# Table of Contents

**Overview** ...............................................................................................................................................3  
**What's New** ...............................................................................................................................................4  
**Compatibility** ...............................................................................................................................................6  
**OCL headers** ...............................................................................................................................................7  
**Compilation** ...............................................................................................................................................8  
**Demo projects** ...............................................................................................................................................9  
**Using OCL** .............................................................................................................................................10  
  * Getting started .........................................................................................................................................11  
  * Working with connection ...........................................................................................................................13  
  * Working with command .............................................................................................................................15  
  * Working with parameters ..........................................................................................................................16  
  * Working with recordset ............................................................................................................................18  
  * Working with fileds ...................................................................................................................................20  
  * Working with query ....................................................................................................................................21  
  * Working with stored procedures ..............................................................................................................23  
  * Working with SQL scripts .........................................................................................................................24  
  * Working with OraLoader ..........................................................................................................................25  
  * Working with external procedures ..........................................................................................................27  
  * Monitoring SQL statements .....................................................................................................................29  
  * Data types .................................................................................................................................................30  
  * Exceptions .................................................................................................................................................31  

**OCL Classes** ........................................................................................................................................32  
  * CRDataset ................................................................................................................................................33  
    - close ....................................................................................................................................................34  
    - CRDataset ..........................................................................................................................................34  
    - field ....................................................................................................................................................35  
    - fieldCount ........................................................................................................................................35  
    - fields ...............................................................................................................................................36  
    - findField ..........................................................................................................................................36  
    - first ..................................................................................................................................................37  
    - freeBookmark ...................................................................................................................................37  
    - getBookmark ....................................................................................................................................38  
    - initFields ..........................................................................................................................................39
isActive ...................................................................................................................... 39
isBOF ......................................................................................................................... 39
isEOF ......................................................................................................................... 40
last .............................................................................................................................. 41
move .......................................................................................................................... 41
moveTo ...................................................................................................................... 42
moveToBookmark ................................................................................................. 42
next ............................................................................................................................ 43
open ......................................................................................................................... 43
operator [] ............................................................................................................. 43
prev .......................................................................................................................... 44
recordCount ......................................................................................................... 44
recordNo ............................................................................................................... 45
refresh ..................................................................................................................... 46

CRField .................................................................................................................. 61

CRDate .................................................................................................................... 46

CRDate .................................................................................................................... 47
dateFormat ........................................................................................................... 47
day ......................................................................................................................... 48
dayOfWeek ............................................................................................................ 48
format ..................................................................................................................... 49
hour ......................................................................................................................... 50
minute ..................................................................................................................... 50
month ...................................................................................................................... 51
now .......................................................................................................................... 51
operator == ............................................................................................................ 52
second ..................................................................................................................... 52
setDateFormat ...................................................................................................... 52
setDay .................................................................................................................... 53
setHour .................................................................................................................. 53
setMinute ............................................................................................................. 54
setMonth .............................................................................................................. 54
setSecond ............................................................................................................. 54
setYear .................................................................................................................. 55
year ......................................................................................................................... 55

CReXception ......................................................................................................... 57

CReXception ......................................................................................................... 57
message ................................................................................................................. 58
setMessage ............................................................................................................ 58

CRField .................................................................................................................. 60

assign ....................................................................................................................... 60
CRField .................................................................................................................. 61
dataType ............................................................................................................... 61
fieldNo ................................................................................................................... 62
getWChars .......................................................................................................... 62
gerChars ............................................................................................................... 63
gerDate .................................................................................................................. 64
gerDouble .............................................................................................................. 64
gerFloat .................................................................................................................. 65
gerInt ....................................................................................................................... 65
gerLob ..................................................................................................................... 66
gerWString ........................................................................................................... 67
getString .............................................................................................................. 67
isNull ...................................................................................................................... 68
name ......................................................................................................................... 68
precision ......................................................................................................................... 69
required ......................................................................................................................... 69
scale ................................................................................................................................ 69
setWChars ....................................................................................................................... 70
setChars ......................................................................................................................... 70
setDate ................................................................................................................................ 71
setDouble ....................................................................................................................... 71
setFloat ......................................................................................................................... 71
setInt .................................................................................................................................. 72
setLob .............................................................................................................................. 72
setNull ............................................................................................................................ 73
setWString ..................................................................................................................... 73
setString ......................................................................................................................... 73
size .................................................................................................................................. 74

CRFields .......................................................................................................................... 75
count ............................................................................................................................... 75
findItem ........................................................................................................................... 76
item ................................................................................................................................... 76
operator [] ....................................................................................................................... 77

CRLob ............................................................................................................................... 79
assign ............................................................................................................................. 79
clear .................................................................................................................................. 80
compress ......................................................................................................................... 80
getWChars ...................................................................................................................... 81
getChars .......................................................................................................................... 81
getWString ..................................................................................................................... 82
getString ........................................................................................................................ 82
loadFromFile .................................................................................................................... 83
read .................................................................................................................................. 83
saveToFile ....................................................................................................................... 84
setWChars ....................................................................................................................... 84
setChars .......................................................................................................................... 85
setWString ..................................................................................................................... 85
setString ......................................................................................................................... 86
size .................................................................................................................................. 86
write .................................................................................................................................. 87

CRMemDataset ................................................................................................................. 88
CRMemDataset ................................................................................................................. 88

CRSharedObject .............................................................................................................. 89
addRef ............................................................................................................................. 89
checkValid ....................................................................................................................... 90
CRSharedObject ............................................................................................................. 90
refCount .......................................................................................................................... 90
release ............................................................................................................................ 91

CRWString ....................................................................................................................... 92

CRString ........................................................................................................................ 93

CRValue .......................................................................................................................... 94

CRValue .......................................................................................................................... 95
getDate ............................................................................................................................ 95
getDouble ......................................................................................................................... 96
getFloat .................................................................................................................. 96
getInt .......................................................................................................................... 97
getString .................................................................................................................... 97
isNull .......................................................................................................................... 97
operator = .................................................................................................................. 98
operator == ................................................................................................................. 98
setNull ....................................................................................................................... 99
type ............................................................................................................................ 99

OraCommand ............................................................................................................. 108

bindParams ................................................................................................................ 110
cancel ......................................................................................................................... 110
commandText ............................................................................................................ 110
commandType ........................................................................................................... 111
connection ................................................................................................................ 111
createProcCall ........................................................................................................ 111
cursor ........................................................................................................................ 112
cursorState ................................................................................................................ 112
errorOffset ............................................................................................................... 113
execute ....................................................................................................................... 113
executeQuery ............................................................................................................ 115
findParam ................................................................................................................. 115
isPLSQL ..................................................................................................................... 116
isPrepared ............................................................................................................... 117
isQuery ..................................................................................................................... 117
isRowsReturn ........................................................................................................... 117
operator [] ................................................................................................................ 118
OraCommand ............................................................................................................ 119
param ......................................................................................................................... 119
paramCount ............................................................................................................. 120
params ....................................................................................................................... 120
prepare ...................................................................................................................... 121
rowId ......................................................................................................................... 121
rowsProcessed ....................................................................................................... 122
setArrayLength ....................................................................................................... 122
setCommandText .................................................................................................... 123
setCommandType .................................................................................................... 123
setConnection ........................................................................................................ 124
setCursor ................................................................................................................ 125
setSQL ...................................................................................................................... 125

Copyright © 2009 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>126</td>
</tr>
<tr>
<td>SQLType</td>
<td>126</td>
</tr>
<tr>
<td>unprepare</td>
<td>127</td>
</tr>
<tr>
<td><strong>OraConnection</strong></td>
<td>128</td>
</tr>
<tr>
<td>breakExec</td>
<td>130</td>
</tr>
<tr>
<td>changePassword</td>
<td>130</td>
</tr>
<tr>
<td>close</td>
<td>131</td>
</tr>
<tr>
<td>commit</td>
<td>131</td>
</tr>
<tr>
<td>connectMode</td>
<td>131</td>
</tr>
<tr>
<td>createCommand</td>
<td>132</td>
</tr>
<tr>
<td>externalName</td>
<td>132</td>
</tr>
<tr>
<td>getUsername</td>
<td>132</td>
</tr>
<tr>
<td>setAutoCommit</td>
<td>137</td>
</tr>
<tr>
<td>setConnectMode</td>
<td>137</td>
</tr>
<tr>
<td>setConnectString</td>
<td>137</td>
</tr>
<tr>
<td>setDefaultHome</td>
<td>144</td>
</tr>
<tr>
<td>setExternalName</td>
<td>144</td>
</tr>
<tr>
<td>setHome</td>
<td>144</td>
</tr>
<tr>
<td>setInternalName</td>
<td>145</td>
</tr>
<tr>
<td>setIsolationLevel</td>
<td>146</td>
</tr>
<tr>
<td>setLDA</td>
<td>146</td>
</tr>
<tr>
<td>setOCICallStyle</td>
<td>147</td>
</tr>
<tr>
<td>setOCISvcCtx</td>
<td>147</td>
</tr>
<tr>
<td>setPassword</td>
<td>148</td>
</tr>
<tr>
<td>setUnicode</td>
<td>148</td>
</tr>
<tr>
<td>setCharset</td>
<td>149</td>
</tr>
<tr>
<td>setServer</td>
<td>149</td>
</tr>
<tr>
<td>setTransaction</td>
<td>150</td>
</tr>
<tr>
<td>setUsername</td>
<td>151</td>
</tr>
<tr>
<td>startTransaction</td>
<td>151</td>
</tr>
<tr>
<td>transaction</td>
<td>152</td>
</tr>
<tr>
<td>username</td>
<td>152</td>
</tr>
<tr>
<td><strong>OraCursor</strong></td>
<td>154</td>
</tr>
</tbody>
</table>
allocCursor ................................................................. 154
canFetch .......................................................................... 155
freeCursor .......................................................................... 155
getCDA ............................................................................. 155
getOCIStmt ....................................................................... 156
state ................................................................................. 156
OraEnvironment .................................................................. 158
defaultOracleHome.......................................................... 159
freeOCI .............................................................................. 159
getOCIEnv .......................................................................... 160
getOCIError ........................................................................ 160
getOCIVersion .................................................................... 160
initOCI .............................................................................. 161
isEvents ........................................................................... 161
isOCIInitied ...................................................................... 162
isOCILoaded ....................................................................... 162
isThreaded ........................................................................ 162
OCICallStyle ...................................................................... 163
OCILibrary .......................................................................... 163
oracleHomeCount ............................................................. 164
oracleHomeName .............................................................. 164
oracleHomePath ............................................................... 164
resetOracleHomes .............................................................. 165
useDefaultOracleHome ........................................................ 165
setEvents ........................................................................ 166
setOracleHome ................................................................. 166
setInitMode ....................................................................... 167
setOCICallStyle ............................................................... 167
setOCILibrary .................................................................... 168
setThreaded ...................................................................... 168
OraException ...................................................................... 169
errorCode ......................................................................... 169
OraException ....................................................................... 170
OraField ............................................................................ 171
getBFile .............................................................................. 172
getBlob ............................................................................. 172
getchars .......................................................................... 173
getWChars ........................................................................ 173
getClob ............................................................................ 174
getlob .............................................................................. 175
getlong ............................................................................. 175
getLongRaw ....................................................................... 176
getWString ....................................................................... 176
getString .......................................................................... 177
isComplex ......................................................................... 177
OraField ........................................................................... 177
setBFile ............................................................................. 178
setBlob ............................................................................ 178
setClob ............................................................................ 179
setlob .............................................................................. 179
setLong ............................................................................. 180
setLongRaw ....................................................................... 180
size ................................................................................... 180

Copyright © 2009 Devart
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OraFile</td>
<td>182</td>
</tr>
<tr>
<td>allocLob</td>
<td>182</td>
</tr>
<tr>
<td>close</td>
<td>183</td>
</tr>
<tr>
<td>exist</td>
<td>183</td>
</tr>
<tr>
<td>fileDir</td>
<td>184</td>
</tr>
<tr>
<td>fileName</td>
<td>184</td>
</tr>
<tr>
<td>freeLob</td>
<td>184</td>
</tr>
<tr>
<td>isOpen</td>
<td>185</td>
</tr>
<tr>
<td>open</td>
<td>185</td>
</tr>
<tr>
<td>refresh</td>
<td>186</td>
</tr>
<tr>
<td>setFileDir</td>
<td>186</td>
</tr>
<tr>
<td>setFileName</td>
<td>186</td>
</tr>
<tr>
<td>OraInterval</td>
<td>188</td>
</tr>
<tr>
<td>compare</td>
<td>189</td>
</tr>
<tr>
<td>day</td>
<td>189</td>
</tr>
<tr>
<td>fracSecond</td>
<td>190</td>
</tr>
<tr>
<td>toString</td>
<td>190</td>
</tr>
<tr>
<td>hour</td>
<td>191</td>
</tr>
<tr>
<td>intervalType</td>
<td>191</td>
</tr>
<tr>
<td>isNull</td>
<td>192</td>
</tr>
<tr>
<td>minute</td>
<td>192</td>
</tr>
<tr>
<td>month</td>
<td>193</td>
</tr>
<tr>
<td>operator -</td>
<td>193</td>
</tr>
<tr>
<td>operator !=</td>
<td>194</td>
</tr>
<tr>
<td>operator +</td>
<td>194</td>
</tr>
<tr>
<td>operator +=</td>
<td>195</td>
</tr>
<tr>
<td>operator &lt;</td>
<td>195</td>
</tr>
<tr>
<td>operator &lt;=</td>
<td>196</td>
</tr>
<tr>
<td>operator -</td>
<td>196</td>
</tr>
<tr>
<td>operator ==</td>
<td>197</td>
</tr>
<tr>
<td>operator &gt;=</td>
<td>197</td>
</tr>
<tr>
<td>operator &gt;</td>
<td>198</td>
</tr>
<tr>
<td>operator &gt;=</td>
<td>198</td>
</tr>
<tr>
<td>OraInterval</td>
<td>199</td>
</tr>
<tr>
<td>second</td>
<td>200</td>
</tr>
<tr>
<td>setDaySecond</td>
<td>200</td>
</tr>
<tr>
<td>setIntervalType</td>
<td>201</td>
</tr>
<tr>
<td>setNull</td>
<td>201</td>
</tr>
<tr>
<td>fromString</td>
<td>202</td>
</tr>
<tr>
<td>setYearMonth</td>
<td>202</td>
</tr>
<tr>
<td>year</td>
<td>203</td>
</tr>
<tr>
<td>OraLoader</td>
<td>204</td>
</tr>
<tr>
<td>addColumn</td>
<td>205</td>
</tr>
<tr>
<td>beginLoad</td>
<td>206</td>
</tr>
<tr>
<td>breakOnError</td>
<td>206</td>
</tr>
<tr>
<td>clearColumns</td>
<td>207</td>
</tr>
<tr>
<td>columnCount</td>
<td>207</td>
</tr>
<tr>
<td>columnError</td>
<td>207</td>
</tr>
<tr>
<td>connection</td>
<td>208</td>
</tr>
<tr>
<td>createColumns</td>
<td>208</td>
</tr>
<tr>
<td>finishLoad</td>
<td>209</td>
</tr>
<tr>
<td>getColumn</td>
<td>209</td>
</tr>
<tr>
<td>OraLoader</td>
<td>209</td>
</tr>
<tr>
<td>putColumnData</td>
<td>210</td>
</tr>
</tbody>
</table>
OraParam

OraMonitor

OraLob

Oracle Class Library Reference

putColumnNull ................................................................. 211
putNextRow ................................................................. 211
rowError ................................................................. 212
setBeakOnError ............................................................. 212
setConnection .......................................................... 212
setTableName .............................................................. 213
tableName ................................................................. 213

OraLob ........................................................................... 215
allocLob ....................................................................... 215
disableBuffering .......................................................... 216
enableBuffering .......................................................... 216
freeLob ....................................................................... 217
getOCILObLocator ....................................................... 217
getOCISvcCtx ............................................................. 217
init ............................................................................... 218
isUnicode ..................................................................... 218
isInited ....................................................................... 218
length ......................................................................... 219
readLob ....................................................................... 219
setUnicode .................................................................... 220
setOCISvcCtx .............................................................. 220
writeLob ...................................................................... 220

OraMonitor .................................................................. 222
isActive ....................................................................... 222
start ............................................................................. 223
stop ............................................................................... 223

OraParam ..................................................................... 224
assign .......................................................................... 226
dataType ....................................................................... 226
direction ....................................................................... 227
getBFile ......................................................................... 227
getBlob ......................................................................... 228
getWChars ...................................................................... 228
getChars ......................................................................... 229
getClob ......................................................................... 230
getCursor ....................................................................... 230
getDate .......................................................................... 231
getDouble ....................................................................... 231
getFloat ......................................................................... 232
getInt ............................................................................. 232
getLob ............................................................................ 233
getLong .......................................................................... 233
getLongRaw ..................................................................... 234
getWString ...................................................................... 234
getString ......................................................................... 235
getValue ......................................................................... 235
isNull ............................................................................. 236
isTable .......................................................................... 236
name ............................................................................. 237
operator = ....................................................................... 237
OraParam ....................................................................... 238
setBFile ......................................................................... 238
setBlob ......................................................................... 238

Copyright © 2009 Devart
setWChars ................................................................. 239
setChars ................................................................. 239
setClob ................................................................. 240
setCursor ............................................................... 240
setDataType ........................................................... 241
setDate ................................................................. 241
setDirection .......................................................... 242
setDouble .............................................................. 243
setFloat ................................................................. 243
setInt ................................................................. 244
setLob ................................................................. 244
setLong ............................................................... 245
setLongRaw ........................................................... 245
setNull ................................................................. 246
setSize ................................................................. 246
setWString ............................................................. 247
setString ............................................................... 247
setTable ............................................................... 248
setTableLength ....................................................... 248
setValue ............................................................... 249
size ................................................................. 249
tableLength .......................................................... 250

OraParams ............................................................ 251
count ................................................................. 251
findItem ............................................................... 252
item ................................................................. 252
operator [] ........................................................... 253

OraQuery ............................................................... 255
close ................................................................. 256
execute ............................................................... 256
isActive .............................................................. 257
open ................................................................. 257
operator [] ........................................................... 258
OraQuery ............................................................... 258
prepare ............................................................... 259
unprepare ........................................................... 260

OraRecordset .......................................................... 261
fetchAll .............................................................. 262
fetchAll .............................................................. 262
field ................................................................. 263
findField ............................................................. 263
isActive .............................................................. 264
isCached ............................................................. 265
operator [] ........................................................... 265
setCached ............................................................ 266
setFetchRows ............................................................... 266

OraScript ............................................................... 268
breakOnError .......................................................... 269
connection ............................................................ 269
errorOffset ............................................................ 269
execute ............................................................... 270
OraScript ............................................................... 270
Top Level Intro

This page is printed before a new top-level chapter starts
Oracle Class Library

Overview
What's New in OCL 3
Compatibility
OCL headers
Compilation
Demo projects
Using OCL
OCL Classes
Index
1 Overview

Oracle Class Library (OCL) provides native connectivity to the Oracle database server. OCL uses Oracle Call Interface (OCI) directly. OCI is low-level native application programming interface to access to Oracle. Using OCI in Oracle Class Library allows to create lightweight and fast applications working with Oracle. Oracle Class Library encapsulates OCI calls in high-level classes that allows to hide the complexity of using OCI directly and keep performance and all abilities of native routines. With OCL you can use the power and flexibility of SQL in your application programs without any restrictions.

OCL contains classes to control connection, execute SQL statements, store and process result rows of queries and some common classes useful for developing database applications. All classes have intuitive, easy to use interface.

OCL is written with ANSI C++ and uses Standard C++ Library only that allows you to port your application easily to another platform.

Oracle Class Library provides easiness in using from Pro*C/C++ and power of Oracle Call Interface.

OCL allows you to

- design and develop highly customized database applications
- improve performance of data processing in your applications
- compile the same application for different platforms
- automatically convert between Oracle internal data types and high-level language data types
- fetch rows quickly
- cache result rows in memory to process them later
- use arrays as input and output program variables
- call stored procedures easily
- handle errors
- use LONG fields in the database

To sum it up, OCL is a full-featured tool that supports a professional approach to develop database applications.

OCL | Index
2 What's New

OCL 3.30 new features:
- Oracle 11g support added.
- Microsoft Visual Studio 2008 support added.

OCL 3.20 new features:
- Oracle 10g for x64 platform support added.
- Unicode character data in OraField, OraParam, OraLob supported.
- Specifying character set in OraField, OraParam supported.

OCL 3.00 new features:
- Oracle 10g support added.
- Oracle 10g Instant Client support added.
- Support BINARY_DOUBLE, BINARY_FLOAT types.
- Add new getChars function to OraField and CRField classes.
- Support reading records from several REF CURSOR parameters.
- Fix memory leak after fetching data from cursor.

OCL 2.55 new features:
- New DB Monitor interface support.
- OCLMonitor class renamed to OraMonitor class.

OCL 2.50 new features:
- MS Visual Studio.NET C++ support, Borland C++ Builder 6 support, Kylix 3 for C++ support.
- OraLoader class for direct path loading Oracle API.
- OraTransaction class for Oracle distributed transaction support.
- OraScript class was added for scripts executing similar to SQL*Plus scripts.
- OraEnvironment class was added for advanced OCI initialization
control and other Oracle specific features.

- OraConnection::setHome function was added for assigning Oracle home to connection.
- OraConnection::changePassword function was added.
- OraConnection::setConnectMode function was added.
- OraConnection::getTNSAliases function was added for retrieving TNS aliases existing in Oracle client.
- OraCommand::errorOffset function was added for retrieving error symbol or statement position.
- Support Binary and Boolean datatypes was added.
- OraInterval and OraTimeStamp classes was added for representing new Oracle 9i data types.
- DB Monitor support.

OCL 2 introduces the following new features and enhancements.

- New model of working with result records
  
  added new class OraRecordset that represents set of records returned by SELECT statement or REF CURSOR parameter. OraRecordset object is returned by OraCommand member functions executeQuery or execute. Also it is possible to use single object of OraQuery class to execute statements that return rows.

- Support Oracle 8 Call Interface
- Support type of Oracle 8 such as CLOB, BLOB and BFILE data types
- Support Oracle 9i
- Simplified work with tables and stored procedures
- Using namespace
- Ability to write external procedures
- Support transaction isolation
- Monitoring of execution SQL statements

OCL | Index
3 Compatibility

As OCL uses only standard libraries and STL it can be compiled with any C++ compiler. OCL is distributed with libraries for Microsoft Visual Studio 2008, Microsoft Visual C++ 8.0, 7.1, 7.0 and 6.0, Borland C++ Builder 6 and 5, Kylix 3 for C++ and gcc 4.0 compiler for Linux.
OCL supports Oracle 11g, 10g, 9i, 8i, 8.0 and 7.3, including Personal and Lite.
# 4 OCL headers

All OCL header files are placed in include folder.

<table>
<thead>
<tr>
<th>Header</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocl.h</td>
<td>main OCL header</td>
</tr>
<tr>
<td>crdef.h</td>
<td>defines general use classes i.e. CRString, CRDate etc.</td>
</tr>
<tr>
<td>crdb.h</td>
<td>defines base classes i.e. CRDataset, CRField, CRLob.</td>
</tr>
<tr>
<td>oracall.h</td>
<td>defines OCI related functions and constants.</td>
</tr>
<tr>
<td>ora.h</td>
<td>defines classes working with Oracle server i.e OraConnection,</td>
</tr>
<tr>
<td></td>
<td>OraCommand, OraRecordset, OraParam, OraCursor.</td>
</tr>
<tr>
<td>oraloader.h</td>
<td>defines classes that use Direct Load API for loading data into database.</td>
</tr>
<tr>
<td>orascript.h</td>
<td>defines classes that execute scripts like SQL*Plus scripts.</td>
</tr>
<tr>
<td>oraextproc.h</td>
<td>defines functions that are necessary to work with extended procedures.</td>
</tr>
<tr>
<td>oramonitor.h</td>
<td>classes that monitor OCL in OraMonitor application.</td>
</tr>
</tbody>
</table>

[OCL](#) | [Index](#)
5 Compilation

To compile and link OCL library follow these command line instructions:

- for Visual C++ assuming that you are at the root of OCL folder:

  cd ocl\src\msvc6
  _Make.bat

- for Borland C++ Builder assuming that you are at the root of OCL folder:

  cd ocl\src\bcb6
  _Make.bat

- for GNU compiler:

  cd .\ocl\src\gnu
  make

Following macros are used to define options for linking OCL with OCI library.

- **OCL DYNAMIC** OCL automatically determine what OCI functionality will be used and OCL will be dynamically linked with OCI library. If you are not define any macros this macros will be defined automatically.

- **OCL OCI7** OCL will use a set of OCI 7 functions and statically linked with OCI library

- **OCL OCI8** OCL will use a set of OCI 8 functions and statically linked with OCI library

- **OCL OCI81, OCL OCI90** Defines OCI library version.

- **OCL MFC** Is used to compile OCL with MFC support

- **OCL VCL** Is used to compile OCL with VCL support
6 Demo projects

Following demo projects accompany OCL library and provide a good bootstrap to start working with it. They are located inside demos folder of OCL library.

<table>
<thead>
<tr>
<th>Demo</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>cgi</td>
<td>CGI program for the Apache EB Server. Demonstrates how to use OCL with EB applications.</td>
</tr>
<tr>
<td>clob</td>
<td>Demonstrates how to work with CLOB fields as well as storing them to a file and retrieving them back into a table.</td>
</tr>
<tr>
<td>crsql</td>
<td>Lightweight version of Oracle sqlplus developed with OCL.</td>
</tr>
<tr>
<td>cursor</td>
<td>Demonstrates how to work with PL/SQL cursors.</td>
</tr>
<tr>
<td>demo</td>
<td>Provides a template of step-by-step procedures necessary to build basic OCL applications.</td>
</tr>
<tr>
<td>dmlarray</td>
<td>Shows usage of OCI DML Arrays with OCL. This feature allows to process a number of identical SQL statements all in a single call to an execute function. This may lead to a significant reduction in the number of round-trips from the client to the server.</td>
</tr>
<tr>
<td>extproc</td>
<td>Demonstrates how to create external procedures for the Oracle. These procedures are then called on the server side from within SQL statements and may improve performance considerably in comparison to the PL/SQL stored procedures.</td>
</tr>
<tr>
<td>loader</td>
<td>Shows usage of OraLoader class that use Direct Path Oracle API for loading large data amount into database.</td>
</tr>
<tr>
<td>long</td>
<td>Working with LONG and LONG RAW fields.</td>
</tr>
<tr>
<td>longchar</td>
<td>Demonstrates the principles of working with long string fields similar to the way as with the fields of type Long.</td>
</tr>
<tr>
<td>mfc</td>
<td>Full-fledged windowed MFC application with OCL.</td>
</tr>
<tr>
<td>plsqltable</td>
<td>Demonstrates work with PL/SQL tables or otherwise known as PL/SQL arrays.</td>
</tr>
<tr>
<td>proc</td>
<td>Shows how to work with stored procedures returning REF CURSOR and how to obtain records from the cursor.</td>
</tr>
<tr>
<td>query</td>
<td>Demonstrates principles of how to work with OraQuery class.</td>
</tr>
<tr>
<td>recordset</td>
<td>Demonstrates basic concepts of working with OraCommand and OraRecordset classes. Provides various examples of using objects and methods from these classes.</td>
</tr>
<tr>
<td>strproc</td>
<td>Demonstrates how OCL works with various types of stored procedures.</td>
</tr>
<tr>
<td>threads</td>
<td>Shows how OCL library works in multi-threaded environment.</td>
</tr>
<tr>
<td>vcl</td>
<td>C++Builder demo project. Introduces basic concepts of OCL inside C++Builder environment.</td>
</tr>
</tbody>
</table>
7 Using OCL

Getting started
  _orking with connection
  _orking with command
  _orking with parameters
  _orking with recordset
  _orking with fields
  _orking with query
  _orking with stored procedures
  _orking with SQL scripts
  _orking with OraLoader
  _orking with external procedures
Monitoring SQL statements
Data types
Exceptions

OCL | Index
7.1 Getting started

When you start the project with Oracle Class Library you should include `ocl.h` header from `ocl/include` directory to your project and link it with `ocl.lib` from `ocl/lib` directory.

```c
#include "ocl.h"
```

Before using OCL classes you can include whole `ocl` namespace

```c
using namespace ocl;
```

The first object which you must obligatory create is `OraConnection` object that opens and controls connection to Oracle server. Use its member functions `setUsername`, `setPassword`, `setServer` to set login information. And establish connection with `connect` function.

```c
OraConnection connection;
connection.setUsername("scott");
connection.setPassword("tiger");
connection.setServer("ora");
connection.open();
```

You can execute any SQL or PL/SQL statement using `OraCommand` object. Before you need link it to `OraConnection` object by `setConnection` member function. Set text of SQL by `setCommandText` member function and execute it by `execute` method.

```c
OraCommand cmd;

cmd.setConnection(connection);
cmd.setCommandText("INSERT INTO Emp (EmpNo) VALUES (8315)" );
cmd.execute();
```

Use `param` function to access to input and output parameters and member functions of `OraParam` to write and read their values.

```c
cmd.setCommandText("INSERT INTO Dept (DeptNo, DName) VALUES (:DeptNo, :DName)");
cmd.param("DeptNo").setInt(70);
cmd.param("DName").setString("Research");
```

To execute SQL that returns rows use `OraQuery` object. Call `setCommandText` function to set text of SQL statement and `open` to retrieve records from database.

```c
OraQuery query;

query.setConnection(connection);
query.setCommandText("SELECT * FROM Dept");
query.open();
```

To navigate through records use `next`, `prev`, `first`, `last` and `move` member functions.
Check current record position by **isEOF**, **isBOF**, **recordNo** member functions. To access to fields use **field** function.

```cpp
while (!query.isEOF()) {
    cout << "DName: " << query.field("DName").getString() << endl;
    query.next();
}
```

**Example**

```cpp
#include <iostream>
#include "ocl.h"
using namespace std;
using namespace ocl;

int main() {
    OraConnection connection;
    OraCommand cmd(connection);
    OraQuery query(connection);
    int deptNo;
    connection.open("scott/tiger@ora");
    cmd.setSQL("begin SELECT Max(DeptNo) INTO :MaxNo FROM Dept; end;");
    cmd["MaxNo"].setDataType(dtInt);
    deptNo = cmd["MaxNo"].getInt();
    cmd.setCommandText("INSERT INTO Dept(DeptNo, DName) VALUES (:DeptNo, :DName)");
    cmd["DeptNo"].setInt(deptNo + 1);
    cmd["DName"].setString("Research");
    cmd.execute();
    query.setCommandText("SELECT * FROM Dept");
    query.open();
    while (!query.isEOF()) {
        cout << "DName: " << query.field("DName").getString() << endl;
        query.next();
    }
    query.close();
    connection.close();
    return 0;
}
```

**See Also**

**OraConnection, OraCommand, OraQuery**

**OCL | Using OCL | Index**
7.2 Working with connection

OCL has special class to establish and manage a connection. This class is OraConnection. All OCL objects which interact with the database server require OraConnection object. To use OraConnection and other OCL classes you have to include ocl.h header file.

```c
#include "ocl.h"
```

Then you declare an instance of the OraConnection class.

```c
OraConnection connection;
```

Before connecting to Oracle you need to set up login information with setUserame, setPassword and setServer member functions. Remember to specify an existing TNS alias when calling setServer method.

```c
connection.setUserName("scott");
connection.setPassword("tiger");
connection.setServer("ora");
```

To establish a connection call connect member function.

```c
connection.open();
```

Now you may execute any SQL or PL/SQL statement, obtain results from queries or call stored procedures. This is accomplished by first associating objects of OraCommand or OraQuery classes with the OraConnection instance through their setConnection on member functions or in constructor.

```c
OraCommand cmd;
cmd.setConnection(connection);
```

Then you finished accessing database close the connection by calling disconnect member function.

```c
connection.close();
```

OraConnection allows you to control transactions in your programs. To explicitly start new transaction use startTransaction method. With the commit function you can afterwards finish the transaction and make all changes permanent. Or you may use the rollback function to drop data changes you have made for the current transaction.

Example

```c
OraConnection connection();
connection.setUserName("scott");
connection.setPassword("tiger");
connection.setServer("ora");
```
connection.open();
. . .
connection.startTransaction();
. . .
connection.commit();
. . .
connection.rollback();
. . .
connection.close();

See Also

OraConnection, OraCommand, OraQuery

OCL | Using OCL | Index
## 7.3 Working with command

OCL lets you effortlessly execute your SQL or PL/SQL statements which may return recordsets for further processing in your application. OraCommand class provides methods to execute either complete SQL statement or just the name of the Oracle table or stored procedure. OCL allows you to call any stored procedure or function easily, including procedures and functions in packages. It is enough to know only the name. To execute stored procedure set command type to ctStoredProc before calling `execute` method. For example, to insert new record into DEPT table with DEPT_INSERT procedure:

```sql
CREATE OR REPLACE PROCEDURE DEPT_INSERT (    p_DeptNo NUMBER,    p_DName VARCHAR2,    p_Loc VARCHAR2)  is  begin    INSERT INTO Scott.Dept(DeptNo, DName, Loc)      VALUES (p_DeptNo, p_DName, p_Loc);   end;
```

you can write this code:

```java
OraConnection connection;  OraCommand cmd;  . . .  cmd.setConnection(connection);  cmd.setCommandText("DEPT_INSERT");  cmd.setCommandType(ctStoredProc);  cmd.param("p_DeptNo").setInt(50);  cmd.param("p_DName").setString("DEVELOPMENT");  cmd.param("p_Loc").setString("LONDON");  cmd.execute();
```

If you call stored function you can access to function result by parameter with "Result" name.

