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Class

`TMemDataSet`

Syntax

```plaintext
function UpdateResult: TUpdateAction;
```

Return Value

- a value of the TUpdateAction enumeration.

Remarks

Call the UpdateResult method to read the status of the latest call to the ApplyUpdates method while cached updates are enabled. UpdateResult reflects updates made on the records that have been edited, inserted, or deleted.

UpdateResult works on the record by record basis and is applicable to the current record only.

See Also

- CachedUpdates

© 1997-2019 Devart
4.5.1.3.21 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

© 1997-2019 Devart. All Rights Reserved.

4.5.1.4 Events

Events of the TMemDataSet class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
</tbody>
</table>
**OnUpdateRecord**

Occurs when a single update component can not handle the updates.

**See Also**
- **TMemDataSet Class**
- **TMemDataSet Class Members**

4.5.1.1.4.1 **OnUpdateError Event**

Occurs when an exception is generated while cached updates are applied to a database.

**Class**
- **TMemDataSet**

**Syntax**

```
property OnUpdateError: TUpdateErrorEvent;
```

**Remarks**

Write the OnUpdateError event handler to respond to exceptions generated when cached updates are applied to a database.

E is a pointer to an EDatabaseError object from which application can extract an error message and the actual cause of the error condition. The OnUpdateError handler can use this information to determine how to respond to the error condition. UpdateKind describes the type of update that generated the error.

UpdateAction indicates the action to take when the OnUpdateError handler exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateError can handle or correct the error, set UpdateAction to uaRetry before exiting the error handler.

The error handler can use the TField.OldValue and TField.NewValue properties to evaluate error conditions and set TField.NewValue to a new value to reapply. In this case, set UpdateAction to uaRetry before exiting.

**Note:** If a call to ApplyUpdates raises an exception and ApplyUpdates is not called within the context of a try...except block, an error message is displayed. If the OnUpdateError handler cannot correct the error condition and leaves UpdateAction set to uaFail, the error message is displayed twice. To prevent redisplay, set UpdateAction to uaAbort in the error handler.
### 4.5.1.4.2 OnUpdateRecord Event

Occurs when a single update component can not handle the updates.

**Class**

`TMemDataSet`

**Syntax**

```object
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`

---

### 4.5.2 Variables

Variables in the `MemDS` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DoNotRaiseExceptionOnUaFail</code></td>
<td>An exception will be raised if</td>
</tr>
</tbody>
</table>
the value of the UpdateAction parameter is uaFail.

| SendDataSetChangeEventAfterOpen | The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. |

4.5.2.1 DoNotRaiseExceptionOnUaFail Variable

An exception will be raised if the value of the UpdateAction parameter is uaFail.

Unit

MemDS

Syntax

DoNotRaiseExceptionOnUaFail: boolean = False;

Remarks

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

To restore the old behaviour, set DoNotRaiseExceptionOnUaFail to True.

4.5.2.2 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 VirtualDAC, 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.

4.6 VirtualDataSet

4.6.1 Classes
Classes in the VirtualDataSet unit.

<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>

4.6.1.1 TCustomVirtualDataSet Class
A base class for representation of arbitrary data in tabular form.
For a list of all members of this type, see TCustomVirtualDataSet members.

Unit
VirtualDataSet

Syntax

```
TCustomVirtualDataSet = class(TMemDataSet);
```

Inheritance Hierarchy

TMemDataSet
## TCustomVirtualDataSet

© 1997-2019 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>Used to enable or disable the use of cached updates for a dataset.</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>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>UpdateRecordTypes</strong></td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong></td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td><strong>ApplyRange</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>Locate</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>LocateEx</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
</tbody>
</table>
### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
4.6.1.2 TVirtualDataSet Class

Dataset that processes arbitrary non-tabular data.
For a list of all members of this type, see TVirtualDataSet members.

Unit

VirtualDataSet

Syntax

TVirtualDataSet = class(TCustomVirtualDataSet);

Inheritance Hierarchy

TMemDataSet
    TCustomVirtualDataSet
    TVirtualDataSet

4.6.1.2.1 Members

TVirtualDataSet 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>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</td>
</tr>
</tbody>
</table>
opening TMemDataSet.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalUpdate</td>
<td>(inherited from TMemDataSet) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>Prepared</td>
<td>(inherited from TMemDataSet) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>(inherited from TMemDataSet) Indicates whether a range is applied to a dataset.</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>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyRange</td>
<td>(inherited from TMemDataSet) Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>(inherited from TMemDataSet) Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>CancelRange</td>
<td>(inherited from TMemDataSet) Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>(inherited from TMemDataSet) Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>(inherited from TMemDataSet) Clears the cached updates buffer.</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>(inherited from TMemDataSet) Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>(inherited from TMemDataSet) Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>(inherited from TMemDataSet) Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>(inherited from TMemDataSet) Overloaded. Retrieves TBlob object for a field or current record when only its</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Locate (inherited from TMemDataSet)</td>
<td>Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx (inherited from TMemDataSet)</td>
<td>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>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>
<tr>
<td>UnPrepare (inherited from TMemDataSet)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult (inherited from TMemDataSet)</td>
<td>Reads the status of the latest call to the</td>
</tr>
</tbody>
</table>
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>OnDeleteRecord</strong></td>
<td>Occurs when a record is deleted from the virtual dataset.</td>
</tr>
<tr>
<td><strong>OnGetFieldValue</strong></td>
<td>Occurs when a virtual dataset asks for the value of a field.</td>
</tr>
<tr>
<td><strong>OnGetRecordCount</strong></td>
<td>Occurs when a virtual dataset queries the number of records.</td>
</tr>
<tr>
<td><strong>OnInsertRecord</strong></td>
<td>Occurs when a new record is added to the virtual dataset.</td>
</tr>
<tr>
<td><strong>OnModifyRecord</strong></td>
<td>Occurs when a record is modified in the virtual dataset.</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>
## Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

## Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnDeleteRecord</td>
<td>Occurs when a record is deleted from the virtual data set.</td>
</tr>
<tr>
<td>OnGetFieldValue</td>
<td>Occurs when a virtual dataset asks for the value of a field.</td>
</tr>
<tr>
<td>OnGetRecordCount</td>
<td>Occurs when a virtual dataset queries the number of records.</td>
</tr>
<tr>
<td>OnInsertRecord</td>
<td>Occurs when a new record is added to the virtual data set.</td>
</tr>
<tr>
<td>OnModifyRecord</td>
<td>Occurs when a record is modified in the virtual data set.</td>
</tr>
</tbody>
</table>

## See Also
- [TVirtualDataSet Class](#)
- [TVirtualDataSet Class Members](#)

© 1997-2019 Devart. All Rights Reserved.
**TVirtualDataSet**

**Syntax**

```property OnDeleteRecord: TOnDeleteRecordEvent;```

**Remarks**

Write the OnDeleteRecord event handler to process deletion of a record from the virtual dataset. When firing the event, the virtual dataset sends the record number of the record being deleted into the event handler.