OCL gives ability to fetch rows from REF CURSOR parameter. This feature is available with OraRecordset. To fetch records from the REF CURSOR parameter call `executeQuery` method that will return recordset for this cursor. In proc demo project you can find an example of how to work with this type of procedures.

### See Also

- `OraCommand::setCommandText`
- `OraCommand::setCommandType`
- `OraCommand::param`
- `OraCommand::execute`

OCL | Using OCL | Index
7.4 Working with parameters

Parameters in OCL are associated with placeholders in SQL and PL/SQL statements and have a set of methods to store and retrieve relevant data.

OraCommand class manages the way placeholders are resolved into instances of OraParam class. This is done implicitly when a new SQL or PL/SQL statement is assigned to the OraCommand object either through constructor or using setCommandText method.

Individual parameters and their values are maintained by the methods of OraParam class. Objects of this class may be used to store various types of data ranging from character types to PL/SQL tables.

For example, first of all let's construct a new instance of OraCommand class:

```java
OraConnection connection("scott/tiger@ora");
OraCommand cmd(connection);
```

Then we will set up an actual SQL statement to be executed by the Oracle:

```java
cmd.setCommandText("BEGIN"
    "  SELECT * INTO :DeptNo, :DName, :Loc"
    "    FROM Dept"
    "    WHERE DeptNo = :DeptNo;"
    "END");
```

Here :DeptNo, :DName and :Loc are placeholders for the values to be passed to and from the database. Now we need to describe them more precisely and when it is required to set their initial values:

```java
    cmd.param("DeptNo").setDataType(dtInt);
    cmd.param("DeptNo").setInt(10);
    cmd.param("DName").setDataType(dtString);
    cmd.param("Loc").setDataType(dtString);
```

We accessed individual parameters by specifying their names in `param` method. Alternatively we could as well provide a numeric value which would tell `param` method to look for a corresponding placeholder position in command text. The following code is equivalent to the previous:

```java
    cmd.param(0).setDataType(dtInt);
    cmd.param(0).setInt(10);
    cmd.param(1).setDataType(dtString);
    cmd.param(2).setDataType(dtString);
```

At this moment we are ready to execute the statement:

```java
    cmd.execute();
```

Oracle reads input parameter values and places results into the output parameters. To retrieve these values into our program we will just use `get`-methods of OraParam class:
cout << cmd.param("deptno").getInt() << endl;
cout << cmd.param("dname").getChars() << endl;

When you call stored procedures OCL implicitly provides necessary parameter placeholders. For more information see [Working with stored procedures](#).

**See Also**

[OraConnection](#), [OraCommand](#), [OraParam](#), [Working with stored procedures](#)

[OCL](#) | [Using OCL](#) | [Index](#)
7.5 Working with recordset

OraRecordset class covers most of data access related tasks in your applications. Its main purpose is to hold results after execution of any SQL or PL/SQL statements. If the statement, such as SELECT, returns a set of rows OraRecordset can receive and store them. This class provides the way to access to any record in resultset and to any of its fields.

OraRecordset object is constructed implicitly when OraRecordset.execute method is called. You may use a OraRecordset pointer to pass it to the execute method and get an instance of this class. For example:

```csharp
OraConnection connection("scott/tiger@ora");
OraRecordset* record;
OraCommand cmd(connection);
...
cmd.execute(&record);
```

Use a field member function to get access to the fields of current record. You have to provide either the field name or its index. For example, you can write this code to get the value of DNAME field:

```csharp
name = cmd.field("DNAME").getString();
```

or to get integer value of the first field:

```csharp
num = cmd.field(1).getInt();
```

OraRecordset has many functions to navigate through records such as next, prev, first, last and move. You should use sEOF, sBOF and recordNo member functions to check current record position.

For example, to process records from the first to the last you can write this loop:

```csharp
cmd.first();
while (!cmd.isEOF()) {
    ...cmd.next();
}
```

By default OraRecordset stores in memory only the current record and prevents us from returning to the previous records. To circumvent this default behaviour we may instruct OraRecordset object to allocate buffers for all fetched records. Doing so OCL effectively provides local cache which may apart from allowing backward navigation also improve performance and reduce network traffic. Turn on cache as follows:

```csharp
cmd.setCached(true);
```

To release all allocated resources call close member function.

```csharp
cmd.close();
```
See Also

OraRecordset

OCL | Using OCL | Index
7.6 Working with fields

OraField class is generally an encapsulation for the values contained in a single field of a recordset. You get access to the fields when you have executed an SQL or PL/SQL statement which returns recordset. Immediately after opening a recordset all its fields hold the first record values. To instruct recordset object to fetch other record or even jump to a specific record you may use navigational methods of CRDataset class which is an ancestor of the OraRecordset class. To get or set individual fields for the record you may use methods provided by OraField class. For example, given the following code that retrieves records from the Dept table:

```cpp
OraConnection connection("scott/tiger@ora");
OraCommand cmd(connection);
cmd.setCommandText("SELECT * FROM Dept");

OraRecordset& rs = cmd.executeQuery();
```

you may now use a field member function to get access to the fields of current record. You have to provide either the field name or its index. For example, you can write this code to get the value of DNAME field:

```cpp
name = rs.field("DNAME").getString();
```

or to get integer value of the first field:

```cpp
num = rs.field(1).getInt();
```

To assign new values to the fields of the record use set-methods. For example:

```cpp
rs.field("DNAME").setString("Marketing");
rs.field(1).setInt(10000);
```

See Also

- OraRecordset

OCL | Using OCL | Index
7.7 Working with query

OraQuery class is directly derived from the OraCommand and OraRecordset classes. This leads to encapsulation of the following features all in one place: preparing and executing SQL and PL/SQL statements, using parameters, keeping and handling recordsets and their individual fields.

Use OraQuery class if your SQL statement returns recordset as is the case with the SELECT statement. Otherwise you may consider using OraCommand class which doesn't take an overhead of storing and handling recordsets.

Following example demonstrates the common usage of OraQuery:

```java
OraConnection connection("scott/tiger@ora");
OraQuery query(connection);
query.setCommandText("SELECT * FROM Dept WHERE DeptNo > :DeptNo");

Here :DeptNo is a placeholder for the value to be passed to the database. Now we need to set its initial value:

query.param("DeptNo").setDataType(dtInt);
query.param("DeptNo").setInt(20);

Now we may post this query to the server by calling open method:

query.open();

Then we should verify whether there are records returned by this query. If so we will cycle through every record reading its fields. We will stop cycling when we reach the last returned record in the recordset:

if(query.isRowsReturn())
    while (!query.isEOF())
    {
        name = query.field("DNAME").getString();
        num = query.field("DEPTNO").getInt();
        ...
        // Make your processing
        ...
        query.next();
    }

Finally to release all allocated resources call close member function:

query.close();

Note that by default OraQuery stores in memory only the current record and prevents us from returning to the previous records. To circumvent this default behaviour we may instruct OraQuery object to allocate buffers for all fetched records. Doing so OCL effectively provides local cache which may apart from allowing backward navigation also improve performance and reduce network traffic. Turn on cache as follows:

```
query.setCached(true);

OCL gives ability to fetch rows from REF CURSOR parameter. To fetch records from the REF CURSOR parameter call `executeQuery` method that will return recordset for this cursor. In `proc` demo project you can find an example of how to work with this type of procedures.

**See Also**

OraQuery, OraCommand, OraRecordset, Proc demo project

OCL | Using OCL | Index
7.8 Working with stored procedures

OCL allows you to call any stored procedure or function easily, including procedures and functions in packages. It is well enough to provide only the name of procedure. To execute stored procedure set command type to ctStoredProc before calling execute method. For example, to insert new record into DEPT table by calling DEPT_INSERT procedure:

```sql
CREATE OR REPLACE
PROCEDURE DEPT_INSERT (    p_DeptNo NUMBER,    p_DName VARCHAR2,    p_Loc VARCHAR2)
is
begin
    INSERT INTO Scott.Dept(DeptNo, DName, Loc)
    VALUES (p_DeptNo, p_DName, p_Loc);
end;
```

you can write this code:

```java
OraConnection connection("scott/tiger@ora");
OraCommand cmd;
    ...
    cmd.setConnection(connection);
    cmd.createProcCall("DEPT_INSERT");
    cmd.param("p_DeptNo").setInt(50);
    cmd.param("p_DName").setString("DEVELOPMENT");
    cmd.param("p_Loc").setString("LONDON");
    cmd.execute();
```

If you call stored function you can access to its result by looking into parameter with the "Result" name.

OCL gives ability to fetch rows from REF CURSOR parameter. This feature is available with OraRecordset. To fetch records from the REF CURSOR parameter call executeQuery method that will return recordset for this cursor. At proc demo project you can find example of calling such procedure.

See Also

OraCommand::setCommandText, OraCommand::setCommandType, OraCommand::param, OraCommand::execute, Proc demo project

OCL | Using OCL | Index
### 7.9 Working with SQL scripts

Often it is necessary to execute several SQL statements one by one. Sometimes it can be performed by anonymous PL/SQL blocks. But unfortunately it doesn't work in all cases. For example DDL statements cannot be used in PL/SQL. Another way is using a lot of objects such as `OraCommand`. Usually it isn't a good solution. With only one `OraScript` object you can execute several SQL statements as one. This sequence of statements is named script. To separate single statements use semicolon (`;`) or slash (`/`) and for PL/SQL only slash. Note slash must be the first character in the line. `OraScript` implements methods to execute a series of SQL statements separated by special symbols, like SQL*Plus scripts.

Following code shows how execute simple script:

```java
...  // Connection that will be used for script execution  OraConnection conn("scott/tiger@ora");

// Create OraScript instance and specify connection  OraScript script(&conn);

// Set script text  script.setSQL("DROP TABLE DEPT;\n"  "CREATE TABLE DEPT (\n"  " DEPTNO NUMBER(2) CONSTRAINT PK_DEPT PRIMARY KEY,\n"  " DNAME VARCHAR2(14),\n"  " LOC VARCHAR2(13)\n"  ");\n"  ";\n"  "INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK');\n"  "INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');\n"  "INSERT INTO DEPT VALUES (30,'SALES','CHICAGO');\n"  "INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON');\n"  ");

// Execute script and ignore all errors  script.execute(true);

// Close connection  conn.close();
...```

**See Also**

- `OraScript`, `OraConnection`
- [OCL](#) | [Using OCL](#) | [Index](#)
7.10 Working with OraLoader

Sometimes you need to put large amount of data to Oracle database. Certainly you may construct INSERT SQL statement and execute it with OraCommand object. But it takes a lot of time. You can greatly speed up loading time of data by using DML array features. Oracle 8i has better way to do it. With Oracle 8i it is possible to use direct path load interface. The direct path load interface allows to access the direct path load engine of the Oracle database server to perform the functions of the Oracle SQL*Loader utility. This functionality provides the ability to load data from external files into Oracle database objects, either a table or a partition of a partitioned table.

OCL simplifies using direct path load interface by OraLoader class. OraLoader allows you to load various formatted data. The capability of OraLoader object to load various formatted data is reached by reading external data in itself writing method.

Suppose you have the following table:

```sql
CREATE TABLE OCL_LOADER (    ID NUMBER,    STR_T VARCHAR(20),    INT_T NUMBER(6),    DOUBLE_T NUMBER,    DATE_T DATE
)
```

To load data to this table you may write such code:

```cpp
...  // Set up connection that will be used for loading  // and loading table name  OraLoader loader(connection, "ocl_loader");  
// Auto-create columns for specified table  loader.createColumns();  
// Prepare loader for loading  loader.beginLoad();  
cout << "Loading...\n";  
srand((unsigned)time(NULL));  
for (int i = 0; i < 100000; i++) {  
  loader.putColumnData(0, i);  
  loader.putColumnData(1, "loader demo string");  
  loader.putColumnData(2, rand());  
  loader.putColumnData(3, (double)rand()/10);  
  loader.putColumnData(4, CRDate(2002, 04, 11));  
  // If loader buffer is full flush it to the table
```
loader.putNextRow();

// Flush existing data and clear internal loader structures
loader.finishLoad();

...

Note that almost all OraLoader methods may throw an exception, therefore you must use C++ exception handling mechanism.

See Also

OraLoader, LoaderColumn, loader demo project

OCL | Using OCL | Index
7.11 Working with external procedures

OCL may be used in your external procedures to manipulate data and call stored procedures. This is all done in a single service context that is passed as a parameter to external procedures.

For example the following declaration of external procedure:

```
CREATE OR REPLACE
PROCEDURE MyExtProc (x BINARY_INTEGER, y VARCHAR2)  AS EXTERNAL
  NAME "C_demoExternal"
  LIBRARY MyLib
  WITH CONTEXT
  PARAMETERS(CONTEXT, x INT, y STRING);
```

and this declaration of MyLib library associated with external DemoLib.DLL library:

```
CREATE LIBRARY MyLib
  AS 'C:\MyDLLs\DemoLib.DLL';
```

will require the following declaration of C function C_demoExternal in DemoLib.DLL:

```
#include "ocl.h"

extern "C" int __declspec(dllexport) externalProcedure(OCIExtProcContext* context, int x, char* y)
{
  OraConnection connection;

  initExtProcSession(context, connection);

  OraQuery query(connection);

  ...
}
```

To associate connection object with the context handle use `nExtProcSess` on global function. Then you may create instances of other classes such as OraCommand and OraQuery and execute your SQL and PL/SQL statements as you would normally do in client applications.

Using external procedures this way gives your applications the following benefits:
- Faster processing since external procedures are located on server side.
- Reduced network round-trip.
- Reusable code is even more reusable as all clients work with only one instance of external procedures.
- Maintenance is only required on server side.

See Also

ExtProc demo project
7.12 Monitoring SQL statements

To extend monitoring capabilities of OCL-based applications there is an additional tool called DB Monitor.

OCL uses DB Monitor tool to trace SQL activity in your applications. You may gather the following information in real time and use it while debugging:

- Connection open/close events.
- Actual SQL and PL/SQL statements passed to the server.
- Parameter values passed to and from the server.
- Error and other messages generated by the server.
- Exact timing information for every monitoring event.

To make use of monitoring in OCL applications call `OraMonitor::start` static function. Then launch DB Monitor tool and it displays all the information on SQL activity in your applications.

OCL | Using OCL | Index
7.13 Data types

OCL uses data type constants to identify types of field and parameters. The following table describes data type mapping used in OCL:

<table>
<thead>
<tr>
<th>OCL type</th>
<th>C++ type</th>
<th>Oracle type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtUnknown</td>
<td>unknown type</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>dtString</td>
<td>char*, CRString</td>
<td>CHAR</td>
</tr>
<tr>
<td>dtFixedString</td>
<td>char*, CRString</td>
<td>CHAR</td>
</tr>
<tr>
<td>dtInteger</td>
<td>int</td>
<td>NUMBER</td>
</tr>
<tr>
<td>dtFloat</td>
<td>float</td>
<td>NUMBER</td>
</tr>
<tr>
<td>dtDouble</td>
<td>double</td>
<td>NUMBER</td>
</tr>
<tr>
<td>dtDate</td>
<td>CRDate</td>
<td>DATE</td>
</tr>
<tr>
<td>dtBlob</td>
<td>OraLob</td>
<td>BLOB</td>
</tr>
<tr>
<td>dtClob</td>
<td>OraLob</td>
<td>CLOB</td>
</tr>
<tr>
<td>dtLong</td>
<td>OraLong</td>
<td>LONG</td>
</tr>
<tr>
<td>dtLongRaw</td>
<td>OraLong</td>
<td>LONG RAW</td>
</tr>
<tr>
<td>dtRowId</td>
<td>char*, CRString</td>
<td>ROWID</td>
</tr>
<tr>
<td>dtCursor</td>
<td>OraCursor</td>
<td>REF CURSOR</td>
</tr>
<tr>
<td>dtTimeStamp</td>
<td>OraTimeStamp</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>dtTimeStampTZ</td>
<td>OraTimeStamp</td>
<td>TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>dtTimeStampLTZ</td>
<td>OraTimeStamp</td>
<td>TIMESTAMP WITH LOCAL TIME ZONE</td>
</tr>
<tr>
<td>dtIntervalYM</td>
<td>OraInterval</td>
<td>INTERVAL YEAR TO MONTH</td>
</tr>
<tr>
<td>dtIntervalDS</td>
<td>OraInterval</td>
<td>INTERVAL DAY TO SECOND</td>
</tr>
</tbody>
</table>

See Also

CRField::dataType, OraParam::dataType, OraParam::setDataType
7.14 Exceptions

OCL uses exceptions to provide application with error information. Each failed OCL call raises the exception. When it is Oracle error OCL raises OraException object. In another cases it might be CRException object. You can process OCL exception in your own way. For example, in console application you can just print error message.

```cpp
try {
  ...
  query.open();
  ...
}
catch (OraException& e) {
  cout << e.message();
}
```

See Also

OraException

OCL | Using OCL | Index
8 OCL Classes

CRLob
CRDataset
CRDate
CRException
CRField
CRFIELDS
CRMemDataset
CRSharedObject
CRString
CRValue
LoaderColumn
OraCommand
OraConnection
OraCursor
OraEnvironment
OraException
OraField
OraFile
OraInterval
OraLoader
OraLob
OraMonitor
OraParam
OraParams
OraQuery
OraRecordset
OraScript
OraTimeStamp
OraTransaction

OCL | Index
8.1 CRDataset

#include "dataset.h"

Members

Construct on

CRDataset();
virtual ~CRDataset();

Open / close

void open();
void close();
virtual void refresh();

bool isActive() const;

Fields

virtual void initFields();

int fieldCount() const;
CRIFields& fields();
CRIField& field(int index) const;
CRIField& field(const char* name) const;
CRIField* findField(int index) const;
CRIField* findField(const char* name) const;

Navigation

virtual bool isBOF() const;
virtual bool isEOF() const;

void first();
void last();
void next();
void prev();
virtual int move(long distance);
virtual void moveTo(long recNo);

long recordCount() const;
virtual long recordNo() const;

Bookmarks

virtual void getBookmark(void** bookmark);
virtual bool moveToBookmark(void* bookmark);
void freeBookmark(void* bookmark);

DESCRIPTION

This class implements basic functionality to manipulate a set of records. CRDataset
defines a lot of field access and navigation functions. Usually application uses
CRDataset’s descendants such as OraRecordset objects.

See Also

OraRecordset, CRField, CRDataset::open

Classes | OCL | Index

8.1.1 CRDataset::close

void close();

DESCRIPTION

Call this member function to close the dataset. You can call open member function again
after calling close. This lets you reuse the dataset object. The alternative is to call
refresh.

See Also

CRDataset::open, CRDataset::refresh

CRDataset | Classes | OCL | Index

8.1.2 CRDataset::CRDataset

CRDataset();

DESCRIPTION

This member function constructs a new CRDataset object.

See Also

CRDataset::open
8.1.3 `CRDataset::field`

```
CRField& field(int index) const;
CRField& field(const CRString& name) const;
```

**Return Value**

The reference to CRField object for the specified field.

**Parameters**

- `index`
  - Zero-based index number of the field.
- `name`
  - The name of the field.

**Description**

Call this member function to retrieve the field when its name or its index is known. If the specified field is not found, the `field` function throws CRException exception.

**Example**

```
CRString address;
// get value of the field with the name address
address = dataset1.field("Address").getString();
// set string to the field with index 0
dataset2.field(0).assign(address);
```

**See Also**

[CRField](#), [CRDataset::findField](#), [CRDataset::fields](#), [CRDataset::operator[]](#), [CRException](#)

8.1.4 `CRDataset::fieldCount`

```
int fieldCount() const;
```

**Return Value**

The number of fields in the dataset.

**Description**

- Zero-based index number of the field.
- The name of the field.
Call this member function to determine the number of fields associated with the dataset.

See Also

CRDataset::field, CRDataset::fields

8.1.5 CRDataset::fields

CRFields& fields();

Return Value

The reference to CRFields object associated with the dataset.

Description

Call this member function to retrieve the fields of the dataset.

Example

CRString address;
// get value of the field with the name address
address = dataset1.fields.item("Address").getString();
// set string to the field with index 0
dataset2.fields.item(0).assign(address);

See Also

CRFields, CRDataset::field

8.1.6 CRDataset::findField

CRField* findField(int index) const;
CRField* findField(const CRString& name) const;

Return Value

The pointer to CRField object for the specified field.

Parameters

index
Zero-based index number of the field.
**name**
Name of the field.

**Description**
Call this member function to determine if a specified field component exists in the dataset. If `findField` finds a field with a matching name, it returns the CRField object for the specified field. Otherwise it returns NULL. `findField` is a useful function to call prior to calling other dataset functions, such as `field`, that require a valid field name as a parameter.

**Example**
```cpp
crstring address;
if (dataset1.findField("Address"))
    address = dataset1.findField("Address") -> getString();
```

**See Also**
- `CRField`, `CRDataset::field`
- `CRDataset` | `Classes` | `OCL` | `Index`

### 8.1.7 `CRDataset::first`

```cpp
void first();
```

**Description**
Call this member function to position the current record pointer to the first record in the dataset. To find out if the current record pointer is positioned on the first record of the dataset use `isBOF` function.

**See Also**
- `CRDataset::last`, `CRDataset::next`, `CRDataset::prev`, `CRDataset::isBOF`
- `CRDataset` | `Classes` | `OCL` | `Index`

### 8.1.8 `CRDataset::freeBookmark`

```cpp
void freeBookmark(void* bookmark);
```

**Parameters**
**bookmark**

Pointer to memory in which to return bookmark.

**Description**

Use this member function to release the memory allocated by `getBookmark` for specified bookmark when it is no longer needed.

**See Also**

`CRDataset::getBookmark`, `CRDataset::moveToBookmark`

---

### 8.1.9 `CRDataset::getBookmark`

```cpp
void getBookmark(void** bookmark);
```

**Parameters**

- **bookmark**
  
  Pointer to memory in which to return bookmark.

**Description**

This function returns bookmark to current record. If bookmark parameter is NULL, `getBookmark` allocates memory to store bookmark. After using bookmark you need call `freeBookmark` member function to release memory. You can use `moveToBookmark` to go to record identified by bookmark.

**Example**

```cpp
void* bookmark = NULL;
dataset.getBookmark(&bookmark);
... dataset.moveToBookmark(bookmark);
dataset.freeBookmark(bookmark);
```

**See Also**

`CRDataset::moveToBookmark`, `CRDataset::freeBookmark`

---
8.1.10  CRDataset::initFields

virtual void initFields();

Description
Creates field objects for the dataset. There is usually no need to call initFields directly.

See Also
CRDataset | Classes | OCL | Index

8.1.11  CRDataset::isActive

bool isActive() const;

Return Value
True if the dataset object’s open or refresh member function has previously been called and the dataset has not been closed, otherwise false.

Description
Call this member function to determine if the dataset is already open.

See Also
CRDataset::open, CRDataset::close, CRDataset::refresh

CRDataset | Classes | OCL | Index

8.1.12  CRDataset::isBOF

virtual bool isBOF() const;

Return Value
True if the dataset contains no records or if you have scrolled backward before the first record, otherwise false.

Description
Call this member function before you scroll from record to record to learn if the current record pointer is positioned to the first record of the dataset. You can also use sBOF with sEOF to determine whether the dataset contains any records or is empty.

Example
dataset.open();
if (dataset.isBOF())
    return; // dataset is empty

// scroll to last record
dataset.last();

// scroll to first record
while (!dataset.isBOF())
    dataset.prev();

See Also

CRDataset::isEOF, CRDataset::first, CRDataset::next

8.1.13 CRDataset::isEOF

virtual bool isEOF() const;

Return Value

True if the dataset contains no records or if you have scrolled beyond the last record, otherwise false.

Description

Call this member function as you scroll from record to record to find out if the current record pointer is positioned to the last record of the dataset. You can also use sEOF to determine whether the dataset contains any records or is empty.

Example

dataset.open();
if (dataset.isEOF())
    return; // dataset is empty

// scroll to last record
while (!dataset.isEOF())
    dataset.next();

See Also

CRDataset::isBOF, CRDataset::last, CRDataset::next

CRDataset | Classes | OCL | Index
8.1.14 CRDataset::last

```cpp
void last();
```

**Description**

Call this member function to position the current record pointer to the last record in the dataset. To learn if the current record pointer is positioned beyond the last record of the dataset use `isEOF` function.

**See Also**

- CRDataset::first
- CRDataset::prev
- CRDataset::next
- CRDataset::isEOF

8.1.15 CRDataset::move

```cpp
int move(long distance);
```

**Return Value**

The actual number of moved records. In most cases, value is the absolute value of distance, but if the move encounters the beginning-of-file or end-of-file before moving distance records, result will be less than the absolute value of distance.

**Parameters**

- `distance`
  - The number of rows to move forward or backward. Positive values move forward, toward the end of the dataset. Negative values move backward, toward the beginning.

**Description**

Call this member function to move the current record pointer within the dataset, either forward or backward.

**See Also**

- CRDataset::prev
- CRDataset::next
- CRDataset::moveTo
- CRDataset::recordNo
8.1.16 CRDataset::moveTo

virtual void moveTo(long recNo);

**Parameters**

*RecNo*

The one-based ordinal position for the current record in the dataset.

**Description**

Call this member function to position the dataset to the record corresponding to the specified record number. **moveTo** moves the current record pointer based on this ordinal position.

**See Also**

CRDataset::move, CRDataset::recordNo

8.1.17 CRDataset::moveToBookmark

bool moveToBookmark(void* bookmark);

**Return Value**

True if bookmark is valid and moveToBookmark found specified record, otherwise false.

**Parameters**

*bookmark*

Pointer to memory in which to return bookmark.

**Description**

This member function allows to back to record identified by bookmark. Bookmark must be valid pointer returned by **getBookmark**.

**See Also**

CRDataset::getBookmark, CRDataset::freeBookmark
8.1.18  CRDataset::next

void next();

**Description**

Call this member function to position the current record pointer to the next record in the dataset. Use **sEOF** before calling **next** to learn if the last record of the dataset is reached.

**See Also**

CRDataset::last, CRDataset::first, CRDataset::prev, CRDataset::isEOF

8.1.19  CRDataset::open

void open();

**Description**

Call this member function to open dataset object. After calling this function **sAct ve** returns true.

**See Also**

CRDataset::close, CRDataset::resfesh

8.1.20  CRDataset::operator []

CRField& operator [](int index) const;
CRField& operator[](const CRString& name) const;

**Return Value**

The reference to CRField object for the specified field.

**Parameters**

*index*

Zero-based index number of the field.
name
The name of the field.

**Descript on**

These operators allow to retrieve a field when its name or its index is known. If the specified field is not found they throw the exception.

**Example**

```cpp
CRString address;
// get value of the field with the name address
address = dataset1["Address"].getString();
// set string to the field with index 0
dataset2[0].assign(address);
```

**See Also**

CRField, CCRDataSet::field

**8.1.21 CRDataset::prev**

```cpp
void prev();
```

**Descript on**

Call this member function to position the current record pointer to the previous record in the dataset. Use **sBOF** before calling **prev** to learn if the first record of the dataset is reached.

**See Also**

CRDataset::first, CRDataset::last, CRDataset::next, CRDataset::isBOF

**8.1.22 CRDataset::recordCount**

```cpp
long recordCount() const;
```

**Return Value**

The total number of records in the dataset.

**Descript on**
Call this member function to determine how many records are in the dataset.

**See Also**

CRDataset::recordNo

8.1.23 **CRDataset::recordNo**

virtual long recordNo() const;

**Return Value**

The number of current record in the dataset.

**Description**

Call this member function to determine record number of the current record in the dataset. Applications might use this function with recordCount to iterate through all the records in the dataset, though typically record iteration is handled with calls to first, last, move, next, and prev.

**See Also**

CRDataset::moveTo, CRDataset::recordCount

8.1.24 **CRDataset::refresh**

void refresh();

**Description**

Call this member function to refresh the dataset. If any records are returned, the first record becomes the current record.

**See Also**

CRDataset::open, CRDataset::close
8.2 CRDate

```
#include "ocldef.h"

Members

Construct on

CRDate();
CRDate(unsigned year, unsigned month, unsigned day, unsigned hour = 0, unsigned
minute = 0, unsigned second = 0);

Attributes

unsigned year() const;
void setYear(unsigned year);

unsigned month() const;
void setMonth(unsigned month);

unsigned day() const;
void setDay(unsigned day);

unsigned hour() const;
void setHour(unsigned hour);

unsigned minute() const;
void setMinute(unsigned minute);

unsigned second() const;
void setSecond(unsigned second);

unsigned dayOfWeek();

Methods

bool operator==(const CRDate& value) const;

CRString format(const char* format = NULL) const;
void now();

static CRString dateFormat();
static void setDateFormat(const CRString& value);

Description

CRDate represents date and time value. You can create CRDate object yourself or get it
by getDate function of CRField or OraParam class. Use member functions to access to
part of date and time such as day, month, minute and others. Use \texttt{format} function to convert date to string.

\textbf{See Also}

\texttt{CRValue, CRString}

\textbf{Classes} | \texttt{OCL} | \texttt{Index}

\section{8.2.1 \texttt{CRDate::CRDate}}

\texttt{CRDate();}
\texttt{CRDate(unsigned year, unsigned month, unsigned day, unsigned hour = 0, unsigned minute = 0, unsigned second = 0);}

\textbf{Parameters}

\textit{year, month, day, hour, minute, second}  
Indicate the date and time values to be copied into the new \texttt{CRDate} object.

\textbf{Description}

This member function constructs a new \texttt{CRDate} object initialized with the specified absolute time.

\textbf{Example}

\begin{verbatim}
// Set 31.12.1999 23:59:59 value to date object
CRDate date(1999, 12, 31, 23, 59, 59);
\end{verbatim}

\textbf{See Also}

\texttt{CRDate::setYear}

\texttt{CRDate} | \texttt{Classes} | \texttt{OCL} | \texttt{Index}

\section{8.2.2 \texttt{CRDate::dateFormat}}

\texttt{static CRString dateFormat();}

\textbf{Return Value}

String representation for the date formatting template.

\textbf{Description}

Returns the date formatting string. For the description of formatting codes refer to \texttt{format} method.
Note On Windows platforms this method calls `GetLocaleInfo` API function to retrieve short date formatting string. See Platform SDK: International Features for more information.

See Also

`CRDate::format`, `CRDate::setDateFormat`

`CRDate` | Classes | OCL | Index

### 8.2.3 `CRDate::day`

```cpp
unsigned day() const;
```

**Return Value**

Day of the month.

**Description**

Returns the day of the month, based on local time, in the range 1 through 31.

See Also

`CRDate::setDay`, `CRDate::dayOfWeek`

`CRDate` | Classes | OCL | Index

### 8.2.4 `CRDate::dayOfWeek`

```cpp
unsigned dayOfWeek();
```

**Return Value**

Day of the week.

**Description**

Returns the day of the week, based on local time, in the range 0 through 6, where 0 is Sunday.

See Also

`CRDate::day`

`CRDate` | Classes | OCL | Index
8.2.5 CRDate::format

CRString format(const char* format = NULL) const;

**Return Value**

CRString object that contains the formatted time.

**Parameters**

*format*

A formatting string is similar to the `printf` formatting string. Formatting codes are replaced by the corresponding CRDate class. Other characters in the formatting string are copied unchanged to the returned string. The value and meaning of the formatting codes for format are listed below:

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Displays</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Displays the day as a number without a leading zero (1-31).</td>
</tr>
<tr>
<td>dd</td>
<td>Displays the day as a number with a leading zero (01-31).</td>
</tr>
<tr>
<td>m</td>
<td>Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.</td>
</tr>
<tr>
<td>mm</td>
<td>Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.</td>
</tr>
<tr>
<td>yy</td>
<td>Displays the year as a two-digit number (00-99).</td>
</tr>
<tr>
<td>yyyy</td>
<td>Displays the year as a four-digit number (0000-9999).</td>
</tr>
<tr>
<td>h</td>
<td>Displays the hour without a leading zero (0-23).</td>
</tr>
<tr>
<td>hh</td>
<td>Displays the hour with a leading zero (00-23).</td>
</tr>
<tr>
<td>n</td>
<td>Displays the minute without a leading zero (0-59).</td>
</tr>
<tr>
<td>nn</td>
<td>Displays the minute with a leading zero (00-59).</td>
</tr>
<tr>
<td>s</td>
<td>Displays the second without a leading zero (0-59).</td>
</tr>
<tr>
<td>ss</td>
<td>Displays the second with a leading zero (00-59).</td>
</tr>
<tr>
<td>t</td>
<td>Displays the time using the format given by the shortTimeFormat global variable.</td>
</tr>
<tr>
<td>tt</td>
<td>Displays the time using the format given by the longTimeFormat global variable.</td>
</tr>
</tbody>
</table>
variable.

hen NULL or "" date is formatted as "dd.mm.yyyy".