**See Also**

- `OnInsertRecord`
- `OnModifyRecord`

© 1997-2019 Devart. All Rights Reserved.

4.6.1.2.2.2 OnGetFieldValue Event

Occurs when a virtual dataset asks for the value of a field.

**Class**

**TVirtualDataSet**

**Syntax**

```property OnGetFieldValue: TOnGetFieldValueEvent;```

**Remarks**

Write the OnGetFieldValue event handler to return field data to the virtual dataset. Since virtual dataset does not store a data, when opening it fires the event to obtain the value of each field.

**See Also**

- `OnGetRecordCount`

© 1997-2019 Devart. All Rights Reserved.
4.6.1.2.2.3 OnGetRecordCount Event

Occurs when a virtual dataset queries the number of records.

Class

TVirtualDataSet

Syntax

```
property OnGetRecordCount: TOnGetRecordCountEvent;
```

Remarks

Write the OnGetRecordCount event handler to let the virtual dataset know, how many records it contains. Since virtual dataset does not store a data, when opening it fires the event to determine how many records will be kept in it. Then the virtual dataset will ask for the value of each field, using a OnGetFieldValue event handler.

See Also

• OnGetFieldValue

© 1997-2019 Devart. All Rights Reserved.

 DAC Forum Request Support Provide Feedback

4.6.1.2.2.4 OnInsertRecord Event

Occurs when a new record is added to the virtual data set.

Class

TVirtualDataSet

Syntax

```
property OnInsertRecord: TOnModifyRecordEvent;
```

Remarks

Write the OnInsertRecord event handler to process insertion of new record in the virtual dataset. When firing the event, the virtual dataset is positioned on the record being inserted.

See Also

• OnModifyRecord
• OnDeleteRecord

© 1997-2019 Devart. All Rights Reserved.

 DAC Forum Request Support Provide Feedback
4.6.1.2.2.5 OnModifyRecord Event

Occurs when a record is modified in the virtual data set.

Class

TVirtualDataSet

Syntax

property OnModifyRecord: TOnModifyRecordEvent;

Remarks

Write the OnModifyRecord event handler to process modification of a record in the virtual dataset. When firing the event, the virtual dataset is positioned on the record being modified.

See Also

- OnInsertRecord
- OnDeleteRecord

Types

Types in the VirtualDataSet unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOnDeleteRecordEvent</td>
<td>This type is used for the TVirtualDataSet.OnDeleteRecord event.</td>
</tr>
<tr>
<td>TOnGetFieldValueEvent</td>
<td>This type is used for the TVirtualDataSet.OnGetFieldValue event.</td>
</tr>
<tr>
<td>TOnGetRecordCountEvent</td>
<td>This type is used for the TVirtualDataSet.OnGetRecordCount event.</td>
</tr>
<tr>
<td>TOnModifyRecordEvent</td>
<td>This type is used for the TVirtualDataSet.OnInsertRecord and</td>
</tr>
</tbody>
</table>
### 4.6.2.1 TOnDeleteRecordEvent Procedure Reference

This type is used for the `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.

### 4.6.2.2 TOnGetFieldValueEvent Procedure Reference

This type is used for the `TVirtualDataSet.OnGetFieldValue` event.

**Unit**

`VirtualDataSet`

**Syntax**

```delphi
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.

4.6.2.3 TOnGetRecordCountEvent Procedure Reference

This type is used for the TVirtualDataSet.OnGetRecordCount event.

Unit

VirtualDataSet

Syntax

TOnGetRecordCountEvent = procedure (Sender: TObject; out Count: Integer) of object;

Parameters

Sender

An object that raised the event.

Count

The number of records that the virtual dataset will contain.

4.6.2.4 TOnModifyRecordEvent Procedure Reference

This type is used for TVirtualDataSet.OnInsertRecord and TVirtualDataSet.OnModifyRecord events.

Unit

VirtualDataSet

Syntax

TOnModifyRecordEvent = procedure (Sender: TObject; var RecNo: Integer) of object;

Parameters

Sender
An object that raised the event.

RecNo
Number of the record being inserted or modified.

### 4.7 VirtualQuery

#### 4.7.1 Classes

Classes in the `VirtualQuery` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TCustomVirtualQuery</code></td>
<td>A base class that implements <code>TVirtualQuery</code> functionality.</td>
</tr>
<tr>
<td><code>TDataSetLink</code></td>
<td>Used to link a <code>TDataSet</code> descendant as a data source for querying data in <code>TVirtualQuery</code>.</td>
</tr>
<tr>
<td><code>TDataSetLinks</code></td>
<td>This type is used for the <code>TCustomVirtualQuery.SourceDataSets</code> property.</td>
</tr>
<tr>
<td><code>TVirtualCollationManager</code></td>
<td>Used to enable user-defined collations.</td>
</tr>
<tr>
<td><code>TVirtualFunctionManager</code></td>
<td>Used to enable user-defined functions.</td>
</tr>
<tr>
<td><code>TVirtualQuery</code></td>
<td>A class used to retrieve data simultaneously from several different RDBMS'es.</td>
</tr>
<tr>
<td><code>TVirtualQueryOptions</code></td>
<td>This class allows setting up the behaviour of the <code>TVirtualQuery</code> class.</td>
</tr>
</tbody>
</table>

#### 4.7.1.1 `TCustomVirtualQuery` Class

A base class that implements `TVirtualQuery` functionality.
For a list of all members of this type, see `TCustomVirtualQuery` members.
Unit
virtualQuery

Syntax

```
TCustomVirtualQuery = class(TCustomDADataSet);
```

Inheritance Hierarchy

```
TMemDataSet
  TCustomDADataSet
  TCustomVirtualQuery
```

© 1997-2019
Devart. All Rights Reserved.

4.7.1.1 Members

TCustomVirtualQuery class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL (inherited from TCustomDADataSet)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions (inherited from TCustomDADataSet)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td>Connection (inherited from TCustomDADataSet)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>DataTypeMap (inherited from TCustomDADataSet)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td>Debug (inherited from TCustomDADataSet)</td>
<td>Used to display executing statement, all its parameters' values, and the type of parameters.</td>
</tr>
<tr>
<td>DetailFields (inherited from TCustomDADataSet)</td>
<td>Used to specify the fields that correspond to the foreign key fields from</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</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>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>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>IsQuery</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MacroCount</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Makes it possible to change</td>
</tr>
<tr>
<td>Documentation Section</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>MasterFields</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>SQL queries easily. Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of TVirtualQuery object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SourceDataSets</strong></td>
<td>Contains a collection of source datasets for querying data.</td>
</tr>
<tr>
<td><strong>SQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to provide a SQL</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</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>
</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>SQLUpdate (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>UniDirectional (inherited from TCustomDADataset)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateRecordTypes (inherited from TMemDataSet)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending (inherited from TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from TCustomDADataset)</td>
<td>Adds condition to the WHERE clause of SELECT</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ApplyRange</strong></td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong></td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong></td>
<td>Used to learn whether TCustomDADataset has already fetched all rows.</td>
</tr>
<tr>
<td>Function</td>
<td>Inherited From</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Fetching</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FindKey</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FindMacro</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FindNearest</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FindParam</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>GetBlob</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>GetDataType</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>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>GotoCurrent</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Locate</strong></td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong></td>
<td>Overloaded. Excludes features that don't need to be included to the <code>TMemDataSet.Locate</code> method of TDataSet.</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 name passed in Name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Sets or uses 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 BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong></td>
<td>Indicates that subsequent assignments to field values</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SetRangeStart (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SQLSaved (inherited from TCustomDADataSet)</td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td>UnLock (inherited from TCustomDADataSet)</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td>UnPrepare (inherited from TMemDataSet)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult (inherited from TMemDataSet)</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus (inherited from TMemDataSet)</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 (inherited from TCustomDADataSet)</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch (inherited from TCustomDADataSet)</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute (inherited from TCustomDADataSet)</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch (inherited from TCustomDADataSet)</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute (inherited from TCustomDADataSet)</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
</tbody>
</table>
### Properties

Properties of the **TCustomVirtualQuery** class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to return SQL text 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>TCustomDADataset</strong>)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to display executing statement, all its parameters'</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</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>IndexFieldNames</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 whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>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>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MacroCount</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</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>Used to specify the behaviour of TVirtualQuery 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>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>SourceDataSets</strong></td>
<td>Contains a collection of source datasets for querying data.</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td><strong>SQLInsert</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the <strong>TCustomDADataset.RefreshRecord</strong> procedure.</td>
</tr>
<tr>
<td><strong>SQLUpdate</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td><strong>UniDirectional</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td><strong>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**
4.7.1.1.2.1  Options Property

Used to specify the behaviour of TVirtualQuery object.

Class

TCustomVirtualQuery

Syntax

property Options: TVirtualQueryOptions;

Remarks

Set the properties of Options to specify the behaviour of a TVirtualQuery object.

See Also

• TVirtualQuery

4.7.1.1.2.2  SourceDataSets Property

Contains a collection of source datasets for querying data.

Class

TCustomVirtualQuery

Syntax

property SourceDataSets: TDataSetLinks;

Remarks

Use the property to create a list of the data sources to which the SQL statement will be executed. Each data source has to be a TDataSet descendant, connected to a database and opened prior to SQL statement execution in the TVirtualQuery (if TVirtualQueryOptions.AutoOpenSources option is set to False). Each data source gets its
own "schema name" and "table name" which are used to identify the data source in the SQL statement. Each data source must have a unique combination of schema name and table name.

See Also
- TVirtualQueryOptions.AutoOpenSources

4.7.1.1.3 Events

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute (inherited from TCustomDADataset)</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch (inherited from TCustomDADataset)</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute (inherited from TCustomDADataset)</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch (inherited from TCustomDADataset)</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute (inherited from TCustomDADataset)</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td>OnRegisterCollations</td>
<td>Occurs when the connection is opened to register the user-defined collation used in the query text.</td>
</tr>
<tr>
<td>OnRegisterFunctions</td>
<td>Occurs when the connection is opened to register the user-defined function used in the query text.</td>
</tr>
<tr>
<td>OnUpdateError (inherited from TMemDataSet)</td>
<td>Occurs when an exception is generated while cached updates are applied to a</td>
</tr>
</tbody>
</table>
OnUpdateRecord (inherited from TMemDataSet) Occurs when a single update component can not handle the updates.

See Also
- TCustomVirtualQuery Class
- TCustomVirtualQuery Class Members

4.7.1.3.1 OnRegisterCollations Event

Occurs when the connection is opened to register the user-defined collation used in the query text.

Class
TCustomVirtualQuery

Syntax

property OnRegisterCollations: TRegisterCollationsEvent;

Remarks

Occurs after a component has executed a query to a database.

See Also
- TCustomDADataSet.Execute

4.7.1.3.2 OnRegisterFunctions Event

Occurs when the connection is opened to register the user-defined function used in the query text.

Class
TCustomVirtualQuery
property OnRegisterFunctions: TRegisterFunctionsEvent;

Remarks
Occurs after a component has executed a query to a database.

See Also
- TCustomDADataset.Execute

4.7.1.2 TDataSetLink Class
Used to link a TDataSet descendant as a data source for querying data in TVirtualQuery.
For a list of all members of this type, see TDataSetLink members.

Unit
virtualQuery

Syntax
TDataSetLink = class(TCollectionItem);

Remarks
Add a TDataSetLink instance to the TCustomVirtualQuery.SourceDataSets collection using the TDataSetLinks.Add method to link a TDataSet descendant as a data source for querying data in TVirtualQuery.

See Also
- TVirtualQuery
- TCustomVirtualQuery.SourceDataSets
- TDataSetLinks
- TDataSetLinks.Add
**TDataSetLink** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataSet</strong></td>
<td>Defines a TDataSet descendant to be linked as a data source for querying data in TVirtualQuery.</td>
</tr>
<tr>
<td><strong>SchemaName</strong></td>
<td>Defines the schema name which will be used to identify the linked source dataset in a SQL statement.</td>
</tr>
<tr>
<td><strong>TableName</strong></td>
<td>Defines the table name which will be used to identify the linked source dataset in a SQL statement.</td>
</tr>
</tbody>
</table>

Properties of the **TDataSetLink** class.

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

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataSet</strong></td>
<td>Defines a TDataSet descendant to be linked as a data source for querying data in TVirtualQuery.</td>
</tr>
<tr>
<td><strong>SchemaName</strong></td>
<td>Defines the schema name which will be used to identify the linked source dataset in a SQL statement.</td>
</tr>
<tr>
<td><strong>TableName</strong></td>
<td>Defines the table name which will be used to identify the linked source dataset in a SQL statement.</td>
</tr>
</tbody>
</table>
4.7.1.2.2.1  DataSet Property

Defines a TDataSet descendant to be linked as a data source for querying data in TVirtualQuery.

Class
TDataSetLink

Syntax

```delphi
property DataSet: TDataSet;
```

See Also
- TVirtualQuery
- SchemaName
- TableName

Remarks
Can be left empty. In this case either no schema name or the "main" schema name can be...