**Description**

Call this member function to create a formatted representation of the date/time value.

**Example**

```cpp
CRDate date(1975, 7, 9, 17, 28);
CRString st;

// Format to "09.07.1975"
st = date.format();

// Format to "09-07-1975 17:28"
st = date.format("dd-mm-yyyy hh:nn");
```

**See Also**

- CRDate::now
- CRDate

### 8.2.6 CRDate::hour

`unsigned hour() const;`

**Return Value**

Hour value for the CRDate object.

**Description**

Returns the hour, based on local time, in the range 0 through 23.

**See Also**

- CRDate::setHour

### 8.2.7 CRDate::minute

`unsigned minute() const;`

**Return Value**
Minute value for the CRDate object.

**Description**

Returns the minute, based on local time, in the range 0 through 59.

**See Also**

CRDate::setMinute

CRDate | Classes | OCL | Index

### 8.2.8 CRDate::month

```cpp
unsigned month() const;
```

**Return Value**

Month value for the CRDate object.

**Description**

Returns the month, based on local time, in the range 1 through 12 (1 = January).

**See Also**

CRDate::setMonth

CRDate | Classes | OCL | Index

### 8.2.9 CRDate::now

```cpp
void now();
```

**Description**

Call this member function to assign current date and time to this object.

**See Also**

CRDate::setYear, CRDate::setMonth, CRDate::setDay

CRDate | Classes | OCL | Index
8.2.10 CRDate::operator ==

```cpp
bool operator == (const CRDate& value);
```

**Return Value**

True if value of this object is equal to date in the `value` object.

**Parameters**

`value`

Date object.

**Description**

This operator compares this CRDate object and the `value` object. Returns true if date and time values of the two CRDate objects are equal.

8.2.11 CRDate::second

```cpp
unsigned second() const;
```

**Return Value**

Seconds value for the CRDate object.

**Description**

Returns the second, based on local time, in the range 0 through 59.

**See Also**

CRDate::setSecond

8.2.12 CRDate::setDateFormat

```cpp
static void setDateFormat(const CRString& value);
```

**Parameters**

`value`
New formatting string.

**Description**

Call this member function to set the date formatting string. For the description of formatting codes refer to **format** method.

**See Also**

CRDate::format, CRDate::dateFormat

CRDate | Classes | OCL | Index

### 8.2.13 CRDate::setDay

```cpp
void setDay(unsigned day);
```

**Parameters**

`day`

Day of the month.

**Description**

Call this member function to set day of the month to date object. Valid range is 1 through 31.

**See Also**

CRDate::day

CRDate | Classes | OCL | Index

### 8.2.14 CRDate::setHour

```cpp
void setHour(unsigned hour);
```

**Parameters**

`day`

Hour value.

**Description**

Call this member function to set hour to date object. Valid range is 0 through 23.

**See Also**
8.2.15 **CRDate::setMinute**

void setMinute(unsigned minute);

**Parameters**

*minute*
   
   Minute value.

**Description**

Call this member function to set minute to date object. Valid range is 0 through 59.

**See Also**

CRDate::minute

CRDate | Classes | OCL | Index

8.2.16 **CRDate::setMonth**

void setMonth(unsigned month);

**Parameters**

*month*

   Month value.

**Description**

Call this member function to set month to date object. Valid range is 1 through 12.

**See Also**

CRDate::month

CRDate | Classes | OCL | Index

8.2.17 **CRDate::setSecond**

void setSecond(unsigned second);
Parameters

*second*
Seonds value.

Description
Call this member function to set second to date object. Valid range is 0 through 59.

See Also
- CRDate::second

8.2.18 CRDate::setYear

```cpp
void setYear(unsigned year);
```

Parameters

*year*
Year value.

Description
Call this member function to set year to date object.

See Also
- CRDate::year

8.2.19 CRDate::year

```cpp
unsigned year() const;
```

Return Value
Year value for the CRDate object.

Description
Returns the year, based on local time.

See Also
CRDate::setYear
8.3 CRException

#include "ocldef.h"

Members

Construct on

CRException(const CRString& msg);

Attributes

CRString& message();
void setMessage(const CRString& msg);

Description

The library uses CRException object to throw some errors. Most of errors is represented by their descendants such as CRDBException and OraException.

See Also

OraException

Classes | OCL | Index

8.3.1 CRException::CRException

CRException(const CRString& msg);

Parameters

msg
Indicates the message string to be copied into the new CRException.

Description

This member function constructs a new CRException object.

Example

if (index < count) {
   . . .
} else
   throw CRException("Index is out of range");
See Also

CRException::message

CRException  |  Classes  |  OCL  |  Index

8.3.2 CRException::message

CRString& message();

Return Value

String description for exception.

Description

Returns message string associated with exception object.

See Also

CRException::setMessage

CRException  |  Classes  |  OCL  |  Index

8.3.3 CRException::setMessage

void setMessage(const CRString& msg);

Parameters

msg

Indicates the message string to be copied into the new CRException.

Description

Call this member function to set message string to exception object.

See Also

CRException::message

CRException  |  Classes  |  OCL  |  Index
8.4 CRField

```
#include "dataset.h"

Members

Construct on

CRField();
CRField(const CRString& name, TCRDataType dataType, int size, bool required = false,
int fieldNo = 0);
~CRField();

Attributes

CRDataType dataType() const;
int fieldNo() const;
CRString& name() const;
bool required() const;
int scale() const;
int precision() const;
virtual int size() const;

bool isNull();
void setNull();

CRDate getDate();
void setDate(const CRDate& value);

double getDouble();
void setDouble(double value);

float getFloat();
void setFloat(float value);

int getInt();
void setInt(int value);

const char* getChars();
char* getChars(char* result);
char* getChars(char* result, int length);
void setChars(const char* value);
CRString getString();
void setString(const CRString& value);

CRLob& getLob();
void setLob(const CRLob& value);
```
### CRField::assign

void assign(const CRDate& value);
void assign(int value);
void assign(double value);
void assign(const char* value);
void assign(const CRString& value);
void assign(const CRField& value);

**Parameters**

*value*  
New value for the field.

**Description**

Call this member function to assign the value to the field in the current record.

**Example**

```cpp
dataset.field("DeptNo").assign(60);
```
8.4.2 CRField::CRField

CRField();
CRField(CRString& name, CRDataType dataType, int size, bool required = false, int fieldNo = 0);

Parameters

name
Name of the field.

dataType
Type of the field.

size
Size of field data.

required
True if the field is required, otherwise false.

fieldNo
One-based number of the field.

Description

This member function constructs a new CRField object.

See Also

CRField::name, CRField::dataType, CRField::size, CRField::required, CRField::fieldNo

8.4.3 CRField::dataType

typedef word CRDataType;

CRDataType dataType() const;

Return Value

Field type. May be one of the following values:
dtUnknown, dtString, dtInt32, dtInt64, dtInt, dtInteger, dtFloat, dtDouble,
dtDate, dtBlob, dtClob.

Description

Returns the type of the field.
Example

```c
int value;
if (dataset.field(0).dataType == dtInt)
    value = dataset.field(0).getInt();
```

See Also

Data types

CRField | Classes | OCL | Index

### 8.4.4 CRField::fieldNo

```c
int fieldNo() const;
```

**Return Value**

Ordinal position for the field.

**Description**

Returns ordinal position of the field in the dataset. Number of the first field is 1.

See Also

CRField::name

CRField | Classes | OCL | Index

### 8.4.5 CRField::getWChars

```c
const wchar_t* getChars();
wchar_t* getChars(wchar_t* result);
wchar_t* getChars(wchar_t* result, int length);
```

**Return Value**

Zero-terminated Unicode string of chars.

**Parameters**

- `result`
  - Buffer to hold copy of the field value.
- `length`
  - Number of characters to be copied.
**Description**

Call this member function to retrieve field data in the current record represented as a Unicode string. For the function that does not have `length` parameter, `result` parameter represents a buffer long enough to hold zero-terminated copy of the field value. For `getChars` function with `length` parameter, if `length` is less than or equal to the length of `result`, a null character is not appended automatically to the copied string. If `length` is greater than the length of `result`, the destination string is padded with null characters up to `length` value. Throws an exception when type conversion is not possible. Use this function for connections in Unicode mode only.

**Example**

```cpp
wchar_t* dName;
wchar_t buffer[100];
dName = dataset.field("DName").getWChars();
dataset.field("DName").getWChars(buffer);
```

**See Also**

`CRField::setWString`, `CRField::setWChars`

**8.4.6 CRField::getChars**

```cpp
const char* getChars();
char* getChars(char* result);
char* getChars(char* result, int length);
```

**Return Value**

Zero-terminated string of chars.

**Parameters**

- `result`  
  Buffer to hold copy of the field value.
- `length`  
  Number of characters to be copied.

**Description**

Call this member function to retrieve field data in the current record represented as a string. For the function that does not have `length` parameter, `result` parameter represents a buffer long enough to hold zero-terminated copy of the field value. For `getChars` function with `length` parameter, if `length` is less than or equal to the length of `result`, a null character is not appended automatically to the copied string. If `length` is greater than the length of `result`, the destination string is padded with null characters up to `length`
value. Throws an exception when type conversion is not possible.

**Example**

```c
char* dName;
char buffer[100];

dName = dataset.field("DName").getChars();
dataset.field("DName").getChars(buffer);
```

**See Also**

CRField::setString, CRField::setChars

---

**8.4.7 CRField::getDate**

CRDate getDate();

**Return Value**

CRDate representation for the field.

**Description**

Call this member function to retrieve field data in the current record represented as CRDate. Throws an exception when type conversion is not possible.

**Example**

```c
CRDate hireDate;
hireDate = dataset.field("HireDate").getDate();
```

**See Also**

CRDate, CRField::setDate, CRField::isNull

---

**8.4.8 CRField::getDouble**

double getDouble();

**Return Value**

Double value stored in the field.
**Description**

Call this member function to retrieve field data in the current record represented as double. The value of the field will be converted to double value if possible. Throws an exception when type conversion is not possible.

**Example**

```cpp
double deptNo;
depthNo = dataset.field("DeptNo").getDouble();
```

**See Also**

[CRField::setDouble](#), [CRField::isNull](#)

---

**8.4.9 CRField::getFloat**

float getFloat();

**Return Value**

Float value stored in the field.

**Description**

Call this member function to retrieve field data in the current record represented as float. The value of the field will be converted to float value if possible. Throws an exception when type conversion is not possible.

**Example**

```cpp
float deptNo;
depthNo = dataset.field("DeptNo").getFloat();
```

**See Also**

[CRField::setFloat](#), [CRField::isNull](#)

---

**8.4.10 CRField::getInt**

int getInt();

**Return Value**
Integer value stored in the field.

**Description**

Call this member function to retrieve field data in the current record represented as int. The value of the field will be converted to int value if possible. Throws an exception when type conversion is not possible.

**Example**

```c++
int deptNo;
depthNo = dataset.field("DeptNo").getInt();
```

**See Also**

`CRField::setInt, CRField::isNull`

`CRField | Classes | OCL | Index`

### 8.4.11 `CRField::getLob`

CRLob& getLob();

**Return Value**

Lob representation of the field value.

**Description**

Call this member function to retrieve the reference to CRLob object that represents blob value of the field in the current record. Throws an exception when type conversion is not possible.

**Example**

```c++
CRString desc;
desc = dataset.field("Description").getLob().getString();
```

**See Also**

`CRLob, CRField::setLob, CRField::isNull`

`CRField | Classes | OCL | Index`
8.4.12 **CRField::getWString**

```
CR String get String();
```

Return Value

Unicode string value stored in the field.

Description

Call this member function to retrieve field data in the current record represented as string. No conversion from nonUnicode string to a CR String value are performed. Use this function for connections in Unicode mode only.

Example

```
CRWString dName;
dName = dataset.field("DName").getWString();
```

See Also

[CR String](#), [CRField::setString](#), [CRField::isNull](#)

8.4.13 **CRField::getString**

```
CRString getString();
```

Return Value

String value stored in the field.

Description

Call this member function to retrieve field data in the current record represented as string. The value of the field will be converted to a CRString value if possible. Throws an exception when type conversion is not possible.

Example

```
CRString dName;
dName = dataset.field("DName").getString();
```

See Also
8.4.14  **CRField::isNull**

```cpp
bool isNull();
```

**Return Value**

True if the current field is a NULL value.

**Description**

Call this member function to determine whether the field in the current record is NULL or not.

**Example**

```cpp
if (!dataset.field("DeptNo").isNull())
    deptNo = dataset.field("DeptNo").getInt();
```

**See Also**

CRField::setNull

8.4.15  **CRField::name**

```cpp
CRString name() const;
```

**Return Value**

Name of the field.

**Description**

Returns name of the field.

**See Also**

CRField::fieldNo

CRField | Classes | OCL | Index
8.4.16 **CRField::precision**

```cpp
int precision() const;
```

**Return Value**
Number of significant digits.

**Description**
Returns number of significant digits for the numeric value.

**See Also**
CRField::size, CRField::scale

---

8.4.17 **CRField::required**

```cpp
bool required() const;
```

**Return Value**
True if this field couldn't be NULL.

**Description**
Returns true if field is required, otherwise false.

**See Also**
CRField::dataType

---

8.4.18 **CRField::scale**

```cpp
int scale() const;
```

**Return Value**
Scale value for the field.

**Description**
Returns the scale of the number field.

See Also

CRField::precision, CRField::size

8.4.19 CRField::setWChars

void setWChars(const wchar_t* value);

Parameters

value
Zero-terminated Unicode string of characters.

Description
Call this member function to assign a new string value to the field.

See Also

CRField::setWString, CRField::getWChars

8.4.20 CRField::setChars

void setChars(const char* value);

Parameters

value
Zero-terminated string of characters.

Description
Call this member function to assign a new string value to the field. Use this function for connections in Unicode mode only.

See Also

CRField::setString, CRField::getChars
8.4.21 CRField::setDate

void setDate(const CRDate& value);

**Parameters**

*value*  
CRDate object.

**Description**

Call this member function to assign the CRDate value to the field in the current record.

**See Also**

CRDate, CRField::getDate, CRField::setNull

CRField | Classes | OCL | Index

8.4.22 CRField::setDouble

void setDouble(double value);

**Parameters**

*value*  
New double value.

**Description**

Call this member function to assign the double value to the field in the current record.

**See Also**

CRField::getDouble, CRField::setNull

CRField | Classes | OCL | Index

8.4.23 CRField::setFloat

void setFloat(float value);

**Parameters**

*value*
New float value.

**Description**

Call this member function to assign the float value to the field in the current record.

**See Also**

CRField::getFloat, CRField::setNull

CRField | Classes | OCL | Index

### 8.4.24 CRField::setInt

void setInt(int value);

**Parameters**

*value*

New integer value.

**Description**

Call this member function to assign the int value to the field in the current record.

**Example**

```cpp
dataset.field("DeptNo").setInt(60);
```

**See Also**

CRField::getInt, CRField::setNull

CRField | Classes | OCL | Index

### 8.4.25 CRField::setLob

void setLob(const CRLob& value);

**Parameters**

*value*

Lob object.

**Description**

Call this member function to assign CRLob object to the field in the current record.
See Also

CRLob, CRField::getLob, CRField::setNull

8.4.26 CRField::setNull

void setNull();

Description
Call this member function to assign a NULL value to the field in the current record.

See Also

CRField::isNull

8.4.27 CRField::setWString

void setWString(const CRWString& value);

Parameters

value
New CR WString value.

Description
Call this member function to assign a Unicode string value to the field in the current record. Use this function for connections in Unicode mode only.

See Also

CR WString, CRField::get WString, CRField::setNull

8.4.28 CRField::setString

void setString(const CRIString& value);

Parameters
**value**
New CRString value.

**Describe on**
Call this member function to assign a string value to the field in the current record. Use this function for connections in Unicode mode only.

**See Also**
CRString, CRField::getString, CRField::setNull

**8.4.29 CRField::size**

```cpp
int size() const;
```

**Return Value**
Size of the field value.

**Describe on**
Returns data size of the field.

**See Also**
CRField::scale

CRField | Classes | OCL | Index
8.5 CRFields

#include "crdb.h"

Members

Attributes

int count() const;

Operations

CRField* findItem(int index) const;
CRField* findItem(const CRString& name) const;
CRField& item(int index) const;
CRField& item(const CRString& name) const;

CRField& operator[](int index) const;
CRField& operator[](const CRString& name) const;

Description

CRFields is used by dataset to manage the field objects that correspond to fields in the database table.

See Also

CRField, CRDataset::fields

Classes | OCL | Index

8.5.1 CRFields::count

int count() const;

Return Value

The number of fields in the fields object.

Description

Call this member function to determine the number of fields referenced by the CRFields object.

See Also
8.5.2 **CRFields::findItem**

CRField* findItem(int index) const;
CRField* findItem(const CRString& name) const;

**Return Value**
A pointer to CRField object for the specified field.

**Parameters**

*index*
Zero-based index number of the field.

*name*
Name of the field.

**Description**
Call this member function to determine if a specified field is referenced in the CRFields object. If **findItem** finds a field with a matching name, it returns the CRField object for the specified field. Otherwise it returns NULL. **findItem** is a useful function to call prior to calling other CRFields functions, such as **item**, that require a valid field name as a parameter.

**Example**

```cpp
CRString address;
if (dataset1.fields.findItem("Address"))
    address = dataset1.fields.findItem("Address")->getString();
```

**See Also**

**CRField**, **CRFields::item**, **CRDataset::findField**

8.5.3 **CRFields::item**

CRField& item(int index) const;
CRField& item(const CRString& name) const;

**Return Value**
A reference to CRField object for the specified field.

**Parameters**

*index*

  Zero-based index number of the field.

*name*

  Name of the field.

**Description**

Call this member function to retrieve field information for a field when its name or its index is known. If the specified field is not found, the function throws CRException exception.

**Example**

```cpp
CRString address;
// get value of the field with the name address
address = dataset1.fields.item("Address").getString();
// set string to the field with index 0
dataset2.fields.item(0).assign(address);
```

**See Also**

CRField, CRFields::findItem, CRFields::operator[], CRDataset::field, CRException

8.5.4 **CRFields::operator []**

```cpp
CRField& operator [](int index) const;
CRField& operator [](const CRString& name) const;
```

**Return Value**

A reference to CRField object for the specified field.

**Parameters**

*index*

  Zero-based index number of the field.

*name*

  Name of the field.

**Description**

These operators allow to retrieve field information for a field when its name or its index is known. If the specified field is not found, they throw CRException exception.
Example

CRString address;
// get value of the field with the name address
address = dataset1.fields["Address"].getString();
// set string to the field with index 0
dataset2.fields[0].assign(address);

See Also

CRField, CRFields::item, CRDataset::operator[], CRException

CRFields | Classes | OCL | Index
# CRLob

```cpp
#include "dataset.h"
```

## Members

### Attributes

- `long size() const;`
- `char* getChars(char* result) const;`
- `void setChars(const char* value);`
- `CRString getString() const;`
- `void setString(const CRString& value);`

## Methods

- `void assign(CRLob& source);`
- `void clear();`
- `void compress();`
- `long read(long position, long count, void* dest);`
- `void write(long position, long count, void* source);`

## Description

CRLob object represents the large objects value. CRLob is used by CRField and OraParam. To get reference to CRLob use its `getBlob` and `getMemo` member functions. Use `size` member function to know current size of value. With `read` and `write` functions it is possible to get or set content of blob. Another way is using `getString` and `setString` to manipulate blob value as string.

## See Also

- `CRField::getLob`, `OraParam::getLob`

## Classes | OCL | Index

### 8.6.1 CRLob::assign

```cpp
void assign(CRLob& source);
```

## Parameters

### Source

The reference to CRLob object that will be assigned.
**Description**

Call this member function to assign the value from another CRLob object.

**Example**

```cpp
CRLob blob1;
CRLob blob2;
...;
blob2.assign(blob1);
```

**See Also**

[CRLob::write](#), [CRLob::setString](#)

### 8.6.2 CRLob::clear

```cpp
void clear();
```

**Description**

Call this member function to delete the current value in blob object.

**See Also**

[CRLob::size](#)

### 8.6.3 CRLob::compress

```cpp
void compress();
```

**Description**

Call this member function to compress blob value. After multiple rewriting blob value by pieces it can be allocated in memory not effective. This function tries to allocate blob value more effective if possible.

**See Also**

[CRLob::write](#)
8.6.4 CRLob::getWChars

```
wchar_t* getWChars(wchar_t* result);
```

**Return Value**

Pointer to character buffer.

**Parameters**

- `result`  
  Pointer to character buffer.

**Description**

Call this member function to retrieve the CLOB value represented as a Unicode string.

**See Also**

- `CRLob::setWChars`, `CRLob::read`

---

8.6.5 CRLob::getChars

```
char* getChars(char* result);
```

**Return Value**

Pointer to character buffer.

**Parameters**

- `result`  
  Pointer to character buffer.

**Description**

Call this member function to retrieve the blob value represented as a string.

**See Also**

- `CRLob::setChars`, `CRLob::read`
8.6.6  CRLoob::getWString

CR String getWString() const;

**Return Value**
Unicode string representation for the Clob object.

**Description**
Call this member function to retrieve the Clob value represented as Unicode string.

**Example**
```
CRWString st;
st = dataset.field("Description").getLob().getWString();
```

**See Also**
CRLoob::setWString, CRLoob::read

CRLoob | Classes | OCL | Index

8.6.7  CRLoob::getString

CRString getString() const;

**Return Value**
String representation for the Blob object.

**Description**
Call this member function to retrieve the blob value represented as string.

**Example**
```
CRString st;
st = dataset.field("Description").getLob().getString();
```

**See Also**
CRLoob::setString, CRLoob::read

CRLoob | Classes | OCL | Index
8.6.8 CRLob::loadFromFile

void loadFromFile(const CRString& fileName);

Parameters

fileName
File name string.

Description
Call this member function to assign the blob value with the data from file.

Example

CRString st = "long.txt";
.
.
dataset.field("Description").getLob().loadFromFile(st);

See Also
CRLob::saveToFile, CRLob::write

8.6.9 CRLob::read

long read(long position, long count, void* dest);

Return Value
The number of bytes actually read.

Parameters

position
The position to start reading from.

count
The count of bytes that will be read.

dest
The pointer to buffer where data will be read.

Description
Call this member function to read count bytes of blob value starting from the specified position.
Example

```c
char* desc;
long size = dataset.field("Description").getLob().size();

desc = new char[size];
dataset.field("Description").getLob().read(0, size, desc);
```

See Also

CRLob::write, CRLob::size, CRLob::getString

8.6.10 CRLob::saveToFile

```c
void saveToFile(const CRString& value);
```

Parameters

value
File name string.

Description
Call this member function to save the blob value to the file.

Example

```c
CRString st = "long.txt";
.
.
dataset.field("Description").getBlob().loadFromFile(st);
```

See Also

CRLob::loadFromFile, CRLob::write

8.6.11 CRLob::setWChars

```c
void setWChars(const wchar_t* value);
```

Parameters

value
The pointer to the Unicode string.
8.6.12 CRLob::setChars

void setChars(const char* value);

**Parameters**

*value*

The pointer to the string of chars.

**Description**

Call this member function to copy the string of chars to this blob object.

**See Also**

CRLob::getChars, CRLob::write

CRLob | Classes | OCL | Index

8.6.13 CRLob::setWString

void setWString(const CRWString& value);

**Parameters**

*value*

The reference to the CRWString object.

**Description**

Call this member function to assign the Unicode string value to the Clob.

**Example**

```cpp
CRWString st;
...
dataset.field("Description").getLob().setWString(st);
```
See Also

CRLob::getString, CRLob::write

CRLob | Classes | OCL | Index

8.6.14 CRLob::setString

void setString(const CRString& value);

Parameters

value
The reference to the CRString object.

Description

Call this member function to assign the string value to the blob.

Example

```cpp
CRString st;
... 
dataset.field("Description").getLob().setString(st);
```

See Also

CRLob::getString, CRLob::write

CRLob | Classes | OCL | Index

8.6.15 CRLob::size

long size() const;

Return Value

The size of blob value.

Description

Call this member function to determinate the size of blob value.

See Also

CRLob::clear

CRLob | Classes | OCL | Index
8.6.16 CRLob::write

void write(long position, long count, void* source);

**Parameters**

*position*
   The position to start writing from.

*count*
   The count of bytes that will be written.

*source*
   The pointer to the buffer with the data.

**Description**

Call this member function to write count bytes to blob value from the specified position.

**Example**

```c++
char* src;
int size;
...
src = new char[size];
...
dataset.field("Description").getBlob().write(0, size, src);
```

**See Also**

CRLob::read, CRLob::clear, CRLob::setString, CRLob::assign

CRLob | Classes | OCL | Index
8.7 CRMemDataset

#include "dataset.h"

Hierarchy
CRDataset
    CRMemDataset

Members

Construct on

CRMemDataset();
virtual ~CRMemDataset();

Description
CRMemDataset is derived from CRDataset class and implements storing records in memory.

See Also
CRDataset, OraRecordset

Classes | OCL | Index

8.7.1 CRMemDataset::CRMemDataset

CRMemDataset();

Description
This member function constructs a new CRMemDataset object.

See Also
CRDataset::open

CRMemDataset | Classes | OCL | Index
8.8 CRSharedObject

#include "crdb.h"

Members

Construct on

CRSharedObject();

Methods

int addRef();
void checkValid();
int refCount();
void release();

Description

CRSharedObject is a base class that handles reference count for its derived classes.

When external object assigns pointer to CRSharedObject instance it calls addRef
member function. When external object no longer needs reference to CRSharedObject
instance then it calls release method for that instance. At construction reference count is
assigned 1. When reference count reaches zero CRSharedObject instance is destroyed.

See Also

OraCursor

8.8.1 CRSharedObject::addRef

int addRef();

Return Value

Current value of the reference count.

Description

This member function increments and then returns reference count for this object.

See Also

release
8.8.2 CRSharedObject::checkValid

void checkValid();

**Description**
Verifies that the reference count for this object is nonzero. If the reference count is zero then this method throws CRException exception.

**See Also**
refCount

---

8.8.3 CRSharedObject::CRSharedObject

CRSharedObject();

**Description**
This member function constructs a new CRSharedObject object and assigns 1 to its reference count.

**See Also**
addRef, release

---

8.8.4 CRSharedObject::refCount

int refCount();

**Return Value**
Current value of the reference count.

**Description**
This member function returns current value of the reference count for this object.
See Also

addRef, release

CRSharedObject | Classes | OCL | Index

8.8.5 **CRSharedObject::release**

void release();

**Description**

This member function decrements current value of the reference count for this object. If reference count reaches zero then CRSharedObject instance is destroyed and occupied memory is released.

See Also

checkValid, addRef

CRSharedObject | Classes | OCL | Index
8.9 CRWString

```
#include "ocldef.h"
```

**Hierarchy**

```
std::wstring
  CR    String
```

**Description**

CRWString represents Unicode string value and is used by many OCL functions. It is inherited from STL `wstring` class.

**See Also**

- [CRString](#)

---

[Classes] [OCL] [Index]
8.10 CRString

#include "ocldef.h"

Hierarchy

std::string
| CRString

Description

CRString represents string value and is used by many OCL functions. It is inherited from STL string class.

See Also

CRValue, CRDate

Classes | OCL | Index
# CRValue

include "dataset.h"

Members

Construct on

CRValue();
CRValue(int value);
CRValue(float value);
CRValue(double value);
CRValue(const CRString& value);
CRValue(const char* value);
CRValue(const CRDate& value);
CRValue(const CRValue& value);
virtual ~CRValue();

Attributes

ValueType type() const;
bool isNull() const;
void setNull();
int getInt() const;
float getFloat() const;
double getDouble() const;
CRString getString() const;
CRDate getDate() const;
bool operator ==(const CRValue& value);

Assignment

void operator =(int value);
void operator =(float value);
void operator =(double value);
void operator =(const CRString& value);
void operator =(const char* value);
void operator =(const CRDate& value);
void operator =(const CRValue& value);

Description

This class allows to store value of various types. These types are int, float, double,
CRString and CRDate. Type of value can be changed at runtime. Many OCL functions return or accept CRValue object. For example, you can get value of field or parameter as CRValue.
8.11.1 CRValue::CRValue

CRValue();
CRValue(int value);
CRValue(float value);
CRValue(double value);
CRValue(const CRString& value);
CRValue(const char* value);
CRValue(const CRDate& value);
CRValue(const CRValue& value);

Parameters

descrpt on

value
The value that will be assigned to the new object.

Example

    CRValue a(5);
    CRValue b("Hello");

See Also

CRValue::setValue

Copyright © 2009 Devart
Call this member function to retrieve the value represented as CRDate.

See Also

CRDate, CRValue::operator =

CRValue | Classes | OCL | Index

8.11.3 CRValue::getDouble

double getDouble() const;

Return Value

Double floating point number.

Description

Call this member function to retrieve the value represented as double.

See Also

CRValue::operator =

CRValue | Classes | OCL | Index

8.11.4 CRValue::getFloat

float getFloat() const;

Return Value

Floating point number.

Description

Call this member function to retrieve the value represented as float.

See Also

CRValue::operator =

CRValue | Classes | OCL | Index
8.11.5 **CRValue::getInt**

    int getInt() const;

**Return Value**

Integer number.

**Description**

Call this member function to retrieve the value represented as int.

**See Also**

CRValue::operator =

 CRValue | Classes | OCL | Index

8.11.6 **CRValue::getString**

    CRString getString() const;

**Return Value**

CRString object.

**Description**

Call this member function to retrieve the value represented as string.

**See Also**

CRString, CRValue::operator =

 CRValue | Classes | OCL | Index

8.11.7 **CRValue::isNull**

    bool isNull() const;

**Return Value**

True if value is NULL, otherwise false.
Description

Call this member function to determinate whether the value for this object is NULL or not.

See Also

CRValue::setNull

8.11.8 CRValue::operator =

CRValue& operator =(int value);
CRValue& operator =(float value);
CRValue& operator =(double value);
CRValue& operator =(const CRString& value);
CRValue& operator =(const char* value);
CRValue& operator =(const CRDate& value);
CRValue& operator =(const CRValue& value);

Return Value

This CRValue object.

Parameters

value

New value for this CRValue object.

Description

Call this member function to assign the value to this CRValue object.

See Also

CRValue::setNull

8.11.9 CRValue::operator ==

bool operator == (const CRValue& value);

Return Value
True if the two objects are equal.

**Parameters**

$value$

The value to be compared with this object.

**Description**

This operator compares this CRValue object and the $value$ parameter object. Returns true if they are equal.

8.11.10 **CRValue::setNull**

void setNull();

**Description**

Call this member function to assign a NULL value to CRValue object.

**See Also**

CRValue::isNull, CRValue::operator =

8.11.11 **CRValue::type**

enum ValueType {
    None,
    Int,
    Float,
    Double,
    String,
    Date
};

ValueType type() const;

**Return Value**

Type of the value.

**Description**
Call this member function to verify the type of the value associated with this CRValue object.

**See Also**

CRValue::isNull

CRValue | Classes | OCL | Index
# include "oraloader.h"

Members

Construct on

LoaderColumn();
LoaderColumn(const CRString& name, OraDataType dataType,
    int size, int precision, const CRString& dateFormat);

Attributes

CRString name() const;
void setName(const CRString& name);

OraDataType dataType() const;
void setDataType(OraDataType dataType);

int size() const;
void setSize(int size);

int precision() const;
void setPrecision(int precision);

int scale() const;
void setScale(int scale);

CRString dateFormat() const;
void setDateFormat(const CRString& dateFormat);

Description

Each OraLoader uses LoaderColumn class to maintain a collection of LoaderColumn objects. LoaderColumn object represents the attributes for table column loading. Every LoaderColumn object corresponds to one of the table fields with the same name as its name property.

See Also

OraLoader

Classes  |  OCL  |  Index
8.12.1 **LoaderColumn::dataType**

```
OraDataType dataType() const;
```

**Return Value**

Data type of the column.

**Description**

Use this method to determine Oracle data type of the column.

**See Also**

LoaderColumn::setDataTypeloaderColumn

LoaderColumn | Classes | OCL | Index

8.12.2 **LoaderColumn::dateFormat**

```
CRString dateFormat() const;
```

**Return Value**

Date format into which loaded data will be converted.

**Description**

Use this method to determine date format into which loaded data will be converted.