4.7.1.2.2.2  SchemaName Property

Defines the schema name which will be used to identify the linked source dataset in a SQL statement.

Class
TDataSetLink

Syntax

```delphi
property SchemaName: string;
```

Remarks
Can be left empty. In this case either no schema name or the "main" schema name can be
used when referring to the linked source dataset in a SQL statement.
Combination of schema name and table name must be unique for each linked dataset.

See Also
- **DataSet**
- **SchemaName**

4.7.1.2.3 **TableName Property**

Defines the table name which will be used to identify the linked source dataset in a SQL statement.

**Class**

**TDataSetLink**

**Syntax**

```property TableName: string stored** GetTableNameStored;```

**Remarks**

Must be filled.
Combination of schema name and table name must be unique for each linked dataset.

See Also
- **DataSet**
- **SchemaName**

4.7.1.3 **TDataSetLinks Class**

This type is used for the **TCustomVirtualQuery.SourceDataSets** property.
For a list of all members of this type, see **TDataSetLinks** members.

**Unit**

**virtualQuery**

**Syntax**
TDataSetLinks = class(TCollection);

Remarks

TDataSetLinks is the TCollection descendant which contains a collection of the TDataSetLink instances, each of which links a TDataSet descendant as a data source for querying data in TVirtualQuery.

See Also

• TVirtualQuery
• TDataSetLink

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Overloaded. Adds a new TDataSetLink instance to the collection.</td>
</tr>
</tbody>
</table>

See Also

TDataSetLinks class overview.

Methods of the TDataSetLinks class.
For a complete list of the TDataSetLinks class members, see the TDataSetLinks Members topic.
4.7.1.3.2.1 Add Method

Adds a new TDataSetLink instance to the collection.

**Class**

TDataSetLinks

**Overload List**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds a new TDataSetLink instance to the collection.</td>
</tr>
<tr>
<td>Add(DataSet: TDataSet; const SchemaName: string; const TableName: string)</td>
<td>Adds a new TDataSetLink instance to the collection and fills its properties.</td>
</tr>
</tbody>
</table>

**Syntax**

```delphi
function Add: TDataSetLink; overload;
```

**Return Value**

A instance which has been added.

**Remarks**

Fill the TDataSetLink.Dataset property of the returned TDataSetLink instance to link a TDataSet descendant as a data source for querying data in TVirtualQuery. Fill TDataSetLink.SchemaName and TDataSetLink.TableName properties to identify the source dataset in a SQL statement. Combination of schema name and table name must be unique.
for each linked dataset. Also, a source dataset can be linked using the \texttt{Add} method.

See Also
- \texttt{TVirtualQuery}
- \texttt{TDataSetLink}
- \texttt{TDataSetLink.DataSet}
- \texttt{TDataSetLink.SchemaName}
- \texttt{TDataSetLink.TableName}
- \texttt{Add}

Adds a new \texttt{TDataSetLink} instance to the collection and fills its properties.

Class
\texttt{TDataSetLinks}

Syntax

\begin{verbatim}
function Add(DataSet: TDataSet; \texttt{const} SchemaName: \texttt{string}; \texttt{const} TableName: \texttt{string}): \texttt{TDataSetLink}; overload;
\end{verbatim}

Parameters

- \textit{DataSet}
  - Defines a TDataSet descendant to be linked as a data source for querying data in \texttt{TVirtualQuery}.

- \textit{SchemaName}
  - Defines the schema name which will be used to identify the linked source dataset in a SQL statement. Can be left empty. In this case either no schema name or the "main" schema name can be used when referring to the dataset in a SQL statement.

- \textit{TableName}
  - Defines the table name which will be used to identify the linked source dataset in a SQL statement. Must be filled.

Return Value

- A instance which has been added.

Remarks

Combination of schema name and table name must be unique for each linked dataset. Also, a source dataset can be linked using the \texttt{TDataSetLinks.Add} method.
See Also

- TVirtualQuery
- TDataSetLink
- TDataSetLink.DataSet
- TDataSetLink.SchemaName
- TDataSetLink.TableName
- TDataSetLinks.Add

Syntax