**See Also**

LoaderColumn::setDateFormat

LoaderColumn | Classes | OCL | Index

8.12.3 **LoaderColumn::LoaderColumn**

```
LoaderColumn();
LoaderColumn(const CRString& name, OraDataType dataType,
             int size, int precision, const CRString& dateFormat);
```

**Parameters**
name
  Name of the column.

dataType
  Oracle data type of the column.

size
  Maximum size of the column data (only for strings).

precision
  Precision of the column.

dateFormat
  Date format into which loaded data will be converted.

**Description**

Initializes a new instance of the LoaderColumn class with the column name, column data type, column size, precision, and date format.

**Return Value**
CRString name() const;

**Description**

Every LoaderColumn object corresponds to one of the table fields with the same name as its Name property. Use this member function to return the name of the column.

**See Also**

LoaderColumn::setName

**8.12.4 LoaderColumn::name**

int precision() const;

**Return Value**
The maximum number of digits used to represent the column value.

**Description**

Use this member function to determine precision of number columns. The default value is 0.

**See Also**

- [LoaderColumn::setPrecision](#)
- [LoaderColumn](#)

### 8.12.6 LoaderColumn::scale

```cpp
int scale() const;
```

**Return Value**

The number of decimal places to which column value is resolved.

**Description**

Use this member function to determine scale of number columns. The default value is 0.

**See Also**

- [LoaderColumn::setScale](#)
- [LoaderColumn](#)

### 8.12.7 LoaderColumn::setDataType

```cpp
void setDataType(OraDataType dataType);
```

**Parameters**

dataType

- Oracle data type for the column.

**Description**

Use this method to specify Oracle data type for the column.

**See Also**

- [LoaderColumn::setDataType](#)
- [LoaderColumn](#)
8.12.8 `LoaderColumn::setDateFormat`

```cpp
void setDateFormat(const CRString& dateFormat);
```

**Parameters**

`dateFormat`

Date format into which loaded data will be converted.

**Description**

Set date format to specify conversion mask for column. OraLoader uses `DateFormat` to convert string representation of date to its internal representation. If not set, the date format defaults to the date conversion mask set in the direct path context.

**See Also**

`LoaderColumn::dateFormat`

8.12.9 `LoaderColumn::setName`

```cpp
void setName(const CRString& name);
```

**Parameters**

`name`

The name of the `LoaderColumn`.

**Description**

Every `LoaderColumn` object corresponds to one of the table fields with the same name as its `Name` property. Use this member function to specify the name of column.

**See Also**

`LoaderColumn::name`
8.12.10  LoaderColumn::setPrecision

void setPrecision(int precision);

Parameters

\textit{precision}  \\
\hspace{1cm} The maximum number of digits used to represent the column value.

Description

Use this member function to specify precision of number columns.

See Also

LoaderColumn::precision
LoaderColumn

8.12.11  LoaderColumn::setScale

void setScale(int scale);

Parameters

\textit{scale}  \\
\hspace{1cm} The number of decimal places to which column value is resolved.

Description

Use this member function to set scale of number columns. The default value is 0.

See Also

LoaderColumn::scale
LoaderColumn

8.12.12  LoaderColumn::setSize

void setSize(int size);

Parameters

\textit{size}
The maximum size, in bytes, of the data within the column.

**Description**

Use this method to set maximum size in bytes of data for the column. Size is used only for string columns.

**See Also**

`LoaderColumn::size`

**8.12.13 LoaderColumn::size**

```cpp
int size() const;
```

**Return Value**

The maximum size, in bytes, of the data within the column.

**Description**

Use this method to get maximum size in bytes of data for the column. Size is used only for string columns.

**See Also**

`LoaderColumn::setSize`
8.13 OraCommand

#include "ora.h"

Members

Construct on

OraCommand();
OraCommand(OraConnection& connection);
OraCommand(OraConnection& connection, const CRString& sql);
~OraCommand();

Preparation

void prepare();
void unprepare();
bool isPrepared() const;

Executing

void execute(int iters = 1);
void execute(OraRecordset** rs);
OraRecordset& executeQuery();
void cancel();

SQL & PL/SQL

CRString commandText() const;
void setCommandText(const CRString& value);
CRCommandType commandType() const;
void setCommandType(CRCommandType value)
CRString SQL() const;
void setSQL(const CRString& value);
int SQLType() const;
bool isQuery() const;
bool isPLSQL() const;

Stored proc

void createProcCall(const CRString& name);

Params

void bindParams();

int paramCount() const;
OraParams& params();
OraParam& param(int index) const;
OraParam& param(const CRString& name) const;
OraParam* findParam(int index) const;
OraParam* findParam(const CRString& name) const;

Misc

const char* rowId() const;
int rowsProcessed() const;
OraConnection* connection() const;
void setConnection(OraConnection& connection);
void setConnection(OraConnection* connection);
OraCursor* cursor() const;
void setCursor(OraCursor& value);
CRCursorState cursorState() const;
int errorOffset() const;

void setArrayLength(int newLength);

OraParam& operator[](int index) const;
OraParam& operator[](const CRString& name) const;

Description

This class allows to execute any SQL, PL/SQL statements or call stored procedures on the database server. OraCommand supports parameters. Use setCommandText member function to assign SQL, PL/SQL statement text or the name of a table or stored procedures. Then call set.CommandType method to indicate the type of command text, param to access to parameter objects and execute member function to execute the statement.

Example

OraConnection connection("scott/tiger");
OraCommand cmd(connection);
int count;

connection.connect();
cmd.setCommandText("begin SELECT Count(*) INTO :Count FROM DeptNo;
end;");
cmd["Count"].setDataType(dtInt);
cmd.execute();
count = cmd["Count"].getInt();

See Also

OraConnection, OraRecordset, OraCommand::execute

Classes | OCL | Index
8.13.1 OraCommand::bindParams

void bindParams();

**Description**
Binds all parameters which are found in the SQL statement. This function is called by execute function.

**See Also**
OraCommand::execute
OraCommand | Classes | OCL | Index

8.13.2 OraCommand::cancel

void cancel();

**Description**
Call this member function to break execution of the SQL statement on the server.

**See Also**
OraCommand::execute
OraCommand | Classes | OCL | Index

8.13.3 OraCommand::commandText

CRString commandText() const;

**Return Value**
String with the command text.

**Description**
Returns the command statement set previously by setCommandText member function.

**See Also**
OraCommand::setCommandText, OraCommand::commandType
8.13.4  OraCommand::commandType

enum CRCommandType {ctSQL, ctTable, ctStoredProc};

CRCommandType commandType() const;

**Return Value**
Type of the command statement. See `setCommandType` for the description of each command type. Default value is ctSQL.

**Description**
Call this member function to check the type of command text statement.

**See Also**
OraCommand::setCommandType, OraCommand::commandText

8.13.5  OraCommand::connection

OraConnection* connection() const;

**Return Value**
Pointer to the connection object.

**Description**
Returns the connection object set previously by `setConnection` member function.

**See Also**
OraConnection, OraCommand::setConnection

8.13.6  OraCommand::createProcCall

void createProcCall(const char* name);
Parameters

name
The name of the stored procedure or function. Write PackageName.ProcedureName if the procedure is inside of the package.

Description
Call this member function to build PL/SQL block that calls stored procedure specified by name. After calling createProcCall you can execute stored procedure by execute function. Retrieves the information about parameters of the procedure from Oracle.

Example

OraCommand SQL(connection);
SQL.createProc("Dept_Insert");
SQL.param("DeptNo") = 60;
SQL.param("DName") = "Research";
SQL.execute();

See Also
OraCommand::execute, OraCommand::SQL, Working with stored procedures

8.13.7 OraCommand::cursor

OraCursor* cursor() const;

Return Value
Pointer to OraCursor object.

Description
Call this member function to access to internal OraCursor object.

See Also
OraCommand::setCursor

8.13.8 OraCommand::cursorState

CRCursorState cursorState() const;
**Return Value**

State of the cursor. Can be one of the following values: `csInactive`, `csOpen`, `csParsed`, `csPrepared`, `csBound`, `csExecuting`, `csExecuted`, `csFetching`, `csFetchingAll`, `csCanceled`.

**Description**

Call this member function to know what the state of cursor associated with `OraCommand` object is.

**Example**

```cpp
if (cmd.cursorState() < csPrepared)
    cmd.prepare();
```

**See Also**

`OraCursor::state`, `OraCursor::canFetch`

**8.13.9 OraCommand::errorOffset**

```cpp
int errorOffset() const;
```

**Return Value**

Error symbol position.

**Description**

Use this member function to determine symbol position in SQL statement that causes an error.

**See Also**

`OraCommand::execute`

**8.13.10 OraCommand::execute**

```cpp
int execute(int iters = 1);
int execute(OraRecordset** rs, bool fieldsInfoOnly = false);
```
Return Value

The actual number of rows processed. In case of SELECT statement this number equals to the number of fetched rows.

Parameters

ters
  Specifies a number of times to execute SQL statement on the server.

rs
  Pointer to a pointer which will point to the OraRecordset object after the command has been executed.

fieldsInfoOnly
  If true, OraCommand dose not execute statement, insted it obtain fields metadata.

Description

Call this member function to execute the SQL statement on the server. This function prepares SQL statement if prepare function wasn't called directly before. Use param function to access individual parameter objects to write and read their values. To avoid unnecessary round trips when processing a number of similar commands on the server you may wish to consider using DML arrays. In this case iters parameter is used to specify a number of times that this command is executed by the server in a single step. If the SQL statement returns recordset then pass rs parameter to get reference to OraRecordSet class instance that holds execution results.

Note  OraRecordSet class instance is managed by the OraCommand object. Use rs parameter to access to returned recordsets only.

Example

```cpp
OraConnection connection("scott/tiger");
OraCommand cmd(connection);
int count;

connection.connect();
cmd.commandText("begin SELECT Count(*) INTO :Count FROM DeptNo; end");
cmd.param("Count").setDataType(dtInt);
cmd.execute();
count = cmd.param("Count").getInt();
```

See Also

OraCommand::setConnection, OraCommand::commandText, OraCommand::prepare, OraCommand::param, OraCommand::executeQuery, COraRecordSet
8.13.11 OraCommand::executeQuery

OraRecordset& executeQuery();

**Return Value**

Instance of the OraRecordset object holding recordset returned after executing of the query.

**Description**

Call this member function to execute the SQL statement that returns recordset. This function prepares SQL statement if `prepare` function wasn't called before. Use `param` function to access individual parameter objects to write and read their values. If `executeQuery` is called for the statement which doesn't return recordset then an exception is thrown.

**Note** OraRecordSet class instance is owned by the COraRecordSet object.

**Example**

```cpp
OraConnection connection("scott/tiger");
OraCommand cmd(connection);
int count;

connection.open();
cmd.setCommandText("begin SELECT Count(*) INTO :Count FROM DeptNo; end");
cmd.param("Count").setDataType(dtInt);
OraRecordset &rs = cmd.executeQuery();
count = cmd.param("Count").getInt();
```

**See Also**

OraCommand::setConnection, OraCommand::commandText, OraCommand::prepare, OraCommand::param, OraCommand::execute, COraRecordSet

8.13.12 OraCommand::findParam

OraParam* findParam(int index) const;
OraParam* findParam(const CRString& name) const;

**Return Value**

The pointer to OraParam object for the specified parameter.

**Parameters**
index
Zero-based index number of the parameter.

name
Name of the parameter.

**Description**
Call this member function to determine if a specified parameter is referenced in the OraCommand object. If `findParam` finds a parameter with a matching name, it returns the OraParam object for the specified parameter. Otherwise it returns NULL. `findItem` is a useful function to call prior to calling other OraCommand functions, such as `param`, that require a valid parameter name as a parameter name.

**Example**
```c++
CRString address;
OraParam* par = cmd.findParam("Address");
if (par)
    address = par->getString();
```

**See Also**
OraParam, OraCommand::param, OraParams::findItem

---

**8.13.13 OraCommand::isPLSQL**

bool isPLSQL() const;

**Return Value**
True if SQL statement is PL/SQL block, otherwise false.

**Description**
Call this member function to check whether SQL statement is PL/SQL block. To get correct value OraCommand object must be prepared before.

**See Also**
OraCommand::prepare, OraCommand::SQLType, OraCommand::isQuery
8.13.14 OraCommand::isPrepared

bool isPrepared() const;

Return Value

True if the OraCommand object's `prepare` member function has previously been called and the OraCommand has not been unprepared, otherwise false.

Description

Call this member function to determine if the OraCommand is already prepared.

See Also

OraCommand::prepare, OraCommand::unprepare, OraCommand::execute

OraCommand | Classes | OCL | Index

8.13.15 OraCommand::isQuery

bool isQuery() const;

Return Value

True if SQL statement is SELECT or PL/SQL block that returns REF CURSOR parameter, otherwise false.

Description

Call this member function to check whether SQL statement returns rows or not. SQL statement returns rows when it is SELECT or PL/SQL block with REF CURSOR parameter. To get correct value OraCommand object must be prepared before.

See Also

OraCommand::prepare, OraCommand::SQLType, OraCommand::isPLSQL

OraCommand | Classes | OCL | Index

8.13.16 OraCommand::isRowsReturn

bool isRowsReturn();

Return Value
Nonzero if the SQL query returns result set; zero otherwise.

**Desc r i pt o n**

Call this member function after `prepare` method to verify whether SQL statement returns record set or not. If SQL query is a SELECT statement or returns REF CURSOR as OUT parameter then this function will evaluate to a nonzero value.

**See Also**

`OraCommand::prepare`, `OraCommand::execute`, Proc demo project

`OraCommand` | Classes | OCL | Index

### 8.13.17 `OraCommand::operator []`

```
OraParam& operator [](int index) const;
OraParam& operator [](const CRString& name) const;
```

**Return Value**

The reference to OraParam object for the specified parameter.

**Parameters**

- **index**
  
  Zero-based index number of the parameter.

- **name**
  
  Name of the parameter.

**Desc r i pt o n**

These operators allow to retrieve parameter information for a parameter when its name or its index is known. If the specified parameter is not found they throw the exception.

**Example**

```cpp
CRString address;
// get value of the parameter with the name address
address = cmd1["Address"].getString();
// set string to the parameter with index 0
cmd2[0].assign(address);
```

**See Also**

`OraParam`, `OraCommand::param`, `OraCommand::params`, `OraParams::operator[]`

`OraCommand` | Classes | OCL | Index
8.13.18 OraCommand::OraCommand

OraCommand();
OraCommand(OraConnection& connection);
OraCommand(OraConnection& connection, const CRString& sql);

Parameters

connection
The reference to the connection object in which SQL will be executed.

sql
The SQL statement.

Description
This member function constructs a new OraCommand object.

See Also
OraCommand::execute, OraCommand::prepare

8.13.19 OraCommand::param

OraParam& param(int index) const;
OraParam& param(const CRString& name) const;

Return Value
A reference to the OraParam object for the specified parameter.

Parameters

index
Zero-based index number of the parameter.

name
Name of the parameter.

Description
Call this member function to retrieve parameter information for a parameter when its name or its index is known. If the specified parameter is not found param function throws the exception.

Example
CRString address;  // get value of the parameter with the name address  
address = cmd1.param("Address").getString();  
// set string to the parameter with index 0  
cmd2.param(0).assign(address);

See Also

OraParam, OraCommand::findParam, OraCommand::operator[], OraParams::item

8.13.20 OraCommand::paramCount

int paramCount() const;

Return Value

The number of parameters in the OraCommand.

Description

Call this member function to determine the number of parameters managed by the OraCommand object.

See Also

OraCommand::params

8.13.21 OraCommand::params

OraParams& params();

Return Value

The reference to OraParams object associated with the OraCommand.

Description

Call this member function to retrieve information for a parameter when its index is known.

Example

CRString name;  // get value of the parameter with the name DName
name = cmd1.params["DName"].getString();
// set string to the parameter with index 0
cmd2.params.item(0).assign(name);

See Also

OraParams, OraCommand::param

OraCommand | Classes | OCL | Index

8.13.22 OraCommand::prepare

void prepare();

Description

Calling prepare before executing a SQL statement improves application performance. OraCommand automatically prepares a SQL statement if it is executed without first being prepared. After execution, OraCommand unprepares the SQL statement. Then a query is executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares. To learn whether the OraCommand is prepared or not use sPrepared member function. The unprepare method unprepares a query.

Example

    cmd.setSQL("INSERT INTO Dept(DeptNo) VALUES (:DeptNo)");
    cmd.prepare();
    for (int i = 60; i < 70; i++) {
        cmd["DeptNo"] = i;
        cmd.execute();
    }
    cmd.unprepare();

See Also

OraCommand::isPrepared, OraCommand::unprepare, OraCommand::execute

OraCommand | Classes | OCL | Index

8.13.23 OraCommand::rowid

const char* rowId() const;

Return Value

Rowid of the last processed record.

Description
Call this member function to know rowid of the last processed record immediately after `execute` member function being called. Useful for SELECT, UPDATE and DELETE statements.

**See Also**

`OraCommand::execute`, `OraCommand::rowsPrecessed`

---

### 8.13.24 `OraCommand::rowsProcessed`

```
int rowsProcessed() const;
```

**Return Value**

The number of rows which were processed by SQL statement.

**Description**

Call this member function to know the number of rows processed by the SQL statement immediately after `execute` member function being called. Useful for SELECT, UPDATE and DELETE statements.

**See Also**

`OraCommand::execute`, `OraCommand::rowId`

---

### 8.13.25 `OraCommand::setArrayLength`

```
void setArrayLength(int newLength)
```

**Parameters**

`newLength`

New size for the parameter arrays.

**Description**

With DML array execution you should provide all parameters as array with the same length. You can do it by calling `setTableLength` function for each parameter. In another way, call `setArrayLength` function to specify a new length for all parameters at the same time. Later it is posible to access items of parameter array by index function of `OraParam` class such as `setInt`, `getString` and etc.
Example

    . . .
    
    cmd.setArrayLength(arraySize);
    
    for (int i = 1; i <= arraySize; i++)
        cmd.param("RowNum").setInt(i, i);
    
    cmd.execute(arraySize);

See Also

OraParam::setTableLength, OraCommand::param, OraCommand::execute

OraCommand | Classes | OCL | Index

8.13.26 OraCommand::setCommandText

void setCommandText(const CRString& value)

Parameters

value
    SQL statement, table name or name of the stored procedure.

Description

Call this member function to specify either SQL statement, name of the table or stored procedure that OraCommand object executes or obtains data from when execute method is called. Type of the statement must be provided with the setCommandType method.

See Also

OraCommand::setCommandType

OraCommand | Classes | OCL | Index

8.13.27 OraCommand::setCommandType

enum CRCommandType {ctSQL, ctTable, ctStoredProc};

void setCommandType(CRCommandType value)

Parameters

value
    Type of the command statement. Must be one of the following values:
### Value Description

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctSQL</td>
<td>Specifies that the command text is an SQL statement.</td>
</tr>
<tr>
<td>ctTable</td>
<td>Specifies that the command text is the name of the table. OraCommand will provide necessary SELECT statement wrapper which after execution will return recordset for this table.</td>
</tr>
<tr>
<td>ctStoredProc</td>
<td>Specifies that the command text is the name of the stored procedure. OraCommand will provide necessary PL/SQL wrapper before execution.</td>
</tr>
</tbody>
</table>

**Descr pt on**

Call this member function to set the type of command text statement before execution.

**See Also**

*OraCommand::setCommandText*

### 8.13.28 OraCommand::setConnection

```cpp
void setConnection(OraConnection& connection);
void setConnection(OraConnection* connection);
```

**Parameters**

- `connection` Reference or pointer to the OraConnection instance.

**Descr pt on**

Call this member function to specify the connection in which SQL will be executed.

**Example**

```cpp
OraConnection connection("scott/tiger");
OraCommand cmd;

cmd.setConnection(connection);
cmd.setSQL("DELETE FROM Dept");
cmd.execute();
```

**See Also**

*OraConnection, OraCommand::connection, OraCommand::execute*
8.13.29 OraCommand::setCursor

void setCursor(OraCursor& value);

Parameters

value
   The reference to OraCursor object.

Description

Call this member function to fetch data from cursor parameter. You can pass value returned by getCursor method.

Example

OraCommand cmd;
OraQuery query;
   . . .
cmd.setCommandText(cursorSQL);
cmd.param("Cur").setDataType(dtCursor);
cmd.execute();
query.setCursor(sql.param("Cur").getCursor());
   . . .
query.close();

See Also

OraCommand::cursor, OraParam::getCursor

OraCommand | Classes | OCL | Index

8.13.30 OraCommand::setSQL

void setSQL(const CRString& value);

Parameters

value
   SQL statement.

Description

Call this member function to specify the SQL statement that OraCommand object executes when execute method is called.

Example

OraConnection connection("scott/tiger");
OraCommand cmd;

cmd.setConnection(connection);
cmd.setSQL("DELETE FROM Dept");
cmd.execute();

See Also
OraCommand::SQL, OraCommand::execute

8.13.31 OraCommand::SQL

CRString SQL() const;

Return Value
SQL statement.

Description
Returns the SQL statement set previously by setSQL member function.

See Also
OraCommand::setSQL

8.13.32 OraCommand::SQLType

int SQLType() const;

Return Value
The type code for SQL statement.

Description
Call this member function to check type of SQL statement. To get correct value
OraCommand object must be prepared before. See Oracle documentation for the full list
of type codes.

Example

    cmd.prepare();
    if (cmd.SQLType() != SQL_SELECT)
        {...}; // SQL isn't SELECT
8.13.33 **OraCommand::unprepare**

```cpp
void unprepare();
```

**Description**

Call this member function to free the resources allocated for a previously prepared SQL statement on the server and client sides.

**See Also**

*OraCommand::prepare, OraCommand::isQuery, OraCommand::isPLSQL*

*OraCommand | Classes | OCL | Index*
8.14 OraConnection

#include "ora.h"

Members

Construct on

OraConnection();
OraConnection(const CRString& connectString);
OraConnection(const CRString& username, const CRString& password, const CRString& server);
~OraConnection();

Attributes

const CRString username();
void setUsername(const CRString& value);
const CRString password();
void setPassword(const CRString& value);
const CRString server();
void setServer(const CRString& value);
void setConnectString(const CRString& value);
LDA* getLDA();
void setLDA(LDA* value);
OCISvcCtx* getOCISvcCtx();
void setOCISvcCtx(OCISvcCtx* value);

CRConnectMode connectMode() const;
void setConnectMode(CRConnectMode connectMode);

CRIssolationLevel* isolationLevel() const;
void setIsolationLevel(CRIssolationLevel value);
OCIVersion OCICallStyle() const;
void setOCICallStyle(OCIVersion value);
OraCommand createCommand();

static CRString home();
static CRString homePath();
static void setHome(int homeIndex);
static void setHome(const CRString& homeName);

Open / close

void open(const CRString& connectString = "");
void close();
bool isConnected();
**Transaction control**

```cpp
void startTransaction();
void startTransaction(CRIsolationLevel level);
void commit();
void rollback();
void savepoint(const CRString& savepoint);
void rollbackToSavepoint(const CRString& savepoint);
bool inTransaction();
bool isAutoCommit();
void setAutoCommit(bool value);
```

```cpp
OraTransaction* transaction() const;
void setTransaction(OraTransaction* trans);
```

```cpp
CRString internalName() const;
void setInternalName(const CRString& internalName);
```

```cpp
CRString externalName() const;
void setExternalName(const CRString& externalName);
```

```cpp
void breakExec();
void changePassword(const CRString& newPassword);
static int getTNSAliases(CRString aliases[], int arrCount);
static void setDefaultHome();
CRString serverVersion();
```

**Description**

This class is used to establish and control connection to the Oracle database. After setting username, password and server, you can open connection to the database by calling `open` member function. Furthermore, you control transactions at the connection level. You can end current transaction with `commit` or `rollback` function.

All the classes that access to data, such as `OraRecordset`, `OraCommand` use `OraConnection`. They have own `setConnection` member functions to set relation with `OraConnection` object.

**Example**

```cpp
OraConnection connection();

connection.setUsername("scott");
connection.setPassword("tiger");
connection.open();
...
connection.close();
```

**See Also**

`OraRecordset`, `OraCommand`, `OraConnection::connect`, `Working with connection`
8.14.1 OraConnection::breakExec

void breakExec();

Description
Call this member function to break execution of the current SQL statement on the server.

See Also
OraCommand
OraConnection

8.14.2 OraConnection::changePassword

void changePassword(const CRString& newPassword);

Parameters
newPassword
New password for the current user.

Description
Call this member function to replace current user password for this session to a new one. Previous values for password and userName properties must already be provided prior to calling changePassword. ChangePassword method is used mainly when login to the user account fails due to the password expiration or any other relevant reason accompanied by an exception with ORA-2800 Oracle error code family (see Oracle Error Messages).

See Also
OraConnection::open
8.14.3 OraConnection::close

```cpp
void close();
```

**Description**

Call this member function to disconnect from Oracle server. Before disconnection close all opened datasets associated with the OraConnection object.

**See Also**

OraConnection::open

OraConnection | Classes | OCL | Index

8.14.4 OraConnection::commit

```cpp
void commit();
```

**Description**

Call this member function to store to the Oracle server all updates of data associated with the current transaction and then end the transaction. The current transaction is the last transaction started by previous call to the **commit** function or calling startTransaction function.

**See Also**

OraConnection::rollback, OraConnection::startTransaction

OraConnection | Classes | OCL | Index

8.14.5 OraConnection::connectMode

```cpp
CRConnectMode connectMode() const;
```

**Return Value**

Connection mode to Oracle database.

**Description**

Use this member function to determine which system privileges user has obtained when connected to Oracle server.
8.14.6 OraConnection::createCommand

OraCommand* createCommand();

Return Value

Pointer to the newly created OraCommand instance.

Description

Call this member function to create a new instance of the OraCommand class and associate it with this connection object.

See Also

OraCommand

OraConnection | Classes | OCL | Index

8.14.7 OraConnection::externalName

CRString externalName() const;

Return Value

The external name is the user-friendly global database name.

Description

The external name is the user-friendly global name stored in sys.props$.value$, where name = 'GLOBAL DB NAME'. It is not guaranteed to be unique unless all databases register their names with a network directory service.

See Also

OraConnection::setExternalName

OraConnection | Classes | OCL | Index
8.14.8 *OraConnection::getLDA*

LDA* getLDA();

**Return Value**

Pointer to the LDA structure.

**Description**

Call getLDA method to get pointer to Oracle7 login data area of the current connection. LDA structure is relevant to OCI7 call interface.

**See Also**

*OraConnection::setLDA*

*OraConnection | Classes | OCL | Index*

8.14.9 *OraConnection::getOCISvcCtx*

OCISvcCtx* getOCISvcCtx();

**Return Value**

Service context handle.

**Description**

Call this member function to return Oracle8 service context handle of the current connection.

**See Also**

*OraConnection::setOCISvcCtx*

*OraConnection | Classes | OCL | Index*

8.14.10 *OraConnection::getTNSAliases*

static int getTNSAliases(CRString aliases , int arrCount);

**Return Value**

Number of returned TNS aliases.
Parameters

aliases
   Array of returned TNS aliases.

arrCount
   Array size.

Description
Use this member function to obtain TNS names which exist in the current Oracle home.

Example
...  
CRString aliases[10];
int aliasCount = OraConnection::getTNSAliases(aliases, 10);
for (int i = 0; i < aliasCount; i++)
   cout << aliases[i] << endl;
...

See Also
OraConnection::open

OraConnection | Classes | OCL | Index

8.14.11 OraConnection::home

static CRString home();

Return Value
Current Oracle home name

Description
Use this method to obtain current Oracle home name that will be used for all OraConnection objects in an application.

See Also
OraConnection::setHome

OraConnection | Classes | OCL | Index
8.14.12 OraConnection::homePath

    static CRString homePath();

**Return Value**

Current Oracle home path.

**Description**

Use this method to obtain current Oracle home path that will be used for all OraConnection objects in an application.

**See Also**

OraConnection::setHome

OraConnection | Classes | OCL | Index

8.14.13 OraConnection::internalName

    CRString(internalName()) const;

**Return Value**

The client database name that will be recorded when performing global transactions.

**Description**

The name can be used by the DBA to track transactions that may be pending in a prepared state due to failures.

**See Also**

OraConnection::setInternalName

OraConnection | Classes | OCL | Index

8.14.14 OraConnection::serverVersion

    CRString(serverVersion());
8.14.15 **OraConnection::inTransaction**

bool inTransaction();

**Return Value**
True if a user transaction is started, otherwise false.

**Description**
Call this member function to determine if a user transaction is currently in progress. Calling **startTransaction** function begins a user transaction and calling one of **commit** or **rollback** functions ends a user transaction.

**Example**
```cpp
connection.startTransaction();
if (connection.inTransaction()) // returns true
c    connection.commit(); // ends current transaction
if (!connection.inTransaction()) // returns false
c    connection.startTransaction(); // begins new transaction
```

**See Also**
- **OraConnection::startTransaction**, **OraConnection::commit**, **OraConnection::rollback**
8.14.16 *OraConnection::isAutoCommit*

bool isAutoCommit();

**Return Value**

True if autocommit is enabled.

**Description**

Call this member function to determine whether all OraRecordset and OraCommand objects, associated with the connection, perform commit after executing SQL. The default value returned by this method is false.

**See Also**

*OraConnection::setAutoCommit*

*OraConnection* | *Classes* | *OCL* | *Index*

8.14.17 *OraConnection::isConnected*

bool isConnected();

**Return Value**

True if connection object is connected, otherwise false.

**Description**

Call this member function to determine if OraConnection object is connected or not.

**See Also**

*OraConnection::connect, OraConnection::close*

*OraConnection* | *Classes* | *OCL* | *Index*

8.14.18 *OraConnection::isolationLevel*

CRIsolationLevel isolationLevel() const;

**Return Value**

Type of the isolation level selected for this transaction. Could be one of the following
values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ilReadCommitted</td>
<td>Specifies that this transaction provides data consistency for single SQL queries only.</td>
</tr>
<tr>
<td>ilReadOnly</td>
<td>Specifies that all queries in this transaction may only read records.</td>
</tr>
<tr>
<td>ilSerializable</td>
<td>Specifies that this transaction operates on one snapshot of data. Modifications done by other transactions either committed or not during the course of this transaction are not seen for it.</td>
</tr>
</tbody>
</table>

**Descr pt on**

Call this member function to retrieve the type of current isolation level previously specified for this transaction. See Oracle documentation on further description of different types of isolation levels.

**Example**

```c
connection.startTransaction(ilReadOnly);

if (connection.isolationLevel() != ilReadOnly) // returns false
    myUpdatingOraQuery.execute(); // modifies database
```

**See Also**

OraConnection::startTransaction

OraConnection | Classes | OCL | Index

8.14.19 **OraConnection::OCICallStyle**

```c
eenum OCIVersion {None,OCI73,OCI80,OCI81,OCI90};

OCIVersion OCICallStyle();
```

**Return Value**

Subgroup of OCI functions that will be used by OraConnection.

**Descr pt on**

Call OCICallStyle method to check what set of OCI routines is used. OraConnection initializes this call style on behalf of default OCI client found in system at the time when OCI library is being loaded.

**See Also**

OraConnection::setOCICallStyle
8.14.20 OraConnection::open

void open(const CRString& conectString = "");

Description

Call this member function to establish a connection to Oracle server. Use setUsername, setPassword, setServer member functions to set login information.

Example

OraConnection connection;

connection setUsername("scott");
connection setPassword("tiger");
connection open();

connection close();

See Also

OraConnection::setUsername, OraConnection::setPassword, OraConnection::setServer, OraConnection::close

8.14.21 OraConnection::OraConnection

OraConnection();
OraConnection(const CRString& connectString);
OraConnection(const CRString& username, const CRString& password, const CRString& server);

Parameters

connectString
A string that contains login information formatted as "username/password@server".

username
A string that contains a name of the user account.

password
A string that contains a password associated with the username.

server
A string that contains a TNS alias name.

**Description**

This member function constructs a new OraConnection object.

**Example**

```cpp
OraConnection connection("scott/tiger");
connection.open();
```

**See Also**

- `OraConnection::setConnectString`
- `OraConnection::open`

---

**8.14.22 OraConnection::password**

```cpp
CRString password();
```

**Return Value**

String with user password.

**Description**

Returns the password set previously by `setPassword` member function.

**See Also**

- `OraConnection::setPassword`
- `OraConnection::username`, `OraConnection::server`

---

**8.14.23 OraConnection::rollback**

```cpp
void rollback();
```

**Description**

Call this member function to cancel all updates for the current transaction and to end the transaction. If user’s transaction is in currently progress ends it.

**See Also**

- `OraConnection::commit`, `OraConnection::startTransaction`
### 8.14.24 OraConnection::rollbackToSavepoint

```cpp
void rollbackToSavepoint(const CRString& savepoint);
```

**Parameters**

**savepoint**
- Indicates the name of the savepoint.

**Description**

Call this member function to cancel all updates for the current transaction to previously identified savepoint.

**See Also**

OraConnection::savepoint, OraConnection::rollback

### 8.14.25 OraConnection::savepoint

```cpp
void savepoint(const CRString& savepoint);
```

**Parameters**

**savepoint**
- Indicates the name of the savepoint.

**Description**

Call this member function to identify a point in the transaction to which you can later rollback. To roll back to set earlier savepoint use **rollbackToSavePoint**.