```delphi
TVirtualCollationManager = class(System.TObject);
```

4.7.4.1 Members

**TVirtualCollationManager** class overview.

4.7.1.5 **TVirtualFunctionManager Class**

Used to enable user-defined functions.
For a list of all members of this type, see **TVirtualFunctionManager** members.
4.7.1.6 TVirtualQuery Class

A class used to retrieve data simultaneously from several different RDBMS'es. For a list of all members of this type, see TVirtualQuery members.

Unit

virtualQuery

Syntax

TVirtualQuery = class(TCustomVirtualQuery);

Remarks

TVirtualQuery component is used to retrieve data simultaneously from several different RDBMS'es. Instead of a database connection, it use a collection of TDataSet descendants defined in the TCustomVirtualQuery.SourceDataSets property as the data source, for which a SQL statement can be build. The SQLite is used as an internal SQL-engine, so the SQLite syntax has to be used for SQL statements.

Use TVirtualQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. Set SQL, SQLInsert, SQLDelete, SQLRefresh, and SQLUpdate properties to define SQL statements for subsequent accesses to the database server. There is no restriction to their syntax, so any SQL statement is allowed. Usually you need to use INSERT, DELETE, and UPDATE statements but you also may use stored procedures in more diverse cases.

To modify records, you can specify KeyFields. If they are not specified, TVirtualQuery will retrieve primary keys for UpdatingTable from metadata. TVirtualQuery can automatically update only one table. Updating table is defined by the UpdatingTable property if this property is set. Otherwise, the table a field of which is the first field in the field list in the SELECT clause is used as an updating table.
The SQLInsert, SQLDelete, SQLUpdate, SQLRefresh properties support automatic binding of parameters which have identical names to fields captions. To retrieve the value of a field as it was before the operation use the field name with the 'OLD_' prefix. This is especially useful when doing field comparisons in the WHERE clause of the statement. Use the TCustomDADataset.BeforeUpdateExecute event to assign the value to additional parameters and the TCustomDADataset.AfterUpdateExecute event to read them.

Inheritance Hierarchy

TMemDataSet
  TCustomDADataset
    TCustomVirtualQuery
      TVirtualQuery

© 1997-2019 Devart. All Rights Reserved.

4.7.1.6.1 Members

TVirtualQuery class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL (inherited from TCustomDADataset)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions (inherited from TCustomDADataset)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td>Connection (inherited from TCustomDADataset)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>DataTypeMap (inherited from TCustomDADataset)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td>Debug (inherited from TCustomDADataset)</td>
<td>Used to display executing statement, all its parameters' values, and the type of parameters.</td>
</tr>
<tr>
<td>DetailFields (inherited from TCustomDADataset)</td>
<td>Used to specify the fields that correspond to the</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>foreign key fields from MasterFields when building master/detail relationship.</td>
<td></td>
</tr>
<tr>
<td>Disconnected</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<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>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>IndexFieldNames</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 whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>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>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-------------------</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>TCustomVirtualQuery</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>ParamCount</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Params</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Prepared</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Ranged</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SourceDataSets</td>
<td>(inherited from TCustomVirtualQuery) Contains a collection of source datasets for querying data.</td>
</tr>
<tr>
<td>SQL</td>
<td>(inherited from TCustomDADataset) Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>(inherited from TCustomDADataset) Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>(inherited from TCustomDADataset) Used to specify the SQL statement that will be used when applying an insertion to a dataset.</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>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>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>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>UpdatingTable</td>
<td>Used to specify which table in a query is assumed to be the target for subsequent operations.</td>
</tr>
</tbody>
</table>
Data-modification queries as a result of user incentive to insert, update or delete records.

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td><strong>ApplyRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong> (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>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>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------</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 learn whether TCustomDADataSet has already fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong></td>
<td>Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong></td>
<td>Used to learn whether TCustomDADataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td><strong>FindKey</strong></td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>Description is not available at the moment.</td>
</tr>
<tr>
<td><strong>FindNearest</strong></td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>GetDataType</strong></td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldObject</strong></td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong></td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><strong>GetFieldScale</strong></td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>Method Name (inherited from)</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>GetKeyFieldNames (TCustomDADataset)</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy (TCustomDADataset)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GotoCurrent (TCustomDADataset)</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td>Locate (TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx (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 (TCustomDADataset)</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td>MacroByName (TCustomDADataset)</td>
<td>Finds a Macro with the name passed in Name.</td>
</tr>
<tr>
<td>ParamByName (TCustomDADataset)</td>
<td>Sets or uses parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td>Prepare (TCustomDADataset)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>RefreshRecord (TCustomDADataset)</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>RestoreSQL (TCustomDADataset)</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td>RestoreUpdates (TMemDataSet)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord (TMemDataSet)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveSQL (TCustomDADataset)</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td>SaveToXML (TMemDataSet)</td>
<td>Overloaded. Saves the current dataset data to a file</td>
</tr>
</tbody>
</table>
SetOrderBy (inherited from TCustomDADataset) | Builds an ORDER BY clause of a SELECT statement.
---|---
SetRange (inherited from TMemDataSet) | Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from TMemDataSet) | Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from TMemDataSet) | Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from TCustomDADataset) | Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from TCustomDADataset) | Releases a record lock.
UnPrepare (inherited from TMemDataSet) | Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from TMemDataSet) | Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from TMemDataSet) | Indicates the current update status for the dataset when cached updates are enabled.

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute (inherited from TCustomDADataset)</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch (inherited from TCustomDADataset)</td>
<td>Occurs after dataset finishes</td>
</tr>
<tr>
<td>Event</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</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>OnRegisterCollations</td>
<td>Occurs when the connection is opened to register the user-defined collation used in the query text.</td>
</tr>
<tr>
<td>OnRegisterFunctions</td>
<td>Occurs when the connection is opened to register the user-defined function used in the query text.</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

Properties of the `TVirtualQuery` class.

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

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL (inherited from <code>TCustomDADataSet</code>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates (inherited from <code>TMemDataSet</code>)</td>
<td>Used to enable or disable the use of cached updates</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Conditions</td>
<td>(inherited from TCustomDACDataSet) Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td>Connection</td>
<td>(inherited from TCustomDACDataSet) Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>DataTypeMap</td>
<td>(inherited from TCustomDACDataSet) Used to set data type mapping rules</td>
</tr>
<tr>
<td>Debug</td>
<td>(inherited from TCustomDACDataSet) Used to display executing statement, all its parameters' values, and the type of parameters.</td>
</tr>
<tr>
<td>DetailFields</td>
<td>(inherited from TCustomDACDataSet) Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Connected</td>
<td>(inherited from TCustomDACDataSet) Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>FetchRows</td>
<td>(inherited from TCustomDACDataSet) Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>(inherited from TCustomDACDataSet) Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>(inherited from TCustomDACDataSet) Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</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 TCustomDACDataSet) Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from TMemDataSet) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>(inherited from TCustomDACDataSet) Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate operations.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>properties if they were empty before updating 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>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 TVirtualQuery object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>TMemDataSet</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>SourceDataSets</strong></td>
<td>TCustomVirtualQuery</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>SQLUpdate</strong></td>
<td>TCustomDADataSet</td>
</tr>
</tbody>
</table>
| **UniDirectional**| TCustomDADataSet                    | Used if an application does }
not need bidirectional access to records in the result set.

**UpdateRecordTypes** (inherited from **TMemDataSet**)
Used to indicate the update status for the current record when cached updates are enabled.

**UpdatesPending** (inherited from **TMemDataSet**)
Used to check the status of the cached updates buffer.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>UpdatingTable</strong></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
- **TVirtualQuery Class**
- **TVirtualQuery Class Members**

© 1997-2019 Devart. All Rights Reserved.
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.

4.7.1.6.2.2 UpdatingTable Property

Used to specify which table in a query is assumed to be the target for subsequent data-
modification queries as a result of user incentive to insert, update or delete records.

Class
`TVirtualQuery`

Syntax

```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 `TCustomVirtualQuery.Options`) if appropriate SQL (SQLInsert, SQLUpdate or SQLDelete) is not provided.
If UpdatingTable is not set then the first table used in a query is assumed to be the target.

Example
Below are two examples for the query, where:
1. the only allowed value for UpdatingTable property is 'Dept';
2. allowed values for UpdatingTable are 'Dept' and 'Emp'.

In the first case (or by default) editable field is ShipName, in the second
4.7.1.7 TVirtualQueryOptions Class

This class allows setting up the behaviour of the TVirtualQuery class.
For a list of all members of this type, see TVirtualQueryOptions members.

Unit
virtualQuery

Syntax

TVirtualQueryOptions = class(TDADatasetOptions);

Inheritance Hierarchy

TDADatasetOptions
  TVirtualQueryOptions

See Also

• TVirtualQuery

© 1997-2019 Devart. All Rights Reserved.