**See Also**

OraConnection::rollbackToSavepoint
8.14.26  OraConnection::server

const CRString server();

**Return Value**

String with TNS alias.

**Description**

Returns TNS alias name set previously by `setServer` member function.

**See Also**

OraConnection::setServer, OraConnection::username, OraConnection::password

8.14.27  OraConnection::setAutoCommit

void setAutoCommit(bool value);

**Parameters**

*value*

True to perform autocommit.

**Description**

Call this member function to set automatic commit for all updates performed by any SQL in the connection. When value is true and `isAutoCommit` function of corresponding OraRecordset or OraCommand object returns true, each SQL statement executed by this object will be automatically committed.

**See Also**

OraConnection::isAutoCommit, OraConnection::commit

8.14.28  OraConnection::setConnectMode

enum CRConnectMode {cmNormal, cmSysOper, cmSysDBA};

void setConnectMode(CRConnectMode connectMode);
Parameters

connectMode
   Specifies connection mode to Oracle database.

Description

Use this member function to specify which system privileges to use when the user sets the connection to Oracle server. Following values for connection mode are supported:

cmNormal (default)   Connect as an ordinary user.
cmSysOper            Connect with SYSOPER role.
cmSysDBA             Connect with SYSDBA role.

See Also

OraConnection::open, OraConnection::connectMode

OraConnection | Classes | OCL | Index

8.14.29 OraConnection::setConnectString

void setConnectString(const CRString& value);

Parameters

A string contains login information formatted as "username/password@server".

Description

Call this member function to set full login information username, password and TNS alias name (server).

Example

OraConnection connection;
connection.setConnectString("scott/tiger@ora");
connection.open();

See Also

OraConnection::connect, OraConnection::serUsername, OraConnection::setPassword, OraConnection::setServer

OraConnection | Classes | OCL | Index
8.14.30 OraConnection::setDefaultHome

static void setDefaultHome();

Description

Use this method to set default Oracle home as current home from which OCI will be loaded.

OraConnection::setHome

See Also

OraConnection | Classes | OCL | Index

8.14.31 OraConnection::setExternalName

void setExternalName(const CRString& externalName);

Parameters

externalName

The external name is the user-friendly global database name.

Description

The external name is the user-friendly global name stored in sys.props$.value$, where name = 'GLOBAL_DB_NAME'. It is not guaranteed to be unique unless all databases register their names with a network directory service.

See Also

OraConnection::externalName

OraConnection | Classes | OCL | Index

8.14.32 OraConnection::setHome

static void setHome(int homeIndex);
static void setHome(const CRString& homeName);

Parameters

homeIndex
Zero-based Oracle home index.

*homeName*  
Oracle home name.

**Description**

Use this member function to set active Oracle home with specified Oracle home index or name from which OCI library will be loaded.

**Example**

```cpp
OraConnection connection("scott/tiger@ora");
connection.setHome("ORACLE_HOME817");
connection.open();
```

**See Also**

- OraConnection::open

---

**8.14.33 OraConnection::setInternalName**

```cpp
void setInternalName(const CRString& internalName);
```

**Parameters**

*internalName*  
The client database name that will be recorded when performing global transactions.

**Description**

Sets the client database name that will be recorded when performing global transactions. The name can be used by the DBA to track transactions that may be pending in a prepared state due to failures.

**See Also**

- OraConnection::internalName

---

*Disclaimer:* The text above is generated from the original document. It may contain errors, omissions, or inaccuracies. Please refer to the original document or the latest version for the most accurate and up-to-date information.
8.14.34 OraConnection::setIsolationLevel

void setIsolationLevel(CRIsoationLevel level);

Parameters

level
Type of the isolation level to use for subsequent transactions in this session. Must be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ilReadCommitted</td>
<td>Specifies that transactions will provide data consistency for single SQL queries only.</td>
</tr>
<tr>
<td>ilReadOnly</td>
<td>Specifies that all queries of subsequent transactions will only read records.</td>
</tr>
<tr>
<td>ilSerializable</td>
<td>Specifies that subsequent transactions will operate in the context of a single snapshot of data. Modifications done by other transactions either committed or not during the course of these transactions will not be seen for them.</td>
</tr>
</tbody>
</table>

Descr pt on

Call this member function to set the type of the isolation level for all subsequent transactions of this database session. Before modifying isolation level this method will commit any transaction in progress.

See Oracle documentation on further description of different types of isolation levels.

See Also

OraConnection::isolationLevel, OraConnection::commit, OraConnection::rollback, OraConnection::inTransaction

OraConnection | Classes | OCL | Index

8.14.35 OraConnection::setLDA

void setLDA(LDA* value);

Parameters

value
Pointer to the LDA structure.

Descr pt on

Call setLDA method to set pointer to Oracle7 login data area for the current connection. LDA structure is relevant to OCI7 call interface.
See Also

OraConnection::LDA

OraConnection | Classes | OCL | Index

8.14.36 OraConnection::setOCICallStyle

defined enum OCIVersion {None,OCI73,OCI80,OCI81,OCI90};

void setOCICallStyle(OCIVersion value);

Parameters

value
  Type of the OCI call interface.

Description

Call this member function before connection to specify what set of OCI routines to use.

See Also

OraConnection::OCICallStyle

OraConnection | Classes | OCL | Index

8.14.37 OraConnection::setOCISvcCtx

void setOCISvcCtx(OCISvcCtx* value);

Parameters

value
  Pointer to the service context value.

Description

Call setOCISvcCtx to set Oracle8 service context handle for the current connection.

See Also

OraConnection::getOCISvcCtx

OraConnection | Classes | OCL | Index
### 8.14.38 OraConnection::setPassword

void setPassword(const CRString& value);

**Parameters**

- **value**
  
  String with a new password.

**Description**

Call this member function to set password which will be used by OraConnection for connection to Oracle.

**See Also**

- OraConnection::connect, OraConnection::password, OraConnection::setUsername, OraConnection::setServer, OraConnection::setConnectString

### 8.14.39 OraConnection::isUnicode

bool isUnicode();

**Return Value**

True if Unicode is enabled, otherwise false.

**Description**

Call this member function to determine if Unicode is currently enabled.

**See Also**

- OraConnection::setUnicode, OraConnection::charset, OraConnection::setCharset

### 8.14.40 OraConnection::setUnicode

void setUnicode(bool value);

**Parameters**
**value**
True if Unicode is enabled.

**Description**
Enables or disables Unicode support. Affects on character data fetched from the server. When set to True all character data is stored as UTF16 strings. Supported starting since Oracle 8.

**See Also**
OraConnection::isUnicode, OraConnection::charset, OraConnection::setCharset

---

### 8.14.41 OraConnection::setCharset

```cpp
void setCharset(const CRString& value);
```

**Parameters**

**value**
String with a new character set.

**Description**
Sets character set that OCL uses to read and write character data. Supported with Oracle 8 client only.

**See Also**
OraConnection::isUnicode, OraConnection::setUnicode, OraConnection::charset

---

### 8.14.42 OraConnection::charset

```
CRString charset();
```

**Return Value**
String with character set.

**Description**
Call this member function to determine character set is currently enabled.
### 8.14.43 `OraConnection::setServer`

```cpp
void setServer(const CRString& value);
```

**Parameters**

- `value`
  - String with a new TNS alias.

**Description**

Call this member function to set TNS alias name which will be used by `OraConnection` for connection to Oracle.

**See Also**

- `OraConnection::connect`, `OraConnection::server`, `OraConnection::setUsername`, `OraConnection::setPassword`, `OraConnection::setConnectString`

### 8.14.44 `OraConnection::setTransaction`

```cpp
void setTransaction(OraTransaction* trans);
```

**Parameters**

- `trans`
  - Distributed transaction to which this `OraConnection` object will be attached.

**Description**

Use this method to attach `OraConnection` instance to distributed transaction.

**See Also**

- `OraConnection::startTransaction`, `OraConnection::commit`, `OraConnection::rollback`
8.14.45 **OraConnection::setUsername**

```cpp
void setUsername(const CRString& value);
```

**Parameters**

- **value**
  - String with a user name.

**Description**

Call this member function to set username which will be used by OraConnection for connection to Oracle.

**See Also**

`OraConnection::connect`, `OraConnection::username`, `OraConnection::setPassword`, `OraConnection::setServer`, `OraConnection::setConnectString`

8.14.46 **OraConnection::startTransaction**

```cpp
void startTransaction();
void startTransaction(CRIsolationLevel level);
```

**Parameters**

- **level**
  - Type of the isolation level to use for this transaction. Must be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ilReadCommited</td>
<td>Specifies that this transaction will provide data consistency for single SQL queries only.</td>
</tr>
<tr>
<td>ilReadOnly</td>
<td>Specifies that all queries in this transaction will only read records.</td>
</tr>
<tr>
<td>ilSerializable</td>
<td>Specifies that this transaction will operate on one snapshot of data. Modifications done by other transactions either commited or not during the course of this transaction will not be seen for it.</td>
</tr>
</tbody>
</table>

**Description**

Call this member function to begin a new transaction against the database server. Before calling `startTransaction`, an application should check the status of the `inTransaction`
function. If `inTransaction` returns 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 throw an exception.
If user starts transaction then auto commit will not be done for all `OraRecordset` and `OraCommand` objects in current connection.
See Oracle documentation on further description of different types of isolation levels.

**Example**

```cpp
connection.startTransaction();
if (connection.inTransaction()) // returns true
    connection.commit();          // ends current transaction
if (not connection.inTransaction()) // returns false
    connection.startTransaction(); // begins new transaction
```

**See Also**

`OraConnection::commit`, `OraConnection::rollback`, `OraConnection::inTransaction`, `OraConnection::isolationLevel`

---

### 8.14.47 OraConnection::transaction

`OraTransaction* transaction() const;`

**Return Value**

Distributed transaction object in which this `OraConnection` object participate.

**Description**

Use this method to obtain distributed transaction which this `OraConnection` object participate.

**See Also**

`OraConnection::startTransaction`, `OraConnection::commit`, `OraConnection::rollback`

---

### 8.14.48 OraConnection::username

`CRString username();`

**Return Value**
String with the user name.

**Description**

Returns the username set previously by **setUsername** member function.

**See Also**

`OraConnection::setUsername`, `OraConnection::password`, `OraConnection::server`
8.15 OraCursor

#include "ora.h"

Members

Attributes

bool canFetch() const;
CRCursorState state() const;

CDA* getCDA();
OCIStmt* getOCIStmt() const;

Methods

void allocCursor();
void freeCursor();

Description

This class represents Oracle cursor. OCL uses it to store value of REF CURSOR parameter. Use getCursor function of OraParam to get reference to OraCursor object.

See Also

OraParam::getCursor, OraCommand::cursor

8.15.1 OraCursor::allocCursor

void allocCursor();

Description

Allocates required resources for the OraCursor objects.

See Also

OraCursor::freeCursor

OraCursor | Classes | OCL | Index
8.15.2 OraCursor::canFetch

bool canFetch() const;

**Return Value**

True if cursor is ready to be fetched.

**Description**

Call this member function to verify whether Oracle cursor may be fetched or not.

**See Also**

OraCursor::state

OraCursor | Classes | OCL | Index

8.15.3 OraCursor::freeCursor

void freeCursor();

**Description**

Releases the cursor resources.

**See Also**

OraCursor::allocCursor

OraCursor | Classes | OCL | Index

8.15.4 OraCursor::getCDA

CDA* getCDA();

**Return Value**

Pointer to the CDA structure.

**Description**

Call this member function to return CDA structure. Use this method for Oracle 7.


**See Also**

OraCursor::getOCIStmt, OraCursor::state

OraCursor | Classes | OCL | Index

### 8.15.5 OraCursor::getOCIStmt

OCIStmt* getOCIStmt() const;

**Return Value**

OCI statement handle.

**Description**

Call this member function to obtain the OCI statement handle. Use this method for Oracle 8 or above.

**See Also**

OraCursor::CDA, OraCursor::state

OraCursor | Classes | OCL | Index

### 8.15.6 OraCursor::state

CRCursorState state() const;

**Return Value**

State of cursor. Can be one of the following values:

- csInactive, csOpen, csParsed, csPrepared, csBound, csExecuting, csExecuted,
- csFetching, csFetchingAll, csCanceled

**Description**

Call this member function to verify the state of cursor.

**Example**

```c++
if (dataset.cursor().state() < csPrepared)
    return;
```

**See Also**

OraCursor::canFetch
# include "oracall.h"

Members

**Attributes**

static bool isThreaded();
static void setThreaded(bool value);

static bool isEvents();
static void setEvents(bool value);

static OCIVersion OCICallStyle();
static void setOCICallStyle(OCIVersion value);

static bool isOCIInit();

static CRString OCILibrary();
static void setOCILibrary(const CRString& name);

static bool isOCILoaded();

static int oracleHomeCount();
static int defaultOracleHome();

**Methods**

static void setInitMode(InitMode mode);

static void initOCI();
static void freeOCI();

static OCIEnv* getOCIEnv();
static OCIError* getOCIErr();

static OCIVersion getOCIVersion();

static CRString oracleHomeName(int index);
static CRString oracleHomePath(int index);

static void setOracleHome(int index);
static void setOracleHome(const CRString& homeName);

static void useDefaultOracleHome();

void static resetOracleHomes();
OraEnvironment represents interfaces to different parameters of Oracle client environment and provides methods to control OCI loading.

See Also

OraConnection::home, OraConnection::setHome

8.16.1 OraEnvironment::defaultOracleHome

static int defaultOracleHome();

Return Value

Default Oracle home index.

See Also

OraEnvironment::useDefaultOracleHome

8.16.2 OraEnvironment::freeOCI

static void freeOCI();

See Also

OraEnvironment::initOCI
8.16.3 OraEnvironment::getOCIEnv

static OCIEnv* getOCIEnv();

Return Value
Handle of the OCI environment.

Description
Use this member function to obtain OCI environment handle. For more information about this handle refer to OCI documentation.

See Also
OraEnvironment::getOCIError
OraEnvironment | Classes | OCL | Index

8.16.4 OraEnvironment::getOCIError

static OCIError* getOCIError();

Return Value
OCI error handle.

Description
Use this member function to obtain OCI error handle. For more information about this handle refer to OCI documentation.

See Also
OraEnvironment::getOCIEnv
OraEnvironment | Classes | OCL | Index

8.16.5 OraEnvironment::getOCIVersion

enum OCIVersion { None, OCI73, OCI80, OCI81, OCI90 };  
static OCIVersion getOCIVersion();

Return Value
Oracle client version.

**Description**

Use this member function to determine the version of Oracle client software.

**See Also**

OraEnvironment::OCILibrary

OraEnvironment | Classes | OCL | Index

### 8.16.6 OraEnvironment::initOCI

```cpp
static void initOCI();
```

**Description**

Call this member function to load OCI library into the application process.

**See Also**

OraEnvironment::freeOCI

OraEnvironment | Classes | OCL | Index

### 8.16.7 OraEnvironment::isEvents

```cpp
static bool isEvents();
```

**Return Value**

True, if OCI was initialized in event mode.

**Description**

Use this member function to determine if OCI is in imEvents mode or not.

**See Also**

OraEnvironment::setInitMode, OraEnvironment::setEvents

OraEnvironment | Classes | OCL | Index
8.16.8 OraEnvironment::isOCIInited

static bool isOCIInited();

Return Value

True, if OCI library was initialized.

Description

Use this member function to determine, if OCI library was initialized or not.

See Also

OraEnvironment::initOCI, OraEnvironment::freeOCI

8.16.9 OraEnvironment::isOCILoaded

static bool isOCILoaded();

Return Value

True, if OCI was loaded into the application process, otherwise false.

Description

Use this member function to determine if OCI library was loaded into the application process or not.

See Also

OraEnvironment::initOCI, OraEnvironment::freeOCI

8.16.10 OraEnvironment::isThreaded

static bool isThreaded();

Return Value

True, if OCI was initialized in multithreaded mode.
**Descrip pt on**

Use this method to determine if OCI is initialized in multithreaded mode. Default value is false.

**See Also**

OraEnvironment::setInitMode, OraEnvironment::setThreaded

8.16.11 **OraEnvironment::OCI.CallStyle**

static OCIVersion OCICallStyle();

**Return Value**

OCI call style that OCL uses.

**Descrip pt on**

Use this member function to obtain which subset of the OCI functionality will be used by OCL.

**See Also**

OraEnvironment::setOCICallStyle

8.16.12 **OraEnvironment::OCILibrary**

static CRString OCILibrary();

**Return Value**

Name of the OCI library.

**Descrip pt on**

Use this member function to obtain the name of OCI library.

**See Also**

OraEnvironment::setOCILibrary
8.16.13  OraEnvironment::oracleHomeCount

static int oracleHomeCount();

Return Value
Number of homes installed on the machine.

Description
Use this member function to obtain number of Oracle homes installed on the machine.

See Also
OraEnvironment::oracleHomeName, OraEnvironment::oracleHomePath

8.16.14  OraEnvironment::oracleHomeName

static CRString oracleHomeName(int index);

Return Value
Oracle home name that corresponds to specific index.

Parameters
index
Zero-based home index.

Description
Use this member function to obtain name of Oracle home specified by home index.

See Also
OraEnvironment::oracleHomeCount, OraEnvironment::oracleHomePath

8.16.15  OraEnvironment::oracleHomePath

static CRString oracleHomePath(int index);
**Return Value**

Path to Oracle home.

**Parameters**

`index`

Zero-based home index.

**Description**

Use this member function to obtain the path where Oracle home installed.

**See Also**

`OraEnvironment::oracleHomeCount`, `OraEnvironment::oracleHomeName`

---

**8.16.16 OraEnvironment::resetOracleHomes**

```cpp
void static resetOracleHomes();
```

**Description**

Use this member function to clear the information about all Oracle homes.

**See Also**

`OraEnvironment::setOracleHome`

---

**8.16.17 OraEnvironment::useDefaultOracleHome**

```cpp
static void useDefaultOracleHome();
```

**Description**

Use this member function to set default Oracle home to the current home from which OCI library will be loaded.

**See Also**

`OraEnvironment::defaultOracleHome`
8.16.18 OraEnvironment::setEvents

static void setEvents(bool value);

Parameters

True if OCI will be initialized in OCI EVENTS mode, otherwise false.

Description

This method utilizes publish-subscribe notifications. You must use this method before OCI initialization and therefore before the first connection or initOCI call. If OCI has already been initiated you must call freeOCI method and then init OCI again. For more information see OCI Programmer's Guide in Oracle documentation.

See Also

OraEnvironment::setInitMode, OraEnvironment::isEvents

8.16.19 OraEnvironment::setOracleHome

static void setOracleHome(int index);
static void setOracleHome(const CRString& homeName);

Parameters

index

Zero-based Oracle home index.

homeName

Oracle home name.

Description

Use this member function to set Oracle home with specified Oracle home index or name and then use this home as current Oracle home.

See Also

OraEnvironment::oracleHomeCount, OraEnvironment::oracleHomeName, OraEnvironment::resetOracleHomes

8.16.20 OraEnvironment::setInitMode

```cpp
enum InitMode {
    imDefault, // OCI_DEFAULT
    imThreaded, // OCI_THREAD
    imThreadedUnmutexed, // OCI_THREAD | OCI_ENV_NO_MUTEX
    imObject, // OCI_OBJECT
    imShared, // OCI_SHARED
    imNoUserCallbacks, // OCI_NO UCB
    imEvents // OCI_EVENTS
};
static void setInitMode(InitMode mode);
```

**Parameters**

`mode`

Mode in which OCI will be initialized.

**Description**

Use this method to specify init mode in which OCI will be initialized. You must use this method before OCI initialization and therefore before the first connection or `initOCI` call. If OCI has already been initiated you must call `freeOCI` method and then init OCI again. For more information see OCI Programmer's Guide: `OCIInitialize`, `OCIEnvInit` and `OCIEnvCreate` functions.

**See Also**

`OraEnvironment::initOCI`, `OraEnvironment::freeOCI`

**OraEnvironment | Classes | OCL | Index**

8.16.21 OraEnvironment::setOCICallStyle

```cpp
static void setOCICallStyle(OCIVersion value);
```

**Parameters**

`value`

OCI call style which will be used by OCL.

**Description**

Use this member function to specify which subset of the OCI functionality will be used by OCL.
8.16.22 **OraEnvironment::setOCILibrary**

static void setOCILibrary(const CRString& name);

**Parameters**

- **name**
  OCI library name.

**Description**

Use this member function to specify the name of OCI library.

**See Also**

- OraEnvironment::OCILibrary

8.16.23 **OraEnvironment::setThreaded**

static void setThreaded(bool value);

**Parameters**

- **value**
  True, if OCI must be initialized in multithreaded mode, otherwise false.

**Description**

Use this member function to initialize OCI in multithreaded mode. The default value is false. You must use this method before OCI initialization and therefore before the first connection or initOCI call. If OCI has already been initiated you must call freeOCI method and then init OCI again. For more information see OCI Programmer's Guide in Oracle documentation.

**See Also**

- OraEnvironment::setInitMode, OraEnvironment::isThreaded
# include "ora.h"

**Hierarchy**

<table>
<thead>
<tr>
<th>CRException</th>
</tr>
</thead>
<tbody>
<tr>
<td>OraException</td>
</tr>
</tbody>
</table>

**Members**

**Construct on**

OraException(const CRString& msg);
OraException(int errorCode, const CRString& msg);

**Attributes**

int errorCode();

**Description**

OCL uses OraException object to throw Oracle related errors. Most of these errors come from database server. OraException objects are created by OCL objects such as OraConnection, OraCommand and OraRecordset. Use **errorCode** member function to get the code of error.

**See Also**

CRException

**Classes** | **OCL** | **Index**

---

**8.17.1 OraException::errorCode**

int errorCode();

**Return Value**

The error code, always positive.

**Description**

Call this member function to determine the error code returned by Oracle.
Example

```cpp
try {
    SQL.execute();
} catch (OraException& e) {
    if (e.errorCode() != 942) // check for ORA-00942
        throw;
}
```

See Also

CRException::message

OraException

8.17.2 OraException::OraException

OraException(const CRString& msg);
OraException(int errorCode, const CRString& msg);

Parameters

errorCode
   Indicates the error code to be copied into the new OraException.

msg
   Indicates the message string to be copied into the new OraException.

Description

This member function constructs a new OraException object.

See Also

OraException::errorCode

OraException
# include "ora.h"

Hierarchy

CRField
   |
OraField

Members

Construct on

OraField();
~OraField();

Attributes

int size() const;

bool isComplex();

const char* getChars();
char* getChars(char* result);
CRString getString();

CRLob& getLob();
void setLob(CRLob& value);

OraLob& getBlob();
void setBlob(OraLob& value);
OraLob& getBlob();
void setBlob(CRLob& value);
OraLob& getBlob();
void setBlob(CRLob& value);
OraLong& getLong();
void setLong(OraLong& value);
OraLong& getLongRaw();
void setLongRaw(OraLong& value);

OraFile& getBFile();
void setBFile(OraFile& value);

Description

This class implements methods to manipulate a single field from the database table. COraField inherits common functionality from the CRField class and also provides additional methods to work with LOB objects.
### 8.18.1 OraField::getBFile

OraFile& getBFile();

**Return Value**

OraFile object reference.

**Description**

Call this member function to retrieve the reference to OraFile object that represents Oracle BLOB value of the field in the current record. Throws an exception if the field value type may not be cast to OraFile.

**Example**

```c++
CRString desc;
desc = query.field("Description").getBFile().getString();
```

**See Also**

OraFile, OraField::setBFile

---

### 8.18.2 OraField::getBlob

OraLob& getBlob();

**Return Value**

OraLob object reference.

**Description**

Call this member function to retrieve the reference to OraLob object that represents Oracle BLOB value of the field in the current record. Throws an exception if the field value type may not be cast to OraLob.

**Example**

```c++
```
CRString desc;
desc = query.field("Description").getBlob().getString();

See Also

OraLob, CRField::getLob, OraField::isComplex

OraField | Classes | OCL | Index

8.18.3 OraField::getChars

const char* getChars();
char* getChars(char* result);

Return Value

Zero-terminated string.

Parameters

result

Buffer long enough to hold zero-terminated copy of the field value.

Description

Call this member function to retrieve field data in the current record represented as a string of chars.

Example

char* dName;
char buffer[100];

dName = query.field("DName").getChars();
dataset.field("DName").getChars(buffer);

See Also

OraField::getString

OraField | Classes | OCL | Index

8.18.4 OraField::getWChars

const wchar_t* getChars();
wchar_t* getChars(wchar_t* result);

Return Value
Zero-terminated UTF16 string.

**Parameters**

`result`
Buffer long enough to hold zero-terminated copy of the field value.

**Description**
Call this member function to retrieve field data in the current record represented as a string of chars. Use this function for connections in Unicode mode only.

**Example**

```c++
wchar_t* dName;
wchar_t buffer[100];

dName = query.field("DName").getWChars();
dataset.field("DName").getWChars(buffer);
```

**See Also**

- `OraField::getWString`
- `OraLob`, `OraField::setClob`, `OraField::isComplex`
8.18.6 OraField::getLob

CRLob& getLob();

Return Value

CRLob object reference.

Description

Call this member function to retrieve the reference to CRLob object that represents blob value of the field in the current record. Throws an exception if the field value type may not be cast to CRLob.

Example

CRString desc;
desc = dataset.field("Description").getLob().getString();

See Also

CRLob, CRField::getBlob, OraField::isComplex

OraField | Classes | OCL | Index

8.18.7 OraField::getLong

OraLong& getLong();

Return Value

OraLong object reference.

Description

Call this member function to retrieve the reference to OraLong object that represents blob value of the field in the current record. Throws an exception if the field value type may not be cast to OraLong.

Example

CRString desc;
desc = dataset.field("Description").getLong().getString();

See Also

OraLong, OraField::getBlob, OraField::isComplex
8.18.8 OraField::getLongRaw

OraLong & getLongRaw();

Return Value

OraLong object reference.

Description

Call this member function to retrieve the reference to OraLong object that represents blob value of the field in the current record. Throws an exception if the field value type may not be cast to OraLong.

Example

```cpp
CRString desc;
desc = dataset.field("Description").getLongRaw().getString();
```

See Also

OraLong, OraField::getBlob, OraField::isComplex

8.18.9 OraField::getWString

CRWString getWString();

Return Value

String representaion of the field value.

Description

Call this member function to retrieve field data in the current record represented as Unicode string. No convertion from nonUnicode string to a CRWString value are performed. Use this function for connections in Unicode mode only.

Example

```cpp
CRWString dName;
dName = dataset.field("DName").getWString();
```

See Also
8.18.10 OraField::getString

CRString getString();

Return Value
String representation of the field value.

Description
Call this member function to retrieve field data in the current record represented as string. The value of the field will be converted to a CRString value if possible.

Example
CRString dName;
dName = dataset.field("DName").getString();

See Also
CRString, OraField::setString, OraField::isNull

8.18.11 OraField::isComplex

bool isComplex() const;

Return Value
True if the field is either a Lob or a container value.

Description
Use this member function to verify whether this field is either a Lob or a container value.

8.18.12 OraField::OraField

OraField();
This member function constructs a new OraField object.

**See Also**

OraField::size

---

### 8.18.13 OraField::setBFile

```cpp
void setBFile(OraFile & value);
```

**Parameters**

*value*

OraFile object.

**Descr pt on**

Call this member function to assign OraFile object to the field in the current record. Throws an exception if the *value* type may not be cast to the current type of the field value.

**See Also**

OraFile, OraField::getBFile, CRField::setNull

---

### 8.18.14 OraField::setBlob

```cpp
void setBlob(OraLob & value);
```

**Parameters**

*value*

OraLob object.

**Descr pt on**

Call this member function to assign OraLob object to the field in the current record. Throws an exception if the *value* type may not be cast to the current type of the field value.

**See Also**
8.18.15 OraField::setClob

void setClob(OraLob & value);

Parameters

value
 OraLob object.

Description

Call this member function to assign OraLob object to the field in the current record. Throws an exception if the value type may not be cast to the current type of the field value.

See Also

OraLob, OraField::getBlob, CRField::setBlob, CRField::setNull

8.18.16 OraField::setLob

void setLob(CRLob & value);

Parameters

value
 CRLob object.

Description

Call this member function to assign CRLob object to the field in the current record. Throws an exception if the value type may not be cast to the current type of the field value.

See Also

CRLob, OraField::getBlob, CRField::setBlob, CRField::setNull
8.18.17 OraField::setLong

```cpp
void setLong(OraLong& value);
```

**Parameters**

`value`

OraLong object.

**Description**

Call this member function to assign CRLob object to the field in the current record.

**See Also**

OraLong, OraField::getLong, OraField::setBlob, CRField::setNull

OraField | Classes | OCL | Index

8.18.18 OraField::setLongRaw

```cpp
void setLongRaw(OraLong& value);
```

**Parameters**

`value`

OraLong object.

**Description**

Call this member function to assign CRLob object to the field in the current record.

**See Also**

OraLong, OraField::getLongRaw, OraField::setBlob, CRField::setNull

OraField | Classes | OCL | Index

8.18.19 OraField::size

```cpp
int size() const;
```

**Return Value**

Size of the field value.
Description

Returns data size of the field.

See Also

CRField::scale

OraField | Classes | OCL | Index
# OraFile

```c
#include "ora.h"
```

## Hierarchy

```
CRLob
 |    OraLob
 |    OraFile
```

## Members

### Attributes

- `bool exist();`
- `CRString fileDir();`
- `CRString fileName();`
- `bool isOpen();`
- `void setFileDir(const CRString& value);`
- `void setFileName(const CRString& value);`

### Operations

- `void allocLob();`
- `void freeLob();`
- `void close();`
- `void open();`
- `void refresh();`

## Description

OraFile class provides methods to retrieve, modify and store Oracle file binary objects.

## See Also

- `CRLob`, `OraLob`

## Classes | OCL | Index

---

### 8.19.1 OraFile::allocLob

```c
void allocLob();
```

## Description
Call this member function to allocate and initialize BFILE locator.

See Also

OraFile::freeLob
OraFile

8.19.2 OraFile::close

void close();

Description

This member function closes previously opened file associated with this object.

See Also

OraFile::open
OraFile

8.19.3 OraFile::exist

bool exist();

Return Value

True if binary file exists.

Description

Returns nonzero value if a binary file refered to by the Lob locator of this object exists.

See Also

OraFile::fileDir, OraFile::fileName
OraFile
8.19.4 *OraFile::fileDir*

CRString fileDir();

**Return Value**

Alias name of a directory for the file Lob objects.

**Description**

Call this member function to get the alias name of the directory for the file Lob objects that were previously set with the `setFileDir` method.

**See Also**

*OraFile::fileName*, *OraFile::setFileDir*

OraFile | Classes | OCL | Index

8.19.5 *OraFile::fileName*

CRString fileName();

**Return Value**

File name for this file Lob object.

**Description**

Call this member function to get the file name for this file Lob object that was previously set with the `setName` method.

**See Also**

*OraFile::fileDir*, *OraFile::setName*

OraFile | Classes | OCL | Index

8.19.6 *OraFile::freeLob*

void freeLob();

**Description**
Call this member function to release BFILE locator.

**See Also**

*OraFile::allocLob*

*OraFile | Classes | OCL | Index*

### 8.19.7 *OraFile::isOpen*

```cpp
bool isOpen();
```

**Return Value**

True if binary file is already opened.

**Description**

Returns true if the file associated with this object is opened on the server.

**See Also**

*OraFile::open, OraFile::close*

*OraFile | Classes | OCL | Index*

### 8.19.8 *OraFile::open*

```cpp
void open();
```

**Description**

This member function opens file associated with this object on the server.

**See Also**

*OraFile::setFileDir, OraFile::setFileName, OraFile::close*

*OraFile | Classes | OCL | Index*
8.19.9 OraFile::refresh

void refresh();

Description

This member function opens on the server file associated with this object and reads all
Lob content. If the file doesn’t exist on the server refresh throws an exception.

See Also

OraFile::open, OraLob::readLob

8.19.10 OraFile::setFileDir

void setFileDir(const CRString& value);

Parameters

value
  
  Alias name of a directory.

Description

Call this member function to set the alias name of the directory for the file Lob objects
that will be searched for the BFILE objects.
Directory alias should be created with the CREATE DIRECTORY command.

See Also

OraFile::setFileName, OraFile::fileDir

8.19.11 OraFile::setFileName

void setFileName(const CRString& value);

Parameters

value
File name for this file Lob object.

**Description**

Call this member function to set the file name for this file Lob object.

**See Also**

OraFile::setFileDir, OraFile::fileName

OraFile | Classes | OCL | Index
# OraInterval

```cpp
#include "ora.h"

Construct on

OraInterval();
OraInterval(int year, int month);
OraInterval(int day, int hour, int minute, int second, int fsecond);

OraInterval(const OraInterval& src);
OraInterval& operator=(const OraInterval& r);

~OraInterval();

Attributes

OraDataType intervalType() const;
void setIntervalType(OraDataType type);
void setYearMonth(int year, int month);
void setDaySecond(int day, int hour, int minute, int second, int fsecond);

int year() const;
int month() const;
int day() const;
int hour() const;
int minute() const;
int second() const;
int fracSecond() const;

bool isNull() const;
void setNull();

Methods

CRString toString(unsigned char lprec = 0, unsigned char fsprec = 6) const;
void fromString(const CRString& val);

int compare(const OraInterval& value) const;
OraInterval& operator+=(const OraInterval& r);
OraInterval& operator-=(const OraInterval& r);

friend OraInterval operator+(const OraInterval& l, const OraInterval& r);
friend OraInterval operator-(const OraInterval& l, const OraInterval& r);
```
friend bool operator==(const OraInterval& l, const OraInterval& r);
friend bool operator>(const OraInterval& l, const OraInterval& r);
friend bool operator<(const OraInterval& l, const OraInterval& r);
friend bool operator!=(const OraInterval& l, const OraInterval& r);
friend bool operator>=(const OraInterval& l, const OraInterval& r);
friend bool operator<=(const OraInterval& l, const OraInterval& r);

**Description**

This class represents Oracle data type such as INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND.

For using OraInterval see [OraTimeStamp](#) class description.

**Classes** | **OCL** | **Index**

---

### 8.20.1 OraInterval::compare

```cpp
int compare(const OraInterval& value) const;
```

**Return Value**

-1 - if current object less than value, 0 - if current object and value are equal, 1 - if current object greater than value.

**Parameters**

`value`

Object that compared with current object.

**Description**

Compares current object and values.

**See Also**

`operator==, operator!=`

---

### 8.20.2 OraInterval::day

```cpp
int day() const;
```

**Return Value**

---

---
Day part of the current object.

**Description**

Call this member function to return day part of the object. Current object type must be `dtIntervalDS`.

**See Also**

`OraInterval::hour`, `OraInterval::minute`, `OraInterval::second`, `OraInterval::fracSecond`

---

### 8.20.3 `OraInterval::fracSecond`

```cpp
int fracSecond() const;
```

**Return Value**

Fractional second part of the current object.

**Description**

Call this member function to return fractional second part of the object. Current object type must be `dtIntervalDS`.

**See Also**

`OraInterval::day`, `OraInterval::hour`, `OraInterval::minute`, `OraInterval::second`

---

### 8.20.4 `OraInterval::toString`

```cpp
CRString toString(unsigned char lfprec = 0, unsigned char fsprec = 6) const;
```

**Return Value**

String that contains `OraInterval` object value.

**Parameters**

*lfprec*  
Leading field precision (the number of digits used to represent the leading field).
Fractional second precision of the interval (the number of digits used to represent the fractional seconds).

Use this member function to get string representation of OraInterval object value.

See Also

OraInterval::fromString

OraInterval::hour

Return Value

Hour part of the current object.

Call this member function to return hour part of the object. Current object type must be dtIntervalDS.

See Also

OraInterval::day, OraInterval::minute, OraInterval::second, OraInterval::fracSecond

OraInterval::intervalType

Return Value

dtIntervalDS - INTERVAL DAY TO SECOND type,
dtIntervalYM - INTERVAL YEAR TO MONTH type.

Call this member function to get the object type. Value can be one of the two INTERVAL types.
8.20.7 OraInterval::isNull

bool isNull() const;

Return Value

True, if value is NULL; otherwise false.

Description

Use this member function to determine if value of the current object is NULL.

Example

...  
    if (!intervalDS.isNull())    
        std::cout << intervalDS.getString() << std::endl;  
    else    
        std::cout << "TIMESTAMP is NULL" << std::endl;  
...

See Also

OraInterval::setNull

8.20.8 OraInterval::minute

int minute() const;

Return Value

Minute part of the current object.

Description

Call this member function to return minute part of the object. Current object type must be dtIntervalDS.
See Also

OraInterval::day, OraInterval::hour, OraInterval::second, OraInterval::fracSecond

8.20.9 OraInterval::month

int month() const;

Return Value

Month part of the current object.

Description

Call this member function to return month part of the object. Current object type must be dtIntervalYM.

See Also

OraInterval::year

8.20.10 OraInterval::operator -

OraInterval& operator-=(const OraInterval& r);

Return Value

Object value after subtracting.

Parameters

value

Interval to be subtracted.

Description

Use this operator to subtract OraInterval object from the current object value.

See Also

operator+
8.20.11 OraInterval::operator !=

bool operator!=(const OraInterval& l, const OraInterval& r);

**Return Value**

True, if the OraInterval object on the left side of the operator is not equal to the OraInterval object on the right side; otherwise false.

**Parameters**

`l`

Left-hand value of an operator.

`r`

Right-hand value of an operator.

**Describe on**

Use this operator to test if the OraInterval object on the left side of the operator is not equal to the OraInterval object on the right side.

**See Also**

operator==

8.20.12 OraInterval::operator +

OraInterval& operator+=(const OraInterval& r);

**Return Value**

Object value after adding.

**Parameters**

`r`

Interval to be added.

**Describe on**
Use this operator to add OraInterval object to the current object value.

See Also

operator-

OraInterval | Classes | OCL | Index

8.20.13 OraInterval::operator +=

OraInterval& operator+=(const OraInterval& r);

Return Value

Object value after adding.

Parameters

r
   Interval to be added.

Description

Use this operator to add OraInterval object to the current object.

See Also

operator-=

OraInterval | Classes | OCL | Index

8.20.14 OraInterval::operator <

bool operator<(const OraInterval& l, const OraInterval& r);

Return Value

True, if the OraInterval object on the left side of the operator is less than the OraInterval object on the right side; otherwise false.

Parameters

l
   Left-hand value of an operator.
Right-hand value of an operator.

**Description**

Use this operator to test if the `OraInterval` object on the left side of the operator is less than the `OraInterval` object on the right side.

**See Also**

`operator>`

`OraInterval` | Classes | OCL | Index

### 8.20.15 `OraInterval::operator <=`

```cpp
bool operator<=(const OraInterval& l, const OraInterval& r);
```

**Return Value**

True if the `OraInterval` object on the left side of the operator is less than or equal to the `OraInterval` object on the right side; otherwise false.

**Parameters**

`l`

Left-hand value of an operator.

`r`

Right-hand value of an operator.

**Description**

Use this operator to test if the `OraInterval` object on the left side of the operator is less than or equal to the `OraInterval` object on the right side.

**See Also**

`operator>=`

`OraInterval` | Classes | OCL | Index

### 8.20.16 `OraInterval::operator -=`

```cpp
OraInterval& operator-=(const OraInterval& r);
```
Return Value

Object value after subtracting.

Parameters

\( r \)

Interval to be subtracted.

Description

Use this operator to subtract OraInterval object from the current object.

See Also

\[ \text{operator-=} \]

OraInterval | Classes | OCL | Index

8.20.17 OraInterval::operator ==

bool operator==(const OraInterval& l, const OraInterval& r);

Return Value

True, if OraInterval object on the left side of the operator is equal to OraInterval object on the right side; otherwise false.

Parameters

\( l \)

Left-hand value of an operator.

\( r \)

Right-hand value of an operator.

Description

Use this operator to test if the OraInterval object on the left side of the operator is equal to the OraTime%Stamp object on the right side.

See Also

\[ \text{operator!=} \]

OraInterval | Classes | OCL | Index
8.20.18 OraInterval::operator >

bool operator>>(const OraInterval& l, const OraInterval& r);

Return Value

True, if the OraInterval object on the left side of the operator is greater than the OraInterval object on the right side; otherwise false.

Parameters

l

Left-hand value of an operator.

r

Right-hand value of an operator.

Description

Use this operator to test if the OraInterval object on the left side of the operator is greater than the OraInterval object on the right side.

See Also

operator<

OraInterval | Classes | OCL | Index

8.20.19 OraInterval::operator >=

bool operator>=(const OraInterval& l, const OraInterval& r);

Return Value

True, if the OraInterval object on the left side of the operator is greater than or equal to the OraInterval object on the right side; otherwise false.

Parameters

l

Left-hand value of an operator.

r

Right-hand value of an operator.

Description

Use this operator to test if the OraInterval object on the left side of the operator is greater than or equal to the OraInterval object on the right side.

See Also

operator<
Use this operator to test if the OraInterval object on the left side of the operator is greater than or equal to the OraInterval object on the right side.

**See Also**

operator<=

OraInterval | Classes | OCL | Index

### 8.20.20 OraInterval::OraInterval

OraInterval();
OraInterval(int year, int month);
OraInterval(int day, int hour, int minute, int second, int fsecond);

OraInterval(const OraInterval& src);
OraInterval& operator=(const OraInterval& r);
~OraInterval();

#### Parameters

*day*
  Day value.

*hour*
  Hour value.

*minute*
  Minute value.

*second*
  Second value.

*fsecond*
  Fractional second value.

*src*
  Object that copied.

*r*
  Object that content assigned to current object.

#### Description

First constructor create uninitialized OraInterval object without type. You must call setIntervalType function to initialize object with type before calling any other function.
Second constructor create OraInterval object with INTERVAL YEAR TO MONTH type and initialize it with specified values.
Third constructor create OraInterval object with INTERVAL DAY TO SECOND type and
initialize it with specified values.

See Also

OraInterval::setIntervalType

OraInterval

8.20.21 OraInterval::second

int second() const;

Return Value

Second part of the current object.

Description

Call this member function to return second part of the object. Current object type must be dtIntervalDS.

See Also

OraInterval::day, OraInterval::hour, OraInterval::minute, OraInterval::fracSecond

8.20.22 OraInterval::setDaySecond

void setDaySecond(int day, int hour, int minute, int second, int fsecond);

Parameters

day
    Day value.

hour
    Hour value.

minute
    Minute value.

second
    Second value.

fsecond
Fractional second value.

**Descr pt on**

Use this member function to set value for object that represents INTERVAL DAY TO SECOND type. Current object type must be dtIntervalDS.

**See Also**

*OraInterval::setYearMonth*

*OraInterval* | Classes | OCL | Index

---

**8.20.23 OraInterval::setIntervalType**

```cpp
void setIntervalType(OraDataType type);
```

**Parameters**

*type*  
One of the following INTERVAL types:
- dtIntervalDS - INTERVAL DAY TO SECOND type,
- dtIntervalYM - INTERVAL YEAR TO MONTH type.

**Descr pt on**

Use this member function to set new type that this object represents. If the old value exists it will be deleted.

**See Also**

*OraInterval::intervalType*

*OraInterval* | Classes | OCL | Index

---

**8.20.24 OraInterval::setNull**

```cpp
void setNull();
```

**Descr pt on**

Set current OraTimeStamp object value to NULL.

**See Also**

*OraInterval::isNull*
8.20.25  OraInterval::fromString

void fromString(const CRString& value);

**Parameters**

`value`
The input string to be converted to an Oracle interval.

**Description**
Use this member function to convert string value to OraInterval value.

**See Also**
OraInterval::toString

8.20.26  OraInterval::setYearMonth

void setYearMonth(int year, int month);

**Parameters**

`year`
Year value.

`month`
Month value.

**Description**
Use this member function to set year and month values to the current object. Type of the current object must be dtIntervalYM.

**Example**
OraInterval intervalYM;
intervalYM.setIntervaltType(dtIntervalYM);
intervalYM.setYearMonth(150, 132);
...

See Also

OraInterval::setDaySecond

OraInterval | Classes | OCL | Index

8.20.27 OraInterval::year

int year() const;

Return Value

Year part of the current object.

Description

Call this member function to return year part of the object. Current object type must be dtIntervalYM.

See Also

OraInterval::month

OraInterval | Classes | OCL | Index
8.21 OraLoader

#include "oraloader.h"

Members

Constructor

OraLoader();
OraLoader(OraConnection& connection, const CRString& tableName);

~OraLoader();

Attributes

OraConnection* connection();
void setConnection(OraConnection* connection);

CRString tableName();
void setTableName(const CRString& tableName);

int columnCount();

bool breakOnError() const;
void setBreakOnError(bool value);

int columnError() const;
int rowError() const;

Methods

void addColumn(LoaderColumn& column);
void addColumn(const CRString& name, OraDataType dataType, int size, int precision, const CRString& dateFormat = "");

void getColumn(int index, LoaderColumn& column);
void getColumn(const CRString name, LoaderColumn& column);

void createColumns();
void clearColumns();

void beginLoad();

void finishLoad();

void putNextRow();

void putColumnData(int columnIndex, const CRString& data);
void putColumnData(const CRString& columnName, const CCRString& data);
void putColumnData(int columnIndex, const char* data);
void putColumnData(const CRString& columnName, const char* data);
void putColumnData(int columnIndex, const CRDate& data);
void putColumnData(const CRString& columnName, const CRDate& data);
void putColumnData(int columnIndex, int data);
void putColumnData(const CRString& columnName, int data);
void putColumnData(int columnIndex, double data);
void putColumnData(const CRString& columnName, double data);
void putColumnData(int columnIndex, const CRValue& data);
void putColumnData(const CRString& columnName, const CRValue& data);
void putColumnNull(int columnIndex);

Description

OraLoader allows to load external data into the Oracle database. OraLoader uses direct path load interface to speed up loading. To specify the name of loading table use tableName member function. Use getColumn method to access to individual columns.

See Also

LoaderColumn

Classes  |  OCL  |  Index

8.21.1 OraLoader::addColumn

void addColumn(LoaderColumn& column);
void addColumn(const CRString& name, OraDataType dataType,
               int size, int precision, const CRString& dateFormat = "");

Parameters

column
    LoaderColumn object that will be added to OraLoader column collection.

name
    Column name.

dataType
    Column data type.
**size**
Column data size in bytes.

**precision**
The maximum number of digits used to represent the column value.

**dateFormat**
Date format into which loaded data will be converted.

**Description**
First method adds existing LoaderColumn object to OraLoader column collection. Second method automatically creates LoaderColumn object with specified properties and adds it to OraLoader columns.

**See Also**
OraLoader::columnCount, OraLoader::createColumns, OraLoader::clearColumns

8.21.2 **OraLoader::beginLoad**

void beginLoad();

**Description**
Use this member function to prepare internal OraLoader structures for loading. If you do not call this method explicitly it will be called automatically.

**See Also**
OraLoader::finishLoad, OraLoader::putNextRow

8.21.3 **OraLoader::breakOnError**

bool breakOnError() const;

**Return Value**
True, if OraLoader throws an exception when error occurred, otherwise false.

**Description**
Use this member function to determine OraLoader behaviour when error occurred.
8.21.4 OraLoader::clearColumns

void clearColumns();

Description
Use this member function to clear OraLoader column collection.

See Also
OraLoader::createColumns, OraLoader::addColumn, OraLoader::columnCount

8.21.5 OraLoader::columnCount

int columnCount();

Return Value
Number of columns in OraLoader column collection.

Description
Use this member function to retrieve the number of columns that OraLoader contains.

See Also
OraLoader::createColumns, OraLoader::addColumn, OraLoader::clearColumns

8.21.6 OraLoader::columnError

int columnError() const;

Return Value
Column number where error occurred.
Descr pt on

Use this member function to retrieve number of the column where error occurred.

See Also

OraLoader::rowError

8.21.7 OraLoader::connection

OraConnection* connection();

Return Value

OraConnection object used by this OraLoader.

Descr pt on

OraLoader uses OraConnection object during loading and for populating columns collection when createColumns method is called. For this cases OraLoader requires open connection.

See Also

OraLoader::setConnection

8.21.8 OraLoader::createColumns

void createColumns();

Descr pt on

Use this member function to retrieve table columns information from the table specified in the tableName attribute and populate OraLoader column collection. If column collection consists of columns they will be deleted after calling this method. Table name must be specified before calling this method.

See Also

OraLoader::addColumn, OraLoader::getColumn, OraLoader::clearColumns
8.21.9 OraLoader::finishLoad

void finishLoad();

Description

You must call this member function after finishing loading session. If loader internal buffer holds rows, they will be flushed to the table. If these rows contain errors no exception will be thrown.

See Also

OraLoader::beginLoad

OraLoader | Classes | OCL | Index

8.21.10 OraLoader::getColumn

void getColumn(int index, LoaderColumn& column);
void getColumn(const CRString name, LoaderColumn& column);

Parameters

index
Zero-based column index.

name
Column name.

column
Returned LoaderColumn object.

Description

Use this method to retrieve LoaderColumn object from OraLoader column collection.

See Also

OraLoader::columnCount, OraLoader::createColumns, OraLoader::clearColumns

OraLoader | Classes | OCL | Index

8.21.11 OraLoader::OraLoader

OraLoader();
OraLoader(OraConnection& connection, const CRString& tableName);

**Parameters**

*connection*  
OraConnection object that represents the connection to Oracle database.

*tableName*  
Name of the table that will be loaded.

**Description**

This member function constructs a new OraLoader with the table name that will be loaded and connection object.

**See Also**

OraLoader::createColumns

OraLoader | Classes | OCL | Index

### 8.21.12 OraLoader::putColumnData

```cpp
void putColumnData(int columnIndex, const CRString& data);
void putColumnData(const CRString& columnName, const CCRString& data);

void putColumnData(int columnIndex, const char* data);
void putColumnData(const CRString& columnName, const char* data);

void putColumnData(int columnIndex, const CRDate& data);
void putColumnData(const CRString& columnName, const CRDate& data);

void putColumnData(int columnIndex, int data);
void putColumnData(const CRString& columnName, int data);

void putColumnData(int columnIndex, double data);
void putColumnData(const CRString& columnName, double data);

void putColumnData(int columnIndex, const CRValue& data);
void putColumnData(const CRString& columnName, const CRValue& data);
```

**Parameters**

*columnIndex*  
Zero-based column index.

*columnName*  
Column name.
data
  Column value that will be loaded into the table.

**Descr pt on**

Use this member function to assign the value to column in the current row with specified column index or name.

**See Also**

`OraLoader::beginLoad`, `OraLoader::finishLoad`, `OraLoader::putNextRow`

**8.21.13 OraLoader::putColumnNull**

```cpp
void putColumnNull(int columnIndex);
```

**Parameters**

`columnIndex`
  Zero-based column index.

**Descr pt on**

Use this member function to assign NULL value to specified column in the current row.

**See Also**

`OraLoader::putColumnData`, `OraLoader::putNextRow`

**8.21.14 OraLoader::putNextRow**

```cpp
void putNextRow();
```

**Descr pt on**

Advance the OraLoader to the next row. Call this method after you putting row to the OraLoader internal buffer. If OraLoader rows buffer is full putNextRow method flush rows buffer to the loading table.

**See Also**

`OraLoader::putColumnData`, `OraLoader::putColumnNull`
### 8.21.15 OraLoader::rowError

```cpp
text
int rowError() const;
```

**Return Value**

Row number where error occurred.

**Description**

Use this member function to determine the row where error occurred.

**See Also**

OraLoader::columnError

---

### 8.21.16 OraLoader::setBreakOnError

```cpp
text
void setBreakOnError(bool value);
```

**Parameters**

`value`

True - OraLoader throws exception when error occurred.
False - when error occurred OraLoader ignores it and continue loading up to the end of data.

**Description**

Use this member function to set OraLoader behaviour that specifies if to break or not loading data when error occurred.

**See Also**

OraLoader::breakOnError

---

### 8.21.17 OraLoader::setConnection

```cpp
text
void setConnection(OraConnection* connection);
```

**Parameters**
**connect on**

OraConnection object used by this OraLoader.

**Description**

OraLoader uses OraConnection object during loading and for populating columns collection when `createColumns` method is called. For this cases OraLoader requires open connection.

**See Also**

`OraLoader::connection`

`OraLoader` | `Classes` | `OCL` | `Index`

---

### 8.21.18 `OraLoader::setTableName`

`void setTableName(const CRString& tableName);`

**Parameters**

`tableName`

Table name that will be loaded.

**Description**

Use this member function to set name of the table that will be loaded. You must set table name before beginning of loading data.

**See Also**

`OraLoader::tableName`

---

### 8.21.19 `OraLoader::tableName`

`CRString tableName();`

**Return Value**

Table name that will be loaded.

**Description**

Use this member function to retrieve name of the table that will be loaded. Note that table name must be specified before beginning of loading data.
See Also

OraLoader::setTableName
# include "ora.h"

\* Hierarchy
\*

**CRLob**

| OraLob

**Members**

**Attributes**

void disableBuffering();

void enableBuffering();

OCILobLocator* getOCILobLocator();

OCISvcCtx* getOCISvcCtx();

void setOCISvcCtx(OCISvcCtx *hSvcCtx);

bool isInit();

unsigned int length();

**Operations**

void allocLob();

void freeLob();

void init();

void readLob();

void writeLob();

**Description**

OraLob class provides methods to retrieve, modify and store BLOB and CLOB Oracle objects.

**See Also**

CRLob, OraFile

**Classes | OCL | Index**

---

### 8.22.1 OraLob::allocLob

void allocLob();

**Description**
Call this member function to allocate and initialize LOB locator.

**See Also**

*OraLob::freeLob*, *OraLob::init*

**8.22.2 OraLob::disableBuffering**

```c
void disableBuffering();
```

**Description**

Disables LOB buffering for the input internal LOB locator. The next time data is read from or written to the LOB through the input locator, the LOB buffering subsystem is not used.

**See Also**

*OraLob::enableBuffering*

**8.22.3 OraLob::enableBuffering**

```c
void enableBuffering();
```

**Description**

Enables LOB buffering for the input internal LOB locator. The next time data is read from or written to the LOB through the input locator, the LOB buffering subsystem is used.

**See Also**

*OraLob::enableBuffering*
8.22.4 OraLob::freeLob

void freeLob();

**Describe**

Call this member function to release Lob locator.

**See Also**

OraLob::allocLob, OraLob::init

OraLob | Classes | OCL | Index

8.22.5 OraLob::getOCILobLocator

OCILobLocator* getOCILobLocator();

**Return Value**

Handle of the Lob locator.

**Describe**

Call this member function to return handle of the Lob locator.

**See Also**

OraLob::getOCISvcCtx

OraLob | Classes | OCL | Index

8.22.6 OraLob::getOCISvcCtx

OCISvcCtx* getOCISvcCtx();

**Return Value**

Handle to the service context.

**Describe**

Call this member function to return handle to the service context.

**See Also**
8.22.7 *OraLob::init*

```cpp
void init();
```

**Description**
Call this member function to initialize empty LOB locator.

**See Also**
- *OraLob::allocLob*, *OraLob::freeLob*

8.22.8 *OraLob::isUnicode*

```cpp
bool isUnicode();
```

**Return Value**
True if the CLOB object is represented data as Unicode text on client side.

**Description**
This member function is used to verify whether OraLob object is in Unicode mode.

**See Also**
- *OraLob::setUnicode*

8.22.9 *OraLob::isInited*

```cpp
bool isInited();
```

**Return Value**
True if the CLOB, BLOB object is initialized.

**Description**

This member function is used to verify whether OraLob object is initialized or not. Result is true if object is initialized.

**See Also**

OraLob::allocLob, OraLob::freeLob

**8.22.10 OraLob::length**

```cpp
unsigned int length();
```

**Return Value**

Length of CLOB, BLOB object in bytes.

**Description**

Returns length in bytes for the CLOB, BLOB object.

**See Also**

OraLob::readLob, OraLob::writeLob

**8.22.11 OraLob::readLob**

```cpp
void readLob();
```

**Description**

Retrieves CLOB, BLOB object which is pointed to by the previously allocated Lob locator.

**See Also**

OraLob::allocLob, OraLob::writeLob
8.22.12  OraLob::setUnicode

void setUnicode(bool value);

**Parameters**

*value*

True if Unicode is enabled.

**Description**

Call this member function for newly created CLOB object to allow it store data as Unicode text on client side.

**See Also**

OraLob::isUnicode

OraLob | Classes | OCL | Index

8.22.13  OraLob::setOCISvcCtx

void setOCISvcCtx(OCISvcCtx *hSvcCtx);

**Parameters**

*hSvcCtx*

Pointer to the context handle.

**Description**

Call this member function to assign handle to the service context.

**See Also**

OraLob::getOCISvcCtx

OraLob | Classes | OCL | Index

8.22.14  OraLob::writeLob

void writeLob();

**Description**
rites CLOB, BLOB object to the database using Lob locator.

**See Also**

OraLob::allocLob, OraLob::readLob

OraLob | Classes | OCL | Index
8.23 OraMonitor

#include "oramonitor.h"

Members

static bool isActive();
static void start();
static void stop();

Description

This class performs application SQL activity. For detailed description see topic "Monitoring SQL statements".

See Also

OraConnection

8.23.1 OraMonitor::isActive

static bool isActive()

Return Value

True, if OraMonitor is active and may monitor SQL activity.

Description

Use this method to determine OraMonitor active state.

See Also

OraMonitor::start, OraMonitor::stop
8.23.2 OraMonitor::start

static void start();

**Description**

Use this method to start application monitoring.

**See Also**

OraMonitor::stop

OraMonitor | Classes | OCL | Index

---

8.23.3 OraMonitor::stop

static void stop();

**Description**

Use this method to stop application monitoring.

**See Also**

OraMonitor::start

OraMonitor | Classes | OCL | Index
# include "ora.h"

Members

Construct on

OraParam();
~OraParam();

Attributes

CRString name() const;
CRDataType dataType() const;
void setDataType(CRDataType value);
CRParamDirection direction() const;
void setDirection(CRParamDirection value);
int size() const;
void setSize(int value);

bool isNull(int index = 1) const;
void setNull(bool value);
void setNull(int index, bool value);

CRDate getDate(int index = 1) const;
void setDate(const CRDate& value);
void setDate(int index, const CRDate& value);

double getDouble(int index = 1) const;
void setDouble(double value);
void setDouble(int index, double value);

float getFloat(int index = 1) const;
void setFloat(float value);

int getInt(int index = 1) const;
void setInt(int value);
void setInt(int index, int value);

const char* getChars(int index = 1) const;
char* getChars(char* result);
char* getChars(int index, char* result);
void setChars(const char* value);
void setChars(int index, const char* value);

CRString getString(int index = 1) const;
void setString(const CRString& value);
void **setString**(int index, const CRString& value);

CRLob& *getLob*();
void **setLob**(CRLob& value);

OraLob& *getBlob*();
void **setBlob**(OraLob& value);
OraLob& *getBlob*();
void **setBlob**(CRLob& value);
OraLong& *getLong*();
void **setLong**(OraLong& value);
OraLong& *getLob*();
void **setLob**(OraLong& value);

OraFile& *getBFile*();
void **setBFile**(OraFile& value);

OraCursor& *getCursor*(() const;
void **setCursor**(OraCursor& value);

CRValue **getValue**(int index = 1) const;
void **setValue**(const CRValue& value);
void **setValue**(int index, const CRValue& value);

bool **isTable**() const;
void **setTable**(bool value);
int **tableLength**() const;
void **setTableLength**(int value);

**Operations**

void **assign**(const CRDate& value);
void **assign**(int value);
void **assign**(double value);
void **assign**(const char* value);
void **assign**(const CRString& value);
void **assign**(const CCRParam& value);

void **assign**(const CCRParam& value);
void **operator =**(const CRValue& value);

**Description**

This class represents single parameter that corresponds to placeholder in SQL statement. OraParam objects are created by OraCommand or OraRecordset object in according to SQL statement. OraParam has a lot of functions to access and write value of parameter for different types.

**Example**

```cpp
OraCommand sql;
sql.setSQL("INSERT INTO Scott.Dept VALUES (:DeptNo, :DName, :Loc)");
```
sql.param("DeptNo").setInt(50);
sql.param("DeptNo").setString("DEVELOPMENT");
sql.param("Loc").setString("LONDON");
sql.execute();

See Also

OraParams, OraCommand::param, OraParam::assign, OraParam::setDataType

8.24.1 OraParam::assign

void assign(const CRDate& value);
void assign(int value);
void assign(double value);
void assign(const char* value);
void assign(const CRString& value);
void assign(const OraParam& value);

Parameters

value
New value for the parameter.

Description

Call this member function to assign the value to the parameter.

Example

    cmd.param("DeptNo").assign(60);

See Also

OraParam::setDate, OraParam::setInt, OraParam::setDouble, OraParam::setString, OraParam::setNull

OraParam | Classes | OCL | Index

8.24.2 OraParam::dataType

typedef word CRDataType;

CRDataType dataType() const;

Return Value
Parameter type. May be one of the following values:
dtUnknown, dtString, dtInt32, dtInt64, dtInt, dtInteger, dtFloat, dtDouble,
dtDate, dtBlob, dtClob, dtLong, dtLongRaw.

**Descr pt on**

Returns the type of the parameter.

**Example**

```cpp
int value;

if (dataset.param(0).dataType == dtInt)
    value = param.field(0).getInt();
```

**See Also**

*OraParam::setDataType*, *Data types*

8.24.3 *OraParam::direction*

```cpp
enum CRParamDirection {pdUnknown, pdInput, pdOutput, pdInputOutput};
CRParamDirection direction() const;
```

**Return Value**

Parameter direction.

**Descr pt on**

Call this member function to get the parameter direction type if applicable.

**See Also**

*OraParam::setDirection*

8.24.4 *OraParam::getBFile*

```cpp
OraFile& getBFile();
```

**Return Value**
Reference to OraFile object.

**Descr pt on**

Call this member function to get OraFile object associated with the parameter.

**See Also**

OraFile, OraParam::setBFile, OraParam::dataType

8.24.5 **OraParam::getBlob**

OraLob& getBlob();

**Return Value**

OraLob object reference.

**Descr pt on**

Call this member function to retrieve the reference to OraLob object that represents value of BLOB parameter. Throws an exception if the parameter value type may not be cast to OraLob.

**See Also**

OraLob, OraParam::setBlob, OraParam::isNull

8.24.6 **OraParam::getWChars**

```cpp
const wchar_t* getWChars(int index = 1) const;
wchar t* getChars(wchar_t* result);
wchar t* getChars(int index, wchar t* result);
```

**Return Value**

Zero-terminated Unicode string. Do not delete this value.

**Parameters**

*index*

One-based item index into the array of parameter items.
result

The buffer where the result value is stored. Must be long enough to receive the whole string of chars.

**Description**

Call this member function to retrieve the value that was assigned to the output parameter, represented as a Unicode string. No conversion from non-Unicode string to a CR-String value are performed. Use this function for connections in Unicode mode only. Use `index` parameter to specify that either you are accessing values of a DML array or a PL/SQL table. See `setTable` method on the difference in behavior of OraParam objects when they are associated with Oracle container types.

**See Also**

OraParam::setWChars, OraParam::getWString, OraParam::isNull, OraParam::setTable, OraParam::setTableLength

OraParam | Classes | OCL | Index

### 8.24.7 OraParam::getChars

```cpp
const char* getChars(int index = 1) const;
char* getChars(char* result);
char* getChars(int index, char* result);
```

**Return Value**

Zero-terminated string. Do not delete this value.

**Parameters**

`index`

One-based item index into the array of parameter items.

`result`

The buffer where the result value is stored. Must be long enough to receive the whole string of chars.

**Description**

Call this member function to retrieve the value that was assigned to the output parameter, represented as a string. The value of the parameter will be converted to the string value if possible. Use `index` parameter to specify that either you are accessing values of a DML array or a PL/SQL table. See `setTable` method on the difference in behavior of OraParam objects when they are associated with Oracle container types.

**See Also**
8.24.8 OraParam::getClob

OraLob& getClob() const;

Return Value

OraLob value of the parameter.

Description

Call this member function to retrieve the reference to OraLob object that represents value of CLOB parameter. Throws an exception if the field value type may not be cast to OraLob.

See Also

OraLob, OraParam::setClob, OraParam::isNull

8.24.9 OraParam::getCursor

OraCursor& getCursor() const;

Return Value

Reference to OraCursor object.

Description

Call this member function to retrieve the reference to OraCursor object that represents cursor parameter value.

See Also

OraCursor, OraParam::setCursor, OraParam::isNull
8.24.10 OraParam::getDate

CRDate getDate(int index = 1) const;

Return Value
CRDate object.

Parameters

index
One-based item index number of array parameter.

Description
Call this member function to retrieve the value that was assigned to the output parameter, represented as CRDate.

See Also
CRDate, OraParam::setDate, OraParam::isNull

8.24.11 OraParam::getDouble

double getDouble(int index = 1) const;

Return Value
Double floating point number.

Parameters

index
One-based item index number of array parameter.

Description
Call this member function to retrieve the value that was assigned to the output parameter, represented as double. The value of the parameter will be converted to double value if possible.

See Also
OraParam::setDouble, OraParam::isNull
8.24.12 OraParam::getFloat

float getFloat(int index = 1) const;

**Return Value**
Floating point number.

**Parameters**

*index*
One-based item index number of array parameter.

**Descr pt on**
Call this member function to retrieve the value that was assigned to the output parameter, represented as float. The value of the parameter will be converted to float value if possible.