4.7.1.7.1 Members

TVirtualQueryOptions class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOpenSources</td>
<td>Used to automatically open data sources when SQL statement executed</td>
</tr>
<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>DefaultValues</td>
<td>(inherited from TDADatasetOptions) Used to request default values/expressions from the server and assign them to</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DetailDelay</td>
<td>(inherited from TDADDataSetOptions) 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 TDADDataSetOptions) 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>(inherited from TDADDataSetOptions) Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td>FullRefresh</td>
<td></td>
</tr>
<tr>
<td>InsertAllSetFields</td>
<td>(inherited from TDADDataSetOptions) Used to include all set dataset fields in the generated INSERT statement</td>
</tr>
<tr>
<td>LocalMasterDetail</td>
<td>(inherited from TDADDataSetOptions) 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>(inherited from TDADDataSetOptions) Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td>MasterFieldsNullable</td>
<td>(inherited from TDADDataSetOptions) 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>(inherited from TDADDataSetOptions) Used to set the MaxValue and MinValue properties of TIntegerField and</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TFloatField</td>
<td>Used for TFloatField to appropriate values.</td>
</tr>
<tr>
<td>QueryRecCount</td>
<td>Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td>RemoveOnRefresh</td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td>RequiredFields</td>
<td>Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td>ReturnParams</td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td>SetEmptyStrToNull</td>
<td>Force replace of empty strings with NULL values in data. Default value is False.</td>
</tr>
<tr>
<td>SetFieldsReadOnly</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>StrictUpdate</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>TrimFixedChar</td>
<td>Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
</tbody>
</table>
4.7.1.7.2 Properties

Properties of the TVirtualQueryOptions class.

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

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrimVarChar</td>
<td>Used to specify whether to discard all trailing spaces in the variable-length 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>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</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 additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td>RemoveOnRefresh</td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td>RequiredFields</td>
<td>Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td>ReturnParams</td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td>SetFieldsReadOnly</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>StrictUpdate</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>TrimFixedChar</td>
<td>Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
<tr>
<td>UpdateAllFields</td>
<td>Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td>UpdateBatchSize</td>
<td>Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>AutoOpenSources</strong></td>
<td>Used to automatically open data sources when SQL statement executed</td>
</tr>
<tr>
<td><strong>FullRefresh</strong></td>
<td>Used to specify the fields to include in the automatically generated SQL statement when calling the method.</td>
</tr>
<tr>
<td><strong>SetEmptyStrToNull</strong></td>
<td>Force replace of empty strings with NULL values in data. Default value is False.</td>
</tr>
<tr>
<td><strong>TrimVarChar</strong></td>
<td>Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.</td>
</tr>
</tbody>
</table>

**See Also**
- [TVirtualQueryOptions Class](#)
- [TVirtualQueryOptions Class Members](#)

© 1997-2019 Devart. All Rights Reserved.

4.7.1.7.2.1 AutoOpenSources Property

Used to automatically open data sources when SQL statement executed

**Class**

[TVirtualQueryOptions](#)

**Syntax**

```property
property AutoOpenSources: boolean default False;
```

**Remarks**

Use the property to automatically open data sources specified in the `TCustomVirtualQuery.SourceDataSets` list when SQL statement executed. If `AutoOpenSources` is False, each data source has to be opened prior to SQL statement execution in the `TVirtualQuery`. If `AutoOpenSources` is True, data sources will be opened automatically. The default value is False;

**See Also**
4.7.1.7.2.2 FullRefresh Property

Used to specify the fields to include in the automatically generated SQL statement when calling the method.

Class

TVirtualQueryOptions

Syntax

```pascal
property FullRefresh: boolean;
```

Remarks

Use the FullRefresh property to specify what fields to include in the automatically generated SQL statement when calling the TCustomDADataset.RefreshRecord method. If the FullRefresh property is True, all fields from a query are included into SQL statement to refresh a single record. If FullRefresh is False, only fields from TVirtualQuery.UpdatingTable are included.

**Note:** If FullRefresh is True, the refresh of SQL statement for complex queries and views may be generated with errors. The default value is False.

© 1997-2019
Devart. All Rights Reserved.

4.7.1.7.2.3 SetEmptyStrToNull Property

Force replace of empty strings with NULL values in data. Default value is False.

Class

TVirtualQueryOptions

Syntax

```pascal
property SetEmptyStrToNull: boolean;
```

© 1997-2019
Devart. All Rights Reserved.

Request Support   DAC Forum   Provide Feedback
4.7.1.7.2.4 TrimVarChar Property

Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

Class

TVirtualQueryOptions

Syntax

```property TrimVarChar: boolean;```

Remarks

Use the TrimVarChar property to specify whether to discard all trailing spaces in the variable-length string fields of a dataset. The default value is False.

4.7.2 Types

Types in the VirtualQuery unit.

## Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRegisterFunctionsEvent</td>
<td>This type is used for the TCustomVirtualQuery.RegisterFunctions events.</td>
</tr>
</tbody>
</table>

4.7.2.1 TRegisterFunctionsEvent Procedure Reference

This type is used for the TCustomVirtualQuery.RegisterFunctions events.

Unit

VirtualQuery

Syntax

```TRegisterFunctionsEvent = procedure (Sender: TObject; const FunctionManager: TVirtualFunctionManager) of object;```
Parameters

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

**FunctionManager**
Used to enable user-defined functions.

Remarks
The result is True if SQL statement is executed successfully. False otherwise.

4.8 **VirtualTable**

4.8.1 **Classes**

Classes in the `VirtualTable` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TVirtualAutoIncField</code></td>
<td>A field class for generating auto-increment field values.</td>
</tr>
<tr>
<td><code>TVirtualTable</code></td>
<td>A base class for storing data in memory.</td>
</tr>
</tbody>
</table>

4.8.1.1 **TVirtualAutoIncField Class**

A field class for generating auto-increment field values.
For a list of all members of this type, see `TVirtualAutoIncField` members.

Unit

`VirtualTable`

Syntax

```plaintext
TVirtualAutoIncField = class(TAutoIncField);
```

Remarks
TVirtualAutoIncField is a TAutoIncField descendant. It is created in cases when TFieldDef of the ftAutoInc type is created either at design time using the TVirtualTable.FieldDefs property, or at run time using the TVirtualTable.AddField method.

An initial field value is specified by the TVirtualAutoIncField.InitialValue property, and its increment is specified by the TVirtualAutoIncField.Increment property. Auto-generation can be enabled or disabled using the TVirtualAutoIncField.AutoGenerateValue property.

See Also
• TVirtualAutoIncField.InitialValue
• TVirtualAutoIncField.Increment
• TVirtualAutoIncField.AutoGenerateValue
### AutoGenerateValue Property

**Indicates whether the field values can be automatically generated.**

<table>
<thead>
<tr>
<th><strong>AutoGenerateValue</strong></th>
<th>Indicates whether the field values can be automatically generated.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Increment</strong></td>
<td>Specifies the increment of the auto-incrementing field.</td>
</tr>
<tr>
<td><strong>InitialValue</strong></td>
<td>Specifies the initial value of the auto-incrementing field.</td>
</tr>
</tbody>
</table>

#### Remarks

Use the AutoGenerateValue property to indicate whether the field values can be automatically generated.