**See Also**

OraParam::setFloat, OraParam::isNull

OraParam | Classes | OCL | Index

8.24.13 OraParam::getInt

int getInt(int index = 1) const;

**Return Value**
Integer number.

**Parameters**

*index*
One-based item index number of array parameter.

**Descr pt on**
Call this member function to retrieve the value that was assigned to the output parameter, represented as int. The value of the parameter will be converted to int value if possible.

**See Also**

OraParam::setInt, OraParam::isNull
8.24.14 OraParam::getLob

CRLob& getLob() const;

**Return Value**

CRLob value of the parameter.

**Description**

Call this member function to retrieve the reference to CRLob object that represents value of large object parameter. You can use this method with BLOB, CLOB, LONG and LONG RAW data types. Throws an exception if the parameter value type may not be cast to CRLob.

**See Also**

CRLob, OraParam::setLob, OraParam::isNull

8.24.15 OraParam::getLong

OraLong& getLong() const;

**Return Value**

OraLong value of the parameter.

**Description**

Call this member function to retrieve the reference to OraLong object that represents LONG value of the parameter.

**See Also**

CRLob, OraParam::setLong, OraParam::isNull
8.24.16 OraParam::getLongRaw

OraLong& getLongRaw() const;

**Return Value**

OraLong value of the parameter.

**Description**

Call this member function to retrieve the reference to OraLong object that represents LONG RAW value of the parameter.

**See Also**

CRLob, OraParam::setLongRaw, OraParam::isNull

OraParam | Classes | OCL | Index

8.24.17 OraParam::getWString

CR String get WString(int index = 1);

**Return Value**

Unicode string representation of the param value.

**Parameters**

index

One-based item index number of array parameter.

**Description**

Call this member function to retrieve the value that was assigned to the output parameter, represented as Unicode string. No conversion from nonUnicode string to a CR String value are performed. Use this function for connections in Unicode mode only.

**See Also**

OraParam::set WString, OraParam::set Chars, OraParam::isNull

OraParam | Classes | OCL | Index
8.24.18  OraParam::getString

CRString getString(int index = 1);

Return Value

String representation of the param value.

Parameters

index
   One-based item index number of array parameter.

Description

Call this member function to retrieve the value that was assigned to the output parameter, represented as string. The value of the parameter will be converted to string value if possible.

See Also

OraParam::setString, OraParam::setChars, OraParam::isNull

8.24.19  OraParam::getValue

CRValue getValue(int index = 1) const;

Return Value

CRValue object.

Parameters

index
   One-based item index number of array parameter.

Description

Call this member function to retrieve the value that was assigned to the output parameter, represented as CRValue. The value of the parameter will be converted to CRValue value if possible.

See Also

OraParam::setValue, OraParam::isNull
8.24.20 OraParam::isNull

bool isNull(int index = 1) const;

**Return Value**

True if the parameter value is NULL.

**Parameters**

*index*

One-based item index number of array parameter.

**Description**

Call this member function to determinate whether the parameter is NULL or not.

**Example**

```c++
if (!cmd.param("DeptNo").isNull())
  deptNo = cmd.param("DeptNo").getInt();
```

**See Also**

OraParam::setNull

OraParam | Classes | OCL | Index

8.24.21 OraParam::isTable

bool isTable() const;

**Return Value**

True if the parameter is a PL/SQL table.

**Description**

Call this member function to verify whether the parameter is a PL/SQL table or not.

**Example**

```c++
if (cmd.param(1).isTable())
  for(int i = 1; i <= cmd.param(1).tableLength(); i++)
    cout << endl << cmd.param(1).getChars(i);
```
8.24.22 OraParam::name

CRString name() const;

**Return Value**

Name of the parameter.

**Description**

Returns name of the parameter.

**See Also**

CRField::dataType

OraParam | Classes | OCL | Index

8.24.23 OraParam::operator =

void operator =(const CRValue& value);

**Parameters**

value

CRValue object to be assigned to this parameter.

**Description**

Call this member function to assign the CRValue value to the parameter. If data type of the parameter is dtUnknown this operator sets it to the most matching type, otherwise converts the value to the previously assigned type if possible. Use **setDataType** member function to change data type of the parameter.

**See Also**

OraParam::setValue, OraParam::setDataType, OraParam::setNull

OraParam | Classes | OCL | Index
**8.24.24 OraParam::OraParam**

OraParam();

**Description**

This member function constructs a new OraParam object.

**See Also**

OraParam::assign, OraParam::setDataType

---

**8.24.25 OraParam::setBFile**

void setBFile(OraFile& value);

**Parameters**

value

New OraFile value.

**Description**

Call this member function to assign OraFile object to the parameter. If data type of the parameter is dtUnknown setBFile member function sets it to dtBFile. Use setDataType member function to change data type of the parameter.

**See Also**

OraFile, OraParam::getBFile, OraParam::setDataType, OraParam::setNull

---

**8.24.26 OraParam::setBlob**

void setBlob(OraLob& value);

**Parameters**

value

OraLob object.

**Description**
Call this member function to assign OraLob object to BLOB parameter. If data type of the parameter is dtUnknown `setBlob` member function sets it to dtBlob. Use `setDataType` member function to change data type of the parameter.

**See Also**

OraLob, OraParam::getBlob, OraParam::setDataType, OraParam::isNull

---

### 8.24.27 OraParam::setWChars

```cpp
void setChars(const wchar_t* value);
void setChars(int index, const wchar_t* value);
```

**Parameters**

- `index`
  - One-based item index into the array of parameter items.
- `value`
  - Zero-terminated buffer with the new Unicode string value.

**Description**

Call this member function to assign the string value to the parameter. If data type of the parameter is dtUnknown `setWChars` member function sets it to dtString, otherwise it casts the value to the current parameter type if possible. Use `setDataType` member function to change data type of the parameter.

Use `index` parameter to specify that either you are accessing values of a DML array or a PL/SQL table. See `setTable` method on the difference in behavior of OraParam objects when they are associated with Oracle container types.

**See Also**

OraParam::getChars, OraParam::set String, OraParam::setDataType, OraParam::isNull, OraParam::setTable, OraParam::setTableLength

---

### 8.24.28 OraParam::setChars

```cpp
void setChars(const char* value);
void setChars(int index, const char* value);
```

**Parameters**
index
   One-based item index into the array of parameter items.

value
   Zero-terminated buffer with the new string value.

**Description**

Call this member function to assign the string value to the parameter. If data type of the parameter is dtUnknown `setChars` member function sets it to dtString, otherwise it casts the value to the current parameter type if possible. Use `setDataType` member function to change data type of the parameter. Use `index` parameter to specify that either you are accessing values of a DML array or a PL/SQL table. See `setTable` method on the difference in behavior of OraParam objects when they are associated with Oracle container types.

**See Also**

OraParam::getChars, OraParam::setString, OraParam::setDataType, OraParam::isNull, OraParam::setTable, OraParam::setTableLength

8.24.29 **OraParam::setClob**

```cpp
void setClob(OraLob& value);
```

**Parameters**

value
   OraLob object.

**Description**

Call this member function to assign OraLob object to CLOB parameter. If data type of the parameter is dtUnknown `setClob` member function sets it to dtClob. Use `setDataType` member function to change data type of the parameter.

**See Also**

OraLob, OraParam::getClob, OraParam::setDataType, OraParam::isNull

8.24.30 **OraParam::setCursor**

```cpp
void setCursor(OraCursor& value);
```
Parameters

value
   OraCursor object.

Description

Call this member function to assign OraCursor object to the parameter. If data type of
the parameter is dtUnknown setCursor member function sets it to dtCursor. Use
setDataType member function to change data type of the parameter.

See Also

OraCursor, OraParam::getCursor, OraParam::setDataType, OraParam::setNull

OraParam | Classes | OCL | Index

8.24.31 OraParam::setDataType

typedef word CRDataType;
void setDataType(CRDataType value);

Parameters

value
   Parameter type. May be one of the following values:
   dtUnknown, dtString, dtInt32, dtInt64, dtInt, dtInteger, dtFloat, dtDouble,
   dtDate, dtBlob, dtClob, dtLong, dtLongRaw.

Description

Call this member function to set data type of the parameter.

See Also

OraParam::dataType

OraParam | Classes | OCL | Index

8.24.32 OraParam::setDate

void setDate(const CRDate& value);
void setDate(int index, const CRDate& value);

Parameters
**index**

One-based item index number of array parameter.

**value**

New CRDate value.

### **Description**

Call this member function to assign the CRDate value to the parameter. If data type of the parameter is `dtUnknown` `setDate` member function sets it to `dtDate`, otherwise converts the value to assigned before type if possible. Use `setDataType` member function to change data type of the parameter.

### **Example**

```cpp
    cmd.param("BirthDate").setDate(CRDate(1975,7,9));
```

### **See Also**

`CRDate`, `OraParam::getDate`, `OraParam::setDataType`, `OraParam::setNull`

`OraParam` | **Classes** | **OCL** | **Index**

---

### **8.24.33 OraParam::setDirection**

```cpp
    void setDirection(CRParamDirection value);
```

### **Parameters**

**value**

Type of the parameter. Must be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th><strong>Descr pt on</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pdUnknown</code></td>
<td>Unknown or undetermined. Before executing a stored procedure, the application must set parameters of this type to another one.</td>
</tr>
<tr>
<td><code>pdInput</code></td>
<td>Used to input a field value. Identifies a parameter used to pass values to a stored procedure for processing.</td>
</tr>
<tr>
<td><code>pdOutput</code></td>
<td>Used to output a field value. Identifies a parameter used by a stored procedure to return values to an application.</td>
</tr>
<tr>
<td><code>pdInputOutput</code></td>
<td>Used for both input and output.</td>
</tr>
</tbody>
</table>

### **Description**

Call this member function to set the type of use for the parameter.

### **See Also**

`OraParam::direction`
8.24.34 OraParam::setDouble

void setDouble(double value);
void setDouble(int index, double value);

Parameters

index
One-based item index number of array parameter.

value
New double value.

Description

Call this member function to assign the double value to the parameter. If data type of the parameter is dtUnknown setDouble member function sets it to dtDouble, otherwise converts the value to assigned before type if possible. Use setDataType member function to change data type of the parameter.

Example

    cmd.param("DeptNo").setDouble(60);

See Also

OraParam::getDouble, OraParam::setDataType, OraParam::setNull

8.24.35 OraParam::setFloat

void setFloat(float value);

Parameters

value
New float value.

Description

Call this member function to assign the float value to the parameter. If data type of the parameter is dtUnknown setFloat member function sets it to dtFloat, otherwise converts the value to assigned before type if possible. Use setDataType member function to change data type of the parameter.
Example

    cmd.param("DeptNo").setFloat(60);

See Also

OraParam::getFloat, OraParam::setDataType, OraParam::setNull

OraParam | Classes | OCL | Index

8.24.36 OraParam::setInt

void setInt(int value);
void setInt(int index, int value);

Parameters

index
    One-based item index number of array parameter.

value
    New integer value.

Description

Call this member function to assign the int value to the parameter. If data type of the
parameter is dtUnknown setInt member function sets it to dtInt, otherwise converts the
value to assigned before type if possible. Use setDataType member function to change
data type of the parameter.

Example

    cmd.param("DeptNo").setInt(60);

See Also

OraParam::getInt, OraParam::setDataType, OraParam::setNull

OraParam | Classes | OCL | Index

8.24.37 OraParam::setLob

void setLob(CRLob& value);

Parameters

value
New CRLob value.

**Describe on**

Call this member function to assign CRLob object to the parameter. Use `setDataType` member function to change data type of the parameter.

**See Also**

CRLob, `OraParam::setLob`, `OraParam::setDataType`, `OraParam::setNull`

`OraParam` | Classes | OCL | Index

### 8.24.38 OraParam::setLong

```cpp
void setLong(OraLong& value);
```

**Parameters**

`value`

OraLong object.

**Describe on**

Call this member function to assign OraLong object to the parameter. If data type of the parameter is `dtUnknown` `setLong` member function sets it to `dtLong`. Use `setDataType` member function to change data type of the parameter.

**See Also**

`OraParam::getLong`, `OraParam::setDataType`, `OraParam::setNull`

`OraParam` | Classes | OCL | Index

### 8.24.39 OraParam::setLongRaw

```cpp
void setLongRaw(OraLong& value);
```

**Parameters**

`value`

OraLong object.

**Describe on**

Call this member function to assign OraLong object to the parameter. If data type of the parameter is `dtUnknown` `setLongRaw` member function sets it to `dtLongRaw`. Use `setDataType` member function to change data type of the parameter.
8.24.40 OraParam::setNull

void setNull(int index);

Parameters

index
One-based item index number of array parameter.

Description

Call this member function to assign a NULL value to the parameter.

See Also

OraParam::isNull

8.24.41 OraParam::setSize

void setSize(int value);

Parameters

value
New DML array or PL/SQL table size.

Description

The new size of the DML array or PL/SQL table.

See Also

OraParam::size
8.24.42 OraParam::setWString

void setWString(const CRWString& value);
void setWString(int index, const CRWString& value);

Parameters

index
One-based item index number of array parameter.

value
New Unicode string value.

Description
Call this member function to assign the Unicode string value to the parameter. If data type of the parameter is dtUnknown setWString member function sets it to dtString, otherwise casts the value to the current parameter type if possible. Use setDataType member function to change data type of the parameter. Use this function for connections in Unicode mode only.

Example

cmd.param("DName").setWString(L"Research");

See Also
CRString, OraParam::getString, OraParam::setDataType, OraParam::setNull

OraParam | Classes | OCL | Index

8.24.43 OraParam::setString

void setString(const CRString& value);
void setString(int index, const CRString& value);

Parameters

index
One-based item index number of array parameter.

value
New string value.

Description
Call this member function to assign the string value to the parameter. If data type of the parameter is dtUnknown setString member function sets it to dtString, otherwise casts
the value to the current parameter type if possible. Use **setDataType** member function to change data type of the parameter.

**Example**

```cpp
    cmd.param("DName").setString("Research");
```

**See Also**

[CRString](#), [OraParam::getString](#), [OraParam::setDataType](#), [OraParam::setNull](#)

8.24.44 **OraParam::setTable**

```cpp
    void setTable(bool value);
```

**Parameters**

`value`

True to specify that this parameter object will be used for the PL/SQL table or DML array operations.

**Description**

Call this member function to specify whether the parameter represents a PL/SQL table or a DML array or not.

**See Also**

[OraParam::isTable](#)

8.24.45 **OraParam::setTableLength**

```cpp
    void setTableLength(int value);
```

**Parameters**

`value`

Size of the PL/SQL table or DML array. Must be greater or equal to one.

**Description**

Call this member function to specify the new size of the PL/SQL table or DML array that is associated with this parameter.
8.24.46 **OraParam::setValue**

```cpp
type void setValue(const CRValue& value);
type void setValue(int index, const CRValue& value);
```

**Parameters**

- **index**
  - One-based item index number of array parameter.

- **value**
  - CRValue object.

**Description**

Call this member function to assign the CRValue value to the parameter. If data type of the parameter is dtUnknown `setValue` member function sets it to most matching type, otherwise converts the value to assigned before type if possible. Use `setDataType` member function to change data type of the parameter.

**See Also**

- `OraParam::getValue`
- `OraParam::setDataType`
- `OraParam::setNull`

8.24.47 **OraParam::size**

```cpp
int size() const;
```

**Return Value**

Size of the parameter value.

**Description**

This method returns the size of the value associated with this parameter.

**See Also**

- `OraParam::setSize`
8.24.48 OraParam::tableLength

```cpp
int tableLength();
```

**Return Value**

Size of the PL/SQL table or DML array.

**Description**

Call this member function to get the size of the PL/SQL table or DML array that is associated with this parameter.

**See Also**

OraParam::setTableLength
8.25 OraParams

#include "ora.h"

Members

Attributes

int count() const;

Operations

OraParam* findItem(int index) const;
OraParam* findItem(const CRString& name) const;
OraParam& item(int index) const;
OraParam& item(const CRString& name) const;

OraParam& operator[](int index) const;
OraParam& operator[](const CRString& name) const;

Description

OraParams serves to store and manage collection of OraParam objects. It is used by OraCommand and OraRecordset objects.

See Also

OraParam, OraCommand::params

Classes | OCL | Index

8.25.1 OraParams::count

int count() const;

Return Value

The number of parameters in the parameters object.

Description

Call this member function to determine the number of parameters referenced by the OraParams object.

See Also
8.25.2 *OraParams::*findItem

OraParam* findItem(int index) const;
OraParam* findItem(const CRString& name) const;

**Return Value**
The pointer to OraParam object for the specified parameter.

**Parameters**

*index*
- Zero-based index number of the parameter.

*name*
- Name of the parameter.

**Description**
Call this member function to determine if a specified parameter is referenced in the OraParams object. If *findItem* finds a parameter with a matching name, it returns the OraParam object for the specified parameter. Otherwise it returns NULL. *findItem* is a useful function to call prior to calling other OraParams functions, such as *item*, that require a valid parameter name.

**Example**

```cpp
CRString address;
if (cmd.params.findItem("Address"))
    address = cmd.params.findItem("Address")->getString();
```

**See Also**
- OraParam, OraParams::*item*, OraCommand::*findParam*

8.25.3 *OraParams::*item

OraParam& item(int index) const;
OraParam& item(const CRString& name) const;

**Return Value**
The reference to OraParam object for the specified parameter.

**Parameters**

- **index**
  Zero-based index number of the parameter.

- **name**
  Name of the parameter.

**Description**

Call this member function to retrieve parameter information for a parameter when its name or its index is known. If the specified parameter is not found the exception function throws the exception.

**Example**

```cpp
CRString address;  // get value of the parameter with the name address
address = cmd1.params.item("Address").getString();  // set string to the parameter with index 0
cmd2.params.item(0).assign(Address);
```

**See Also**

OraParam, OraParams::findItem, OraParams::operator[], OraCommand::param

---

**8.25.4 OraParams::operator []**

```cpp
OraParam& operator [](int index) const;
OraParam& operator [](const CRString& name) const;
```

**Return Value**

The reference to OraParam object for the specified parameter.

**Parameters**

- **index**
  Zero-based index number of the parameter.

- **name**
  Name of the parameter.

**Description**

These operators allow to retrieve parameter information for a parameter when its name or its index is known. If the specified parameter is not found they throw the exception.
Example

```c++
CRString address;
// get value of the field with the name address
address = cmd1.params["Address"].getString();
// set string to the field with index 0
cmd2.params[0].assign(address);
```

See Also

- OraParam, OraParams::item, OraCommand::operator
- OraParams | Classes | OCL | Index
# 8.26 OraQuery

```c
#include "ora.h"
```

## Hierarchy

```
CRDataset
  | CRMemDataset
  | OraRecordset  OraCommand
    |      OraQuery
```

## Members

### Construction

- `OraQuery();`
- `OraQuery(OraConnection& connection);`
- `OraQuery(OraConnection& connection, const CRString& sql)`
- `virtual ~OraQuery();`

### Preparation

- `void prepare();`
- `void unprepare();`

### Executing

- `int execute();`

### Attributes

- `bool isActive() const;`

### Misc

- `OraField& operator[](int index) const;`
- `OraField& operator[](const CRString& name) const;`

## Description

OraQuery implements methods to execute any SQL, PL/SQL statement or call stored procedure, store and manage returned rows, provide access to field values and navigation through records. Use `setConnection` to link to the connection object and `setCommandText` function to set SQL statement. Call `open` to retrieve records from the database. To access the fields
use **field** function. You can use navigational functions **next, prev, first, last** and **move** and **sEOF, sBOF, recordNo** member functions to check current record position.

**See Also**

OraRecordset, OraCommand, OraConnection, Working with dataset

---

### 8.26.1 OraQuery::close

```c
void close();
```

**Description**

Call this member function to close the query. You can call **open** method again after calling **close**. This lets you to reuse the query object. The alternative is to call **refresh**.

**See Also**

OraQuery::open, CRDataset::refresh

---

### 8.26.2 OraQuery::execute

```c
int execute();
```

**Return Value**

The actual number of rows processed. In case of SELECT statement this number equals to the number of fetched rows.

**Description**

Call this member function to execute an SQL statement on the server. **execute** function prepares SQL statement if **prepare** function wasn't called before. Calls **open** member function to open the dataset if SQL statement returns rows. Use **param** function to access to parameter objects to write and read their values.

**Example**

```c
OraConnection connection("scott/tiger");
OraQuery query(connection, "SELECT Count(*) FROM DeptNo");
int count;

connection.open();
query.execute();
count = query[0].getInt();
```
See Also

OraCommand::setConnection, OraCommand::setSQL, OraQuery::open, OraQuery::prepare, OraCommand::isQuery, OraCommand::param, OraCommand::createProcCall

OraQuery | Classes | OCL | Index

### 8.26.3 OraQuery::isActive

```cpp
bool isActive() const;
```

**Return Value**

True if the query object's **open** or **refresh** member function has previously been called and the query has not been closed, otherwise false.

**Description**

Call this member function to determine if the query is already open.

**See Also**

OraQuery::open, OraQuery::close, CRDataset::refresh

OraQuery | Classes | OCL | Index

### 8.26.4 OraQuery::open

```cpp
void open();
```

**Description**

Call this member function to open the dataset object. After calling this function **isActive** returns true. You should assign connection object and SQL text by **setConnect on** and **setSQL** member functions before opening.

**Example**

```cpp
OraQuery query;

query.setConnection(connection);
query.setCommandText("SELECT * FROM Dept");
query.open();
while (!query.isEOF()) {
    . . .
    query.next();
}
```
query.close();

See Also

OraQuery::close, OraQuery::execute

OraQuery | Classes | OCL | Index

8.26.5 OraQuery::operator []

OraField& operator (int index) const;
OraField& operator (const CRString& name) const;

Return Value

The reference to OraField object for the specified field.

Parameters

index
Zero-based index number of the field.

ame
The name of the field.

Description

These operators allow to retrieve field information for a field when its name or its index is known. If the specified field is not found they throw the exception.

Example

CRString address;
// get value of the field with the name address
address = query1["Address"].getString();
// set string to the field with index 0
query2[0].assign(address);

See Also

OraField, OraRecordset::field

OraQuery | Classes | OCL | Index

8.26.6 OraQuery::OraQuery

OraQuery();
OraQuery(OraConnection& connection);
OraQuery(OraConnection& connection, const CRString& sql);
**Parameters**

connect on
Reference to the connection object in which SQL will be executed.

sql
The SQL statement.

**Description**

This member function constructs a new OraQuery object.

**See Also**

OraQuery::open, OraQuery::prepare

OraQuery | Classes | OCL | Index

---

**8.26.7 OraQuery::prepare**

void prepare();

**Description**

Call this member function to prepare the query. Calling `prepare` before executing a query improves application performance. `prepare` function is called automatically by `open` or `execute` member function if the dataset is not prepared. After execution, OraRecordset unprepares the query. When a query will be executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.

To learn whether the dataset is prepared or not use **sPrepared** member function. The `unprepare` method unprepares a query.

**Example**

```c++
query.setConnection(connection);
query.setCommandText("SELECT * FROM Dept WHERE DeptNo = :DeptNo");
query.prepare();
for (int i = 60; i < 70; i++) {
    query.param("DeptNo").setInt(i);
    query.open();
    ...;
    query.close();
}query.unprepare();
```

**See Also**

OraCommand::isPrepared, OraQuery::unprepare
8.26.8 OraQuery::unprepare

void unprepare();

**Description**

Call this member function to free the resources allocated for a previously prepared SQL statement on the server and client side.

**See Also**

OraQuery::prepare, OraCommand::isPrepared
8.27 OraRecordset

#include "ora.h"

Hierarchy

CRDataset
  | CRMemDataset
  | OraRecordset

Members

Construct on

virtual ~OraRecordset();

Fetch

void fetchAll();
bool isActive() const;
int fetchRows() const;
void setFetchRows(int value);

bool isCached() const;
void setCached(bool value);

Fields

OraField& field(int index) const;
OraField& field(const CRString& name) const;
OraField* findField(int index) const;
OraField* findField(const CRString& name) const;

OraField& operator[](int index) const; *
OraField& operator[](const CRString& name) const; *

Description

OraRecordset implements methods that manipulate with individual fields from a set of rows. You may not instantiate an object of this class, though you may declare a pointer to the OraRecordset class and assign it with a recordset returned by a OraCommand::execute method.

OraRecordset may allocate space to store all returned records or only one of them which is being worked with. Use setCached method to switch between either cached mode or noncached. Cached mode allows you to navigate backward in your record set but requires additional memory to store rows.
Individual fields are accessible through the `field` function. You can use navigational functions `next`, `prev`, `first`, `last`, `move` and `sEOF`, `sBOF`, `recordNo` member functions to check current record position.

**See Also**

`CRDataset`, `CRMemDataset`, `OraCommand`, `OraConnection`, `Working with dataset`

### 8.27.1 `OraRecordset::fetchAll`

```cpp
void fetchAll();
```

**Description**

Call this member function to retrieve all the records returned by query from server. The recordset object must be opened before. Use this member function if recordset is opened in cached mode. To switch the dataset to cached mode use `setCached` member function.

**Example**

```cpp
OraConnection connection("scott/tiger");
OraRecordset rs(connection, "SELECT * FROM DeptNo");

    connection.open();
    rs.setCached(true); // switch to cached mode
    rs.open();
    rs.fetchAll;
```

**See Also**

`CRDataset::open`, `OraRecordset::setCached`

### 8.27.2 `OraRecordset::fetchRows`

```cpp
int fetchRows() const;
```

**Return Value**

Number of rows to be fetched.

**Description**

Returns the value set previously by `setFetchRows` member function. By default, returns 25.
8.27.3 OraRecordset::field

OraField& field(int index) const;
OraField& field(const CRString& name) const;

Return Value

The reference to OraField object for the specified field.

Parameters

index
Zero-based index number of the field.

name
The name of the field.

Description

Call this member function to retrieve field information for a field when its name or its index is known. If the specified field is not found field function throws the exception.

Example

CRString address;
// get value of the field with the name address
address = rs1.field("Address").getString();
// set string to the field with index 0
rs2.field(0).assign(address);

See Also

OraField, OraRecordset::findFirst, CRDataset::fields, OraRecordset::operator___

8.27.4 OraRecordset::findFirst

OraField* findField(int index) const;
OraField* findField(const CRString& name) const;

Return Value


The pointer to OraField object for the specified field.

**Parameters**

*index*

Zero-based index number of the field.

*name*

Name of the field.

**Description**

Call this member function to determine if a specified field component exists in the dataset. If `findField` finds a field with a matching name, it returns the OraField object for the specified field. Otherwise it returns NULL. `findField` is a useful function to call prior to calling other dataset functions, such as `field`, that require a valid field name as a parameter.

**Example**

```cpp
CRString address;
if (rs.findField("Address") != NULL)
    address = rs.findField("Address")->getString();
```

**See Also**

OraField, OraRecordset::field

---

**8.27.5 OraRecordset::isActive**

```cpp
bool isActive() const;
```

**Return Value**

True if the dataset object's `open` or `refresh` member function has previously been called and the dataset has not been closed, otherwise false.

**Description**

Call this member function to determine if the dataset is already open.

**See Also**

CRDataset::open, CRDataset::close, CRDataset::refresh

---
8.27.6 OraRecordset::isCached

bool isCached() const;

Return Value

True if the recordset is cached locally.

Description

Returns the value set previously by setCached member function. By default, returns false.

See Also

OraRecordset::setCached

OraRecordset | Classes | OCL | Index

8.27.7 OraRecordset::operator []

OraField& operator (int index) const;
OraField& operator (const CRString& name) const;

Return Value

The reference to OraField object for the specified field.

Parameters

index
   Zero-based index number of the field.

name
   The name of the field.

Description

These operators allow to retrieve field information for a field when its name or its index is known. If the specified field is not found they throw the exception.

Example

CRString address;
   // get value of the field with the name address
   address = rs1["Address"].getString();
   // set string to the field with index 0
   rs2[0].assign(address);
8.27.8 OraRecordset::setCached

```cpp
void setCached(bool value);
```

**Parameters**

- **value**
  - True to allow local cache for the recordset.

**Description**

Call this member function to control whether or not OraRecordset object caches fetched records. Caching records allows to use previously fetched records and move the current record pointer backward by `prev` member function. By default, OraRecordset doesn't cache records.

**Example**

```cpp
OraConnection connection("scott/tiger");
OraRecordset rs(connection, "SELECT * FROM Dept");

connection.open();
dataset.setCached(true);
rs.open();
rs.last();
while (!rs.isBOF())
  rs.prev();
```

See Also

- OraRecordset::isCached, OraRecordset::fetchAll

8.27.9 OraRecordset::setFetchRows

```cpp
void setFetchRows(int value);
```

**Parameters**

- **value**
  - Number of records to cache locally.
**Description**

Call this member function to set number of records that will be transferred across the network at once. This value can have a great influence on speed of fetching. If the query contains a LONG or LONG RAW field, OraRecordset fetches records one by one.

**See Also**

- `OraRecordset::fetchRows`
- `OraRecordset` | `Classes` | `OCL` | `Index`
8.28 OraScript

```c
#include "orascript.h"
```

**Members**

**Construct on**

```c
OraScript();
OraScript(OraConnection* conn);
```

```c
~OraScript();
```

**Attributes**

```c
OraConnection* connection() const;
void setConnection(OraConnection* conn);
```

```c
CRString SQL() const;
void setSQL(const CRString& sql);
```

```c
bool breakOnError() const;
void setBreakOnError(bool value);
```

```c
int errorOffset() const;
```

**Methods**

```c
void toBegin();
bool execute(bool continueExec = false);
```

**Description**

OraScript implements methods to execute a series of SQL statements separated by special symbols, like SQL*Plus scripts. Often it is necessary to execute several SQL statements one by one. Sometimes it can be performed by anonymous PL/SQL blocks. But unfortunately it doesn't work in all cases. For example DDL statements cannot be used in PL/SQL. Another way is using a lot of objects such as OraCommand. Usually it isn't a good solution. With only one OraScript object you can execute several SQL statements as one. This sequence of statements is named script. To separate single statements use semicolon (;) or slash (/) and for PL/SQL only slash. Note slash must be the first character in the line.

**See Also**

OraCommand

Classes | OCL | Index
8.28.1 OraScript::breakOnError

bool breakOnError() const;

Return Value
True, if OraScript throws an exception when error occurred, otherwise false.

Description
Use this member function to determine OraScript behaviour when error occurred.

See Also
OraScript::setBreakOnError

8.28.2 OraScript::connection

OraConnection* connection();

Return Value
OraConnection object used by this OraScript.

Description
OraScript uses OraConnection object during the script execution. OraScript object requires open connection. If connection is closed an exception will be thrown.

See Also
OraScript::setConnection

8.28.3 OraScript::errorOffset

int errorOffset() const;

Return Value
Error symbol position.
**Description**

Use this member function to obtain offset of error inside SQL script if Execute method raised exception.

**See Also**

OraScript::breakOnError, OraScript::setBreakOnError

OraScript | Classes | OCL | Index

---

**8.28.4 OraScript::execute**

```cpp
bool execute(bool continueExec = false);
```

**Return Value**

True, if script was executed without errors, otherwise false.

**Parameters**

`continueExec`

If false and during script execution error occurred, then script processed from the beginning, else if `continueExec` is true, after error occurred script execution continued from the next SQL statement.

---

**Description**

Use this member function to execute SQL script. Before calling this method SQL and connect on attributes must be specified. If `continueExec` parameter is false and error occurred script execution will be processed from the beginning. If true script execution will be continued from the next SQL statement after error statement.

**See Also**

OraScript::setBreakOnError, OraScript::setSQL

OraScript | Classes | OCL | Index

---

**8.28.5 OraScript::OraScript**

```cpp
OraScript();
OraScript(OraConnection* conn);
```

**Parameters**

`conn`
OraConnection object used by this OraScript.

**Description**

This member function constructs a new OraScript object with connection object.

**See Also**

OraScript::setSQL, OraScript::execute

---

### 8.28.6 OraScript::setBreakOnError

void setBreakOnError(bool value);

**Parameters**

*value*

True, if OraScript throws an exception when error occurred, otherwise false.

**Description**

Use this member function to specify OraScript behaviour when error occurred.

**See Also**

OraScript::breakOnError

---

### 8.28.7 OraScript::setConnection

void setConnection(OraConnection& conn);
void setConnection(OraConnection* conn);

**Parameters**

*conn*

OraConnection object used by this OraScript.

**Description**

OraScript uses **OraConnection** object during the script execution. OraScript object requires open connection. If connection is closed an exception will be thrown.
See Also

OraScript::connection

OraScript | Classes | OCL | Index

8.28.8 OraScript::setSQL

void setSQL(const CRString& sql);

Parameters

sql
  Script text that will be executed.

Description

Use this member function to provide the sequence of SQL statements that OraScript executes when `execute` method is called. Use semicolon (`;`) or slash (`/`) symbol to separate single statements in script. You must specify script text before calling `execute` method.

See Also

OraScript::SQL

OraScript | Classes | OCL | Index

8.28.9 OraScript::SQL

CRString SQL() const;

Return Value

Script text to be executed.