When the property value is `arAutoInc`, a new incrementing value for the field will be generated automatically when inserting a new record. Otherwise (arNone and arDefault values), the field will be empty. The default value is `arAutoInc`.

**Note:** An attempt to change the AutoGenerateValue property when the field dataset is open generates an exception. Thus, AutoGenerateValue only works with persistent field components, which remain when the dataset is closed.

#### See Also
- **InitialValue**
- **Increment**
4.8.1.1.2.2 Increment Property

Specifies the increment of the auto-incrementing field.

Class

TVirtualAutoIncField

Syntax

```pascal
property Increment: integer default 1;
```

Remarks

Use the Increment property to specify the increment of the auto-incrementing field. The increment can be negative. The default value is 1.

See Also

- InitialValue
- AutoGenerateValue

4.8.1.1.2.3 InitialValue Property

Specifies the initial value of the auto-incrementing field.

Class

TVirtualAutoIncField

Syntax

```pascal
property InitialValue: integer default -1;
```

Remarks

Use the InitialValue property to specify the initial value of the auto-incrementing field. The initial value can be negative. The default value is 1.

See Also

- Increment

© 2019 Devart

Devart. All Rights Reserved.
4.8.1.2 TVirtualTable Class

A base class for storing data in memory.
For a list of all members of this type, see TVirtualTable members.

Unit
virtualTable

Syntax

TVirtualTable = class(TMemDataSet);

Remarks

TVirtualTable is inherited from the TMemDataSet component. TVirtualTable stores data in memory and does not have linked data files. To add fields to virtual table at design time use Fields Editor. Call the TVirtualTable.AddField method to add fields at run time.
Immediately after creating, virtual table will be empty. Then you define new fields or load existing table files so that the virtual table object becomes initialized and ready to be opened. When you close virtual table it will discard its record set. To keep the data you entered at design-time for later use you may wish to include the voStored option in the TVirtualTable.Options property. At run time you will need to call the TVirtualTable.SaveToFile method explicitly to store modifications to the file that may be retrieved back into the virtual table by calling the TVirtualTable.LoadFromFile method later.

Note: TVirtualTable component is added to the Data Access page of the component palette, not to the VirtualDAC page.
TVirtualTable supports auto-incrementing fields using a special TVirtualAutoIncField field class.

Inheritance Hierarchy

TMemDataSet
   TVirtualTable
4.8.1.2.1 Members

**TVirtualTable** class overview.

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **CachedUpdates**     | (inherited from **TMemDataSet**)
|                       | Used to enable or disable the use of cached updates for a dataset.          |
| **DefaultSortType**   | Used to determine the default type of local sorting for string fields.     |
| **IndexFieldNames**   | (inherited from **TMemDataSet**)
|                       | Used to get or set the list of fields on which the recordset is sorted.     |
| **KeyExclusive**      | (inherited from **TMemDataSet**)
|                       | Specifies the upper and lower boundaries for a range.                      |
| **LocalConstraints**  | (inherited from **TMemDataSet**)
|                       | Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet. |
| **LocalUpdate**       | (inherited from **TMemDataSet**)
|                       | Used to prevent implicit update of rows on database server.                |
| **Options**           | Used to specify actions to take on the fields data at the time of opening or closing TVirtualTable dataset. |
| **Prepared**          | (inherited from **TMemDataSet**)
|                       | Determines whether a query is prepared for execution or not.               |
| **Ranged**            | (inherited from **TMemDataSet**)
|                       | Indicates whether a range is applied to a dataset.                         |
| **UpdateRecordTypes** | (inherited from **TMemDataSet**)
|                       | Used to indicate the update status for the current record when cached updates are enabled. |
| **UpdatesPending**    | (inherited from **TMemDataSet**)
<p>|                       | Used to check the status of the cached updates buffer.                     |</p>
<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddField</strong></td>
<td>Adds a new TFieldDef object with the name determined by Name.</td>
</tr>
<tr>
<td><strong>ApplyRange</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>Assign</strong></td>
<td>Copies fields and data from another TDataSet component.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>Clear</strong></td>
<td>Removes all records from TVirtualTable.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteField</strong></td>
<td>Deletes a field specified by name.</td>
</tr>
<tr>
<td><strong>DeleteFields</strong></td>
<td>Deletes all fields.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>LoadFromFile</strong></td>
<td>Loads data from file into a TVirtualTable component.</td>
</tr>
<tr>
<td><strong>LoadFromStream</strong></td>
<td>Copies data of a stream into a TVirtualTable component.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Locate (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</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>SaveToFile</td>
<td>Saves data of a TVirtualTable component to a file.</td>
</tr>
<tr>
<td>SaveToStream</td>
<td>Copies data from a TVirtualTable component to a stream.</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>
<tr>
<td>UnPrepare (inherited from TMemDataSet)</td>
<td>Frees the resources allocated for a previously prepared dataset.</td>
</tr>
</tbody>
</table>
prepared query on the server and client sides.

**UpdateResult** (inherited from **TMemDataSet**)

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**UpdateStatus** (inherited from **TMemDataSet**)

Indicates the current update status for the dataset when cached updates are enabled.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>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

**Properties of the TVirtualTable class.**

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

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>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</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>(inherited from TMemDataSet) Used to avoid setting the Required property of</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>database server.</td>
</tr>
<tr>
<td>Prepared</td>
<td>(inherited from TMemDataSet) Determines whether a query is prepared for</td>
</tr>
<tr>
<td></td>
<td>execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>(inherited from TMemDataSet) Indicates whether a range is applied to a</td>
</tr>
<tr>
<td></td>
<td>dataset.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>(inherited from TMemDataSet) Used to indicate the update status for the</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>buffer.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify actions to take on the fields data at the time of opening or</td>
</tr>
<tr>
<td></td>
<td>closing TVirtualTable dataset.</td>
</tr>
</tbody>
</table>

See Also

- TVirtualTable Class
- TVirtualTable Class Members
4.8.1.2.2.1 DefaultSortType Property

Used to determine the default type of local sorting for string fields.

Class

TVirtualTable

Syntax

```property DefaultSortType: TSortType default stCaseSensitive;```

Remarks

The DefaultSortType property is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.

© 1997-2019 Devart. All Rights Reserved.

4.8.1.2.2 Options Property

Used to specify actions to take on the fields data at the time of opening or closing TVirtualTable dataset.

Class

TVirtualTable

Syntax

```property Options: TVirtualTableOptions default [voPersistentData, voStored, voSkipUnSupportedFieldTypes];```

Remarks

The Options property specifies what actions to take on the fields data at the time of opening or closing TVirtualTable dataset.