Description

Use this member function to obtain the text of SQL script. Script text must be specified before calling `execute` method.

See Also

OraScript::setSQL

OraScript | Classes | OCL | Index
8.28.10 OraScript::tobegin

```cpp
void toBegin();
```

**Description**

Call this member function to start script execution from the beginning.

**See Also**

OraScript::execute

OraScript | Classes | OCL | Index
# include "ora.h"

Members

Construct on

OraTimeStamp();
OraTimeStamp(const OraTimeStamp& src);
OraTimeStamp(short year, unsigned char month, unsigned char day,
    unsigned char hour, unsigned char min, unsigned char sec, unsigned fsec,
    const CRString& timeZone = "");

OraTimeStamp& operator=(const OraTimeStamp& r);

~OraTimeStamp();

Attributes

OraDataType timeStreamType() const;
void setStreamType(OraDataType type);

void getDate(short& year, unsigned char& month, unsigned char& day) const;
void setDate(short year, unsigned char month, unsigned char day);

void getTime(unsigned char& hour, unsigned char& min,
    unsigned char& sec, unsigned& fsec) const;
void setTime(unsigned char hour, unsigned char min,
    unsigned char sec, unsigned fsec);

void timeZoneOffset(unsigned char& hour, signed char& min) const;
void setTimeZoneOffset(unsigned char hour, signed char min);

CRString timeZoneName() const;

void setSysTimeStream();

bool isStream() const;
void setStream();

Methods

CRString toString(const CRString& format = ",", unsigned char fsprecision = 6) const;
void fromString(const CRString& value, const CRString& format = ");

int compare(const OraTimeStamp& value) const;
OraTimeStamp\& operator+(const OraInterval\& value);
OraTimeStamp\& operator-(const OraInterval\& value);

OraInterval operator-(const OraTimeStamp \&l, const OraTimeStamp \&r);

bool operator==(const OraTimeStamp \&l, const OraTimeStamp \&r);
bool operator>(const OraTimeStamp \&l, const OraTimeStamp \&r);
bool operator<(const OraTimeStamp \&l, const OraTimeStamp \&r);
bool operator!= (const OraTimeStamp \&l, const OraTimeStamp \&r);
bool operator>= (const OraTimeStamp \&l, const OraTimeStamp \&r);
bool operator<= (const OraTimeStamp \&l, const OraTimeStamp \&r);

**Description**

This class represents Oracle data types such as TIMESTAMP, TIMESTAMP WITH TIMEZONE, TIMESTAMP WITH LOCAL TIME ZONE. When you create OraTimeStamp object with default constructor, object state is uninitialized and TIMESTAMP type is unknown. Before you can do anything with such object you must call setTimeStampType function to set up OraTypeStamp type and initialize it or assign value to this object from other OraTimeStamp object with operator=:

```cpp
// Create object with default constructor
OraTimeStamp timeStampLTZ;
// Initialize object and set up data type
timeStampLTZ.setTimeStampType(dtTimeStampLTZ);
```

Or

```cpp
// Create object with default constructor
OraTimeStamp timeStampLTZ1;
// Create object and initialize it
OraTimeStamp timeStampLTZ(2002, 6, 28, 13, 45, 30, 456);
// Initialize object and set up data type with assigning
timeStampLTZ1 = timeStampLTZ2;
```

OraTimeStamp object can represent one of the three Oracle types:
- TIMESTAMP - corresponds to dtTimeStamp constant;
- TIMESTAMP WITH TIMEZONE - corresponds to dtTimeStampTZ constant;
- TIMESTAMP WITH LOCAL TIME ZONE - corresponds to dtTimeStampLTZ constant.

When you fetch data from Oracle, fields data types in OraRecordset are initialized automatically and you don't need set type for OraTimeStamp fields manually. But when you work with parameters you must set appropriate data type for OraTimeStamp parameter.

```cpp
OraRecordset\& rs = cmd.executeQuery();
while (!rs.isEOF()) {
    ...
    // Get TIMESTAMP field value as string
    std::cout << rs.field("TIMESTAMP FIELD").getTimeStamp().getString() << std::endl;
    ...
```
rs.next();
}
...

// Set value from string with format parameter
cmd.param("TIMESTAMP TZ FIELD").getTimeStamp().setString("01:16:17.12
04/03/1825",
"hh:mi:ssxff dd/mm/yyyy");
...
cmd.execute();

To determine is OraTimeStamp object is NULL you can use **isNull** member function. Call **setNull** member function to set OraTimeStamp object value to NULL.

**Classes** | **OCL** | **Index**

### 8.29.1 OraTimeStamp::compare

```cpp
type compare(const OraTimeStamp& value) const;
```

**Return Value**

-1, if current object less than value.
0, if current object and value are equal.
1, if current object greater than value.

**Parameters**

*value*  
Object that will be compared with the current object.

**Description**

Use this member function to compare current object with values.

**See Also**

OraTimeStamp::operator==, OraTimeStmp::operator!=

**OraTimeStamp** | **Classes** | **OCL** | **Index**

### 8.29.2 OraTimeStamp::getDate

```cpp
void getDate(short& year, unsigned char& month, unsigned char& day) const;
```

**Parameters**

*year*  

Year value for OraTimeStamp object.

month
   Month value for OraTimeStamp object.

day
   Day value for OraTimeStamp object.

Description
Use this member function to return date part of the object.

Example

...  
unsigned char hour, minute, second, fsecond;  
timeStamp.getTime(hour, minute, second, fsecond);  
std::cout << "Hour: " << hour  
   << "Minute: " << minute  
   << "Second: " << second  
   << "Frac Second: " << fsecond << std::endl;  
...

See Also
OraTimeStamp::getTime, OraTimeStamp::setDate

OraTimeStamp | Classes | OCL | Index

8.29.3 OraTimeStamp::toString

CRString toString(const CRString& format = "", unsigned char fsprecision = 6) const;

Return Value
String that contains OraTimeStamp object value.

Parameters

format
   Conversion format.

fsprec s on
   Specifies the precision in which the fractional seconds will be returned.

Description
Use this member function to obtain string representation of OraTimeStamp object. To
learn more about conversion format string refer to Oracle documentation.

See Also

OraTimeStamp::fromString

OraTimeStamp | Classes | OCL | Index

8.29.4 OraTimeStamp::getTime

void getTime(unsigned char& hour, unsigned char& min, unsigned char& sec, unsigned& fsec) const;

Parameters

*hour*
   Hour value for OraTimeStam object.

*min*
   Minute value for OraTimeStam object.

*sec*
   Second value for OraTimeStam object.

*fsec*
   Fractional second value for OraTimeStam object.

Description

Use this member function to return time part of the object.

Example

...  unsigned char hour, minute, second, fsecond;  timeStamp.getTime(hour, minute, second, fsecond);  std::cout << "Hour: "    << hour  << "Minute: "    << minute  << "Second: "    << second  << "Frac Second: " << fsecond << std::endl;  ...

See Also

OraTimeStamp::getDate, OraTimeStamp::setTime

OraTimeStamp | Classes | OCL | Index
### 8.29.5 OraTimeStamp::isNull

```cpp
bool isNull() const;
```

**Return Value**

True, if value is NULL, otherwise false.

**Description**

Use this member function to determine if value of the current object is NULL.

**Example**

```cpp
...  if (!timeStamp.isNull())    std::cout << timeStamp.getString() << std::endl;  else    std::cout << "TIMESTAMP is NULL" << std::endl;  ...
```

**See Also**

- `OraTimeStamp::setNull`
- `OraTimeStamp` | `Classes` | `OCL` | `Index`

### 8.29.6 OraTimeStamp::operator -

```cpp
OraTimeStamp& operator-(const OraInterval& value);
```

**Return Value**

Current object value after subtracting.

**Parameters**

- `value`  
  OraInterval object value.

**Description**

Use this operator to subtract an OraInterval object from the current object value.

**See Also**

- `OraTimeStamp::operator+`
8.29.7 OraTimeStamp::operator -

friend OraInterval operator-(const OraTimeStamp &l, const OraTimeStamp &r);

Return Value
Resulting OraInterval value.

Parameters

l
  Left-hand value of an operator.

r
  Right-hand value of an operator.

Description
This operator takes two datetimes as input and stores their difference in OraInterval object.

See Also
OraTimeStamp::operator+, OraInterval

8.29.8 OraTimeStamp::operator !=

friend bool operator!=(const OraTimeStamp &l, const OraTimeStamp &r);

Return Value
True, if the OraTimeStamp object on the left side of the operator is not equal to the OraTimeStamp object on the right side; otherwise false.

Parameters

l
  Left-hand value of an operator.

r
  Right-hand value of an operator.
**Description**

Use this operator to test if the OraTimeStamp object on the left side of the operator is not equal to the OraTimeStamp object on the right side.

**See Also**

OraTimestamp::operator==

OraTimeStamp | Classes | OCL | Index

---

### 8.29.9 OraTimeStamp::operator +

OraTimeStamp& operator+(const OraInterval& value);

**Return Value**

Current object value after adding.

**Parameters**

*value*

OraInterval object value that will be added to the current object value.

**Description**

Use this operator to add an interval to the current object.

**See Also**

OraTimestamp::operator-

OraTimeStamp | Classes | OCL | Index

---

### 8.29.10 OraTimeStamp::operator <

friend bool operator<(const OraTimeStamp &l, const OraTimeStamp &r);

**Return Value**

True, if the OraTimeStamp object on the left side of the operator is less than the OraTimeStamp object on the right side; otherwise false.

**Parameters**

/
Left-hand value of an operator.

\( r \)

Right-hand value of an operator.

**Description**

Use this operator to test if the OraTimeStamp object on the left side of the operator is less than the OraTimeStamp object on the right side.

**See Also**

OraTimeStamp::operator>

OraTimeStamp | Classes | OCL | Index

### 8.29.11 OraTimeStamp::operator <=

```cpp
friend bool operator<=(const OraTimeStamp &l, const OraTimeStamp &r);
```

**Return Value**

True, if the OraTimeStamp object on the left side of the operator is less than or equal to the OraTimeStamp object on the right side; otherwise false.

**Parameters**

\( l \)

  Left-hand value of an operator.

\( r \)

  Right-hand value of an operator.

**Description**

Use this operator to test if the OraTimeStamp object on the left side of the operator is less than or equal to the OraTimeStamp object on the right side.

**See Also**

OraTimeStamp::operator>=

OraTimeStamp | Classes | OCL | Index
8.29.12 OraTimeStamp::operator =

OraTimeStamp& operator=(const OraTimeStamp& r);

**Return Value**

This OraTimeStamp object.

**Parameters**

- **r**
  
  New value for this OraTimeStamp object.

**Descrip on**

Call this operator to assign the value \( r \) to this OraTimeStamp object.

**See Also**

OraTimeStamp::OraTimeStamp

OraTimeStamp | Classes | OCL | Index

8.29.13 OraTimeStamp::operator ==

friend bool operator==(const OraTimeStamp &l, const OraTimeStamp &r);

**Return Value**

True, if the OraTimeStamp object on the left side of the operator is equal to the OraTimeStamp object on the right side; otherwise false.

**Parameters**

- **l**
  
  Left-hand value of an operator.

- **r**
  
  Right-hand value of an operator.

**Descrip on**

Use this operator to test if the OraTimeStamp object on the left side of the operator is equal to the OraTimeStamp object on the right side.

**See Also**
OraTimeStamp::operator!=

**Return Value**

True, if the OraTimeStamp object on the left side of the operator is greater than the OraTimeStamp object on the right side; otherwise false.

**Parameters**

/  
Left-hand value of an operator.

r  
Right-hand value of an operator.

**Description**

Use this operator to test if the OraTimeStamp object on the left side of the operator is greater than the OraTimeStamp object on the right side.

**See Also**

OraTimeStamp::operator<

OraTimeStamp::operator>=

friend bool operator>=(const OraTimeStamp &l, const OraTimeStamp &r);

**Return Value**

True, if the OraTimeStamp object on the left side of the operator is greater than or equal to the OraTimeStamp object on the right side; otherwise false.

**Parameters**

/  
Left-hand value of an operator.
Right-hand value of an operator.

**Description**

Use this operator to test if the OraTimeStamp object on the left side of the operator is greater than or equal to the OraTimeStamp object on the right side.

**See Also**

OraTimeStamp::operator<=

OraTimeStamp | Classes | OCL | Index

### 8.29.16 OraTimeStamp::OraTimeStamp

OraTimeStamp();
OraTimeStamp(const OraTimeStamp& src);

OraTimeStamp(short year, unsigned char month, unsigned char day, unsigned char hour, unsigned char min, unsigned char sec, unsigned fsec, const CRString& timeZone = "");

**Parameters**

**year**
Year value.

**month**
Month value.

**day**
Day value.

**hour**
Hour value.

**mn**
Minute value.

**sec**
Second value.

**fsec**
Fractional second value.

**timeZone**
Time zone string.
**Description**

This member function creates an object with `TIMESTAMP WITH TIME ZONE` type and initializes it with specified values. If one of the parameters is invalid an exception will be thrown.

**Example**

```c
OraTimeStamp timeStampTZ;
timeStampTZ.setTimeStamp(dtTimeStampTZ);
...
```

**See Also**

- `OraTimeStamp::setTimeStampType`
- `OraTimeStamp` | Classes | OCL | Index

### 8.29.17 OraTimeStamp::setDate

```c
void setDate(short year, unsigned char month, unsigned char day);
```

**Parameters**

- `year`  
  New year value.

- `month`  
  New month value.

- `day`  
  New day value.

**Description**

Call this member function to set date part of the object. If one of the parameters is invalid an exception will be thrown.

**See Also**

- `OraTimeStamp::getDate`
- `OraTimeStamp` | Classes | OCL | Index
### 8.29.18 OraTimeStamp::setNull

```cpp
void setNull();
```

**Description**
Use this member function to set current OraTimeStamp object value to NULL.

**See Also**
- `OraTimeStamp::isNull`
- `OraTimeStamp`

### 8.29.19 OraTimeStamp::fromString

```cpp
void fromString(const CRString& value, const CRString& format = "");
```

**Parameters**
- `value`
  - The input string to be converted to Oracle TIMESTAMP type.
- `format`
  - The conversion format.

**Description**
Use this member function to convert string to OraTimeStamp object value according to the specified format. For more information about format string see Oracle documentation.

**See Also**
- `OraTimeStamp::toString`
- `OraTimeStamp`

### 8.29.20 OraTimeStamp::setSysTimeStamp

```cpp
void setSysTimeStamp();
```

**Description**

Call this member function to set the system current date and time to the current OraTimeStamp object value.

**Example**

```cpp
... 
timeStamp.setSysTimeStamp();
std::cout << "Now " << timeStamp.getString() << std::endl;
...
```

**See Also**

`OraTimeStamp::toString`, `OraTimeStamp::fromString`

`OraTimeStamp` | Classes | OCL | Index

### 8.29.21 OraTimeStamp::setTime

```cpp
void setTime(unsigned char hour, unsigned char min, unsigned char sec, unsigned fsec);
```

**Parameters**

- **hour**
  
  New hour value.

- **min**
  
  New minute value.

- **sec**
  
  New second value.

- **fsec**
  
  New fractional second value.

**Description**

Call this member function to set time part of the object. If one of the parameters is invalid an exception will be thrown.

**See Also**

`OraTimeStamp::getTime`

`OraTimeStamp` | Classes | OCL | Index
8.29.22 OraTimeStamp::setTimeStampType

void setTimeStampType(OraDataType type);

Parameters

type
   One of the following Oracle types:
   dtTimeStamp - TIMESTAMP type;
   dtTimeStampTZ - TIMESTAMP WITH TIME ZONE type;
   dtTimeStampLTZ - TIMESTAMP WITH LOCAL TIME ZONE type.

Description

Use this member function to set new type that OraTimeStamp object represents. If the old value exists it will be deleted.

See Also

OraTimeStamp::timeStampType

OraTimeStamp | Classes | OCL | Index

8.29.23 OraTimeStamp::setTimeZoneOffset

void setTimeZoneOffset(signed char hour, signed char min);

Parameters

hour
   New hour offset value.

min
   New minute offset value.

Description

Call this member function to set time zone offset as hours and minutes. Type of the object must be TIMESTAMP WITH TIME ZONE. If one of the parameters is invalid an exception will be thrown.

See Also

OraTimeStamp::timeZoneOffset

OraTimeStamp | Classes | OCL | Index
8.29.24 OraTimeStamp::timeStampType

OraDataType timeStampType() const;

**Return Value**

dtTimeStamp - TIMESTAMP type;
dtTimeStampTZ - TIMESTAMP WITH TIME ZONE type;
dtTimeStampLTZ - TIMESTAMP WITH LOCAL TIME ZONE type.

**Description**

Use this member function to get OraTypeStamp object type. Value can be one of the three Oracle TIMESTAMP types.

**See Also**

OraTimeStamp::setTimeStampType

OraTimeStamp | Classes | OCL | Index

8.29.25 OraTimeStamp::timeZoneName

CRString timeZoneName() const;

**Return Value**

String that contains time zone offset.

**Description**

Call this member function to return string representation of time zone offset. Type of the object must be TIMESTAMP WITH TIME ZONE.

**See Also**

OraTimeStamp::timeZoneOffset

OraTimeStamp | Classes | OCL | Index

8.29.26 OraTimeStamp::timeZoneOffset

void timeZoneOffset(signed char& hour, signed char& min) const;
Parameters

*hour*

  Hour offset value.

*minute*

  Minute offset value.

Description

Use this member function to return time zone offset as hours and minutes. Type of the object must be `TIMESTAMP WITH TIME ZONE`.

See Also

`OraTimeStamp::setTimeZoneOffset`

`OraTimeStamp` | Classes | OCL | Index
8.30 OraTransaction

#include "ora.h"

Members

Construct on

OraTransaction();
OraTransaction(char gId[], int gIdLength);

~OraTransaction();

Attributes

int timeout() const;
void setTimeout(int timeout);

Description

OraTransaction represents local or distributed transaction. In the case of the distributed transaction different sessions on different Oracle servers work within the same transaction.

See Also

OraConnection::setTransaction

8.30.1 OraTransaction::OraTransaction

OraTransaction();
OraTransaction(char gId[], int gIdLength);

Parameters

$gId$
  Unique identifier of a distributed transaction.

$gIdLength$
  Length of gId array.

Description

Use first member function to create a local transaction. Use second method to create a
distributed transaction with specified unique identifier. The type of distributed transaction identifier is an array of decimal numbers of 128 bytes length.

See Also

OraConnection::commit, OraConnection::rollback

OraTransaction | Classes | OCL | Index

8.30.2 OraTransaction::setTimeout

```cpp
void setTimeout(int timeout);
```

**Parameters**

*timeout*

Indicates the number of seconds while the transaction can be inactive.

**Description**

Use this member function to specify the number of seconds while the transaction can be inactive before it is automatically aborted by the system.

**See Also**

OraTransaction::setTimeout

OraTransaction | Classes | OCL | Index

8.30.3 OraTransaction::timeout

```cpp
int timeout() const;
```

**Return Value**

The number of seconds while the transaction can be inactive.

**Description**

Use this member function to obtain the number of seconds while the transaction can be inactive till it is automatically aborted by the system.

**See Also**

OraTransaction::setTimeout
9 Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y_

A
addRef, CRSharedObject
allocCursor, OraCursor
allocLob, OraFile
allocLob, OraLob
assign, CRLob
assign, CRField
assign, OraParam

B
bindParams, OraCommand
breakExec, OraConnection

C
cancel, OraCommand
canFetch, OraCursor
CDA, OraCursor
checkValid, CRSharedObject
clear, CRLob
close, CRDataset
close, OraConnection
close, OraFile
close, OraQuery
commandText, OraCommand
commandType, OraCommand
commit, OraConnection
compress, CRLob
connection, OraCommand
count, CRFields
count, OraParams
CRLob
CRDataset
CRDataset, CRDataset
CRDate
CRDate, CRDate
createCommand, OraConnection
createProcCall, OraCommand
CRException
CRException, CRException
CRField
CRField, CRField
CRFields
CRMemDataset
CRMemDataset, CRMemDataset
CRSharedObject
CRSharedObject, CRSharedObject
CRString
CRValue
CRValue, CRValue
cursor, OraCommand
cursorState, OraCommand

dataType, CRField
dataType, OraParam
dateFormat, CRDate
day, CRDate
dayOfWeek, CRDate
direction, OraParam
disableBuffering, OraLob

enableBuffering, OraLob
errorCode, OraException
execute, OraCommand
execute, OraQuery
executeQuery, OraCommand
exist, OraFile

fetchAll, OraRecordset
fetchRows, OraRecordset
field, CRDataset
field, OraRecordset
fieldCount, CRDataset
fieldNo, CRField
fields, CRDataset
fileDir, OraFile
fileName, OraFile
findField, CRDataset
**findField**, OraRecordset
**findItem**, CRFields
**findItem**, OraParams
**findParam**, OraCommand
**first**, CRDataset
**format**, CRDate
**freeBookmark**, CRDataset
**freeCursor**, OraCursor
**freeLob**, OraFile
**freeLob**, OraLob

---

**getBFile**, OraField
**getBFile**, OraParam
**getBlob**, OraField
**getBlob**, OraParam
**getBookmark**, CRDataset
**getCDA**, OraCursor
**getChars**, CRLob
**getChars**, OraField
**getChars**, OraParam
**getChars**, CRField
**getClob**, OraField
**getClob**, OraParam
**getCursor**, OraParam
**getDate**, CRField
**getDate**, CRValue
**getDate**, OraParam
**getDouble**, CRField
**getDouble**, CRValue
**getDouble**, OraParam
**getFloat**, CRField
**getFloat**, CRValue
**getInt**, CRField
**getInt**, CRValue
**getInt**, OraParam
**getLDA**, OraConnection
**getLob**, OraField
**getLob**, OraParam
**getLob**, CRField
**getLong**, OraField
**getLong**, OraParam
**getLongRaw**, OraField
**getLongRaw**, OraParam
**getOCILOBLocator**, OraLob
**getOCIStmt**, OraCursor
**getOCISvcCtx**, OraConnection
**getOCISvcCtx**, OraLob
**getString**, CRLob
**getString**, CRField
getString, CRValue
getString, OraField
getString, OraParam
getValue, OraParam

H

hour, CRDate

I

init, OraLob
initFields, CRDataset
inTransaction, OraConnection
isActive, CRDataset
isActive, OraQuery
isActive, OraRecordset
isAutoCommit, OraConnection
isBOF, CRDataset
isCached, OraRecordset
isComplex, OraField
isConnected, OraConnection
isEOF, CRDataset
isInited, OraLob
isNull, CRField
isNull, CVValue
isNull, OraParam
isolationLevel, OraConnection
isOpen, OraFile
isPLSQL, OraCommand
isPrepared, OraCommand
isQuery, OraCommand
isRowsReturn, OraCommand
isTable, OraParam
item, CRFields
item, OraParams

L

last, CRDataset
length, OraLob
loadFromFile, CRLob
message, CRException
minute, CRDate
month, CRDate
move, CRDataset
moveTo, CRDataset
moveToBookmark, CRDataset

N

name, CRField
name, OraParam
next, CRDataset
now, CRDate

O

OCICallStyle, OraConnection
open, CRDataset
open, OraConnection
open, OraFile
open, OraQuery
operator, CRDataset
operator, CRFields
operator, OraParams
operator, OraCommand
operator, OraQuery
operator, OraRecordset
operator =, CRValue
operator =, OraParam
operator ==, CRDate
operator ==, CRValue
OraCursor
OraCommand
OraCommand, OraCommand
OraConnection
OraConnection, OraConnection
OraException
OraException, OraException
OraField, OraField
OraParam
OraParam, OraParam
OraParams
OraQuery, OraQuery
OraRecordset

P
param, OraCommand
paramCount, OraCommand
params, OraCommand
password, OraConnection
precision, CRField
prepare, OraCommand
prepare, OraQuery
prev, CRDataset

R

read, CRLob
readLob, OraLob
recordCount, CRDataset
recordNo, CRDataset
refCount, CRSharedObject
refresh, CRDataset
refresh, OraFile
release, CRSharedObject
required, CRField
rollback, OraConnection
rollbackToSavepoint, OraConnection
rowId, OraCommand
rowsProcessed, OraCommand

S

savepoint, OraConnection
saveToFile, CRLob
scale, CRField
second, CRDate
server, OraConnection
setArrayLength, OraCommand
setAutoCommit, OraConnection
setBFile, OraField
setBFile, OraParam
setBlob, CRField
setBlob, OraField
setBlob, OraParam
setCached, OraRecordset
setChars, CRLob
setChars, CRField
setChars, OraParam
setClob, OraField
setClob, OraParam
setCommandText, OraCommand
setCommandType, OraCommand
setConnection, OraCommand
setConnectString, OraConnection
setCursor, OraParam
setCursor, OraCommand
setDataType, OraParam
setDate, CRField
setDate, OraParam
setDateFormat, OraParam
setDay, CRDate
setDirection, OraParam
setDouble, CRField
setDouble, OraParam
setFetchRows, OraRecordset
setFileDir, OraFile
setFileName, OraFile
setFloat, CRField
setHour, CRDate
setInt, CRField
setInt, OraParam
setIsolationLevel, OraConnection
setLDA, OraConnection
setLob, CRField
setLob, OraField
setLob, OraParam
setLong, OraField
setLong, OraParam
setLongRaw, OraField
setLongRaw, OraParam
setMessage, CRException
setMinute, CRDate
setMonth, CRDate
setNull, CRField
setNull, CRValue
setNull, OraParam
setOCICallStyle, OraConnection
setOCISvcCtx, OraConnection
setOCISvcCtx, OraLob
setPassword, OraConnection
setSecond, CRDate
setServer, OraConnection
setSize, OraParam
setSQL, OraCommand
setText, CRLob
setText, CRField
setText, OraParam
setTextLength, OraParam
setUsername, OraConnection
setValue, CRValue
setValue, OraParam
setYear, CRDate
size, CRLob
size, CRField
size, OraField
size, OraParam
SQL, OraCommand
SQLType, OraCommand
startTransaction, OraConnection
state, OraCursor

tableLength, OraParam
type, CRValue

unprepare, CORaCommand
unprepare, OraQuery
username, OraConnection

write, CRLob
writeLob, OraLob

year, CRDate

Classes, OCL
Index

- A -
  addColumn 205
  addRef 89
  allocCursor 154
  allocLob 182, 215
  assign 60, 79, 226

- B -
  beginLoad 206
  bindParams 110
  breakExec 130
  breakOnError 206

- C -
  cancel 110
  canFetch 155
  charset 149
  checkValid 90
  clear 80
  clearColumns 207
  close 34, 131, 183, 256
  columnCount 207
  columnError 207
  commandText 110
  commandType 111
  commit 131
  compare 189, 276
  compress 80
  connection 111, 208
  count 75, 251
  CRDataset 33, 34
  CRDataset::RecordNo 41
  CRDate 46, 47
  createColumns 208
  createCommand 132
  createProcCall 111
  CRException 57
  CRField 59, 61
  CRFields 75
  CRLob 79
  CRMemDataset 88
  CRSharedObject 89, 90
  CRString 93
  CRValue 94, 95
  CRWString 92
  cursor 112
  cursorState 112

- D -
  dataType 61, 102, 226
  dateFormat 47, 102
  day 48, 189
  dayOfWeek 48
  defaultOracleHome 159
  direction 227
  disableBuffering 216

- E -
  enableBuffering 216
  errorCode 169
  errorOffset 113
  execute 113, 256
  executeQuery 115
  exist 183
  externalName 144

- F -
  fetchAll 262
  fetchRows 262
  field 35, 263
  fieldCount 35
  fieldNo 62
  fields 36
  fileDir 184
  fileName 184
  findField 36, 263
  findItem 76, 252
  findParam 115
  finishLoad 209
  first 37
  format 49
  fracSecond 190
  freeBookmark 37
  freeCursor 155
freeLob 184, 217
defreeOCI 159
defrFromString 202, 287

- G -

defgetBFile 172, 227
defgetBlob 172, 228
defgetBookmark 38
defgetCDA 155
defgetChars 63, 81, 173, 229
defgetClob 174, 230
defgetColumn 209
defgetCursor 230
defgetDate 64, 95, 231, 276
defgetDouble 64, 96, 231
defgetFloat 65, 96, 232
defgetInt 65, 97, 232
defgetLDA 133
defgetLob 66, 175, 233
defgetLong 175, 233
defgetLongRaw 176, 234
defgetOCIEnv 160
defgetOICLError 160
defgetOICILobLocator 217
defgetOCISstmt 156
defgetOCISvct 133, 217
defgetOCIVersion 160
defgetString 67, 82, 97, 177, 235
defgetTime 278
defgetValue 235
defgetWChars 62, 81, 173, 228
defgetWString 67, 82, 176, 234

- H -
defhome 134
defhomePath 135
defhour 50, 191

- I -
definit 218
definitFields 39
definitOCI 161
definternalName 145
defintervalType 191
definTransaction 136
defisActive 39, 222, 257, 264
defisAutoCommit 137
defisBOF 39
defisCached 265
defisComplex 177
defisConnected 137
defisEOF 40
defisEvents 161
defisInit 218
defisNull 68, 97, 192, 236, 279
defisOClInit 162
defisOClLoaded 162
defisolationLevel 137
defisOpen 185
defisPLSQL 116
defisPrepared 117
defisQuery 117
defisRowsReturn 117
defisThreaded 162
defisUnicode 148, 218
defitem 76, 252

- L -
deflast 41
deflength 219
defLoaderColumn 101, 102
defloadFromFile 83

- M -
defmessage 58
defminute 50, 192
defmonth 51, 193
defmove 41
defmoveTo 42
defmoveToBookmark 42

- N -
defname 68, 103, 237
defnext 43
defnow 51
Index

- O -

OCICallStyle 138, 163
OCI Library 163
open 43, 139, 185, 257
operator 279, 280, 281, 282, 283, 284
operator- 193
operator[] 43, 77, 258, 265
operator = 98, 237
operator == 52, 98
operator!= 194
operator_index 118, 253
operator+ 194
operator+= 195
operator< 195
operator<= 196
operator-= 196
operator== 197
operator> 198
operator>= 198
oracleHomeCount 164
oracleHomeName 164
oracleHomePath 164
OraCommand 108, 119
OraConnection 128, 139
OraCursor 154
OraException 169, 170
OraField 171, 177
OraFile 182
OraInterval 199
OraLoader 209
OraLob 215
OraParam 224, 238
OraParams 251
OraQuery 255, 258
OraRecordset 261
OraTimeStamp 274, 285
OraTransaction 292

- P -

param 119
paramCount 120
params 120
password 140
precision 69, 103
prepare 121, 259
prev 44
putColumnData 210
putColumnNull 211
putNextRow 211

- R -

read 83
readLob 219
recordCount 44
recordNo 45
refCount 90
refresh 45, 186
release 91
required 69
resetOracleHomes 165
rollback 140
rollbackToSavepoint 141
rowError 212
rowId 121
rowsProcessed 122

- S -

savepoint 141
saveToFile 84
scale 69, 104
second 52, 200
server 142
setArrayLength 122
setAutoCommit 142
setBeakOnError 212
setBFile 178, 238
setBlob 178, 238
setCached 255, 261, 266
setChars 70, 85, 239
setCharset 149
setClob 179, 240
setCommandText 123
setCommandType 123
setConnection 124, 212
setConnectMode 142
setConnectString 143
setCursor 125, 240
setDataType 104, 241
setDate 71, 241, 286
setDateFormat 52, 105
setDay 53
setDaySecond 200
setDefaultHome 144
setDirection 242
setDouble 71, 243
setEvents 166
setFetchRows 266
setFileDir 186
setFileName 186
setFloat 71, 243
setHome 144
setHour 53
setInitMode 167
setInt 72, 244
setIntervalType 201
setIsolationLevel 146
setLDA 146
setLob 72, 179, 244
setLong 180, 245
setLongRaw 180, 245
setMessage 58
setMinute 54
setMonth 54
setName 105
setNull 73, 99, 201, 246, 287
setOCIStyle 147, 167
setOCIStyle 168
setOCISvcCtx 147, 220
setOracleHome 166
setPassword 148
setPrecision 106
setScale 106
setSecond 54
setServer 150
setSize 106, 246
setSQL 125
setString 73, 86, 247
setSystemTimestamp 287
setTable 248
setTableLength 248
setTableName 213
setThreaded 168
setTime 288
setTimeout 293
setTimeType 289
setTimeZoneOffset 289
setTransaction 150
setUnicode 148, 220
setUsername 151
setValue 249
setWChars 70, 84, 239
setWString 73, 85, 247
setYear 55
setYearMonth 202
size 74, 86, 107, 180, 249
SQL 126
SQLType 126
start 223
startTransaction 151
state 156

tableLength 250
tableName 213
timeout 293
timeStampType 290
timeZoneName 290
timeZoneOffset 290
toString 190, 277
transaction 152
type 99
	number 127, 260
useDefaultOracleHome 165
username 152

write 87
writeLob 220

year 55, 203
Endnotes 2... (after index)