© 1997-2019 Devart. All Rights Reserved.

4.8.1.2.3 Methods

Methods of the TVirtualTable class.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddField</td>
<td>Adds a new TFieldDef object with the name determined by Name.</td>
</tr>
<tr>
<td>ApplyRange</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>Assign</td>
<td>Copies fields and data from another TDataSet component.</td>
</tr>
<tr>
<td>CancelRange</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>Clear</td>
<td>Removes all records from TVirtualTable.</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>DeleteField</td>
<td>Deletes a field specified by name.</td>
</tr>
<tr>
<td>DeleteFields</td>
<td>Deletes all fields.</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 file into a TVirtualTable component.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Copies data of a stream into a TVirtualTable component.</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>SaveToFile</td>
<td>Saves data of a TVirtualTable component to a file.</td>
</tr>
<tr>
<td>SaveToStream</td>
<td>Copies data from a TVirtualTable component to a stream.</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 prepared query on the server and client sides.

UpdateResult (inherited from TMemDataSet)
Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

UpdateStatus (inherited from TMemDataSet)
Indicates the current update status for the dataset when cached updates are enabled.

See Also
- TVirtualTable Class
- TVirtualTable Class Members

4.8.1.2.3.1 AddField Method

Adds a new TFieldDef object with the name determined by Name.

Class
TVirtualTable

Syntax

```pascal
procedure AddField(const Name: string; FieldType: TFieldType;
Size: integer = 0; Required: boolean = False);
```

Parameters

Name
Holds the name of the TFieldDef object to add.

FieldType
Holds the type of the TFieldDef object to add.

Size
Holds the size of the string (if the type of TFieldDef object was specified as ftString or ftWideString).

Required
Holds an indicator that determines whether filling the Size parameter is required.
Remarks
Call the AddField method to add a new TFieldDef object with the name determined by Name. FieldType can be ftString, ftWideString, ftSmallint, ftInteger, ftAutoInc, ftWord, ftBoolean, ftLargeint, ftFloat, ftCurrency, ftDate, ftTime, ftDateTime, ftBlob, or ftMemo. When you add ftString or ftWideString field you should specify Size of the string. When the ftAutoInc field type is set, a field of the special TVirtualAutoIncField type will be created in the virtual table, which can generate auto-increment field values.

Example

```pascal
VirtualTable1.AddField('CODE', ftInteger, 0);
VirtualTable1.AddField('NAME', ftString, 30);
```

See Also
• DeleteField
• DeleteFields

© 1997-2019 Devart. All Rights Reserved.

4.8.1.2.3.2 Assign Method

Copies fields and data from another TDataSet component.

Class
TVirtualTable

Syntax

```pascal
procedure Assign(Source: TPersistent); override;
```

Parameters

`Source`
Holds the TDataSet component to copy fields and data from.

Remarks
Call the Assign method to copy fields and data from another TDataSet component. **Note:** Unsupported field types are skipped (i.e. destination dataset will contain less fields than the source one). This may happen when Source is not a TVirtualTable component but some SQL server oriented dataset.
Example

```delphi
VirtualQuery1.SQL.Text := 'SELECT * FROM DEPT';
VirtualQuery1.Active := True;
VirtualTable1.Assign(VirtualQuery1);
VirtualTable1.Active := True;
```

See Also
- TVirtualTable

### 4.8.1.2.3.3 Clear Method

Removes all records from TVirtualTable.

**Class**

TVirtualTable

**Syntax**

```delphi
procedure Clear;
```

**Remarks**

Call the Clear method to remove all records from TVirtualTable.

### 4.8.1.2.3.4 DeleteField Method

Deletes a field specified by name.

**Class**

TVirtualTable

**Syntax**

```delphi
procedure DeleteField(const Name: string);
```

**Parameters**

- `Name`
  
  Holds the name of the field to delete.
Remarks
Call the DeleteField method to delete a field specified by Name.

See Also
- AddField
- DeleteFields

Class
TVirtualTable

Syntax

```pascal
procedure DeleteFields;
```

Remarks
Call the DeleteFields method to delete all fields.

See Also
- DeleteField

Class
TVirtualTable

Syntax

```pascal
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

Remarks

Call the LoadFromFile method to load data from 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. File format will be detected automatically.

4.8.1.2.3.7 LoadFromStream Method

Copies data of a stream into a TVirtualTable component.

Class

TVirtualTable

Syntax

procedure LoadFromStream(Stream: TStream; LoadFields: boolean = True; DecodeHTMLEntities: boolean = True);

Parameters

Stream
   Holds the stream from which the field's value is copied.

LoadFields
   Indicates whether to load fields from the stream.

DecodeHTMLEntities

Remarks

Call the LoadFromStream method to copy data of 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. Data format will be detected automatically.
4.8.1.2.3.8 SaveToFile Method

Saves data of a TVirtualTable component to a file.

Class

TVirtualTable

Syntax

procedure SaveToFile(const FileName: string; StoreFields: boolean = True; StoreAllData: boolean = false);

Parameters

FileName
Holds the name of the file to save data to.

StoreFields
Indicates whether to save fields to a file.

StoreAllData

Remarks

Call the SaveToFile method to save data of a TVirtualTable component to a file. Specify the name of the file as the value of the FileName parameter.

4.8.1.2.3.9 SaveToStream Method

Copies data from a TVirtualTable component to a stream.

Class

TVirtualTable

Syntax

procedure SaveToStream(Stream: TStream; StoreFields: boolean = True; StoreAllData: boolean = false);

Parameters

Stream
Holds the name of the stream to which the field's value is saved.
**StoreFields**
Indicates whether to save the fields names to a file.

**StoreAllData**

Remarks
Call the SaveToStream method to copy data from a TVirtualTable component to a stream. Specify the name of the stream to which the field's value is saved as the value of the Stream parameter.

4.8.2 **Types**
Types in the VirtualTable unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVirtualTableOptions</td>
<td>Represents the set of TVirtualTableOption.</td>
</tr>
</tbody>
</table>

4.8.3 **Enumerations**
Enumerations in the VirtualTable unit.
### TVirtualTableOption Enumeration

Specifies the actions to take on fields data at the time of opening or closing TVirtualTable dataset.

#### Unit

**VirtualTable**

#### Syntax

```plaintext
TVirtualTableOption = (voPersistentData, voStored);
```

#### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>voPersistentData</td>
<td>Dataset will not dispose of its data at the time of dataset closing.</td>
</tr>
<tr>
<td>voStored</td>
<td>Dataset will keep its data set at design-time in DFM file along with other form's stored properties.</td>
</tr>
</tbody>
</table>

© 1997-2019 Devart. All Rights Reserved.  
[Request Support]  [DAC Forum]  [Provide Feedback]