

O₂DB Access User Manual

C++ Interface

Release 5.0 - May 1998



Information in this document is subject to change without notice and should not be construed as a commitment by O₂ Technology.

The software described in this document is delivered under a license or nondisclosure agreement.

The software can only be used or copied in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium for any purpose other than the purchaser's own use.

Copyright 1992-1998 O₂ Technology.

All rights reserved. No part of this publication can be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopy without prior written permission of O₂ Technology.

O₂, O₂API, O₂C, O₂DBAccess, O₂Engine, O₂Graph, O₂Kit, O₂Look, O₂Store, O₂Tools, and O₂Web are registered trademarks of O₂ Technology.

SQL and AIX are registered trademarks of International Business Machines Corporation.

Sun, SunOS, and SOLARIS are registered trademarks of Sun Microsystems, Inc.

X Window System is a registered trademark of the Massachusetts Institute of Technology.

Unix is a registered trademark of Unix System Laboratories, Inc.

HPUX is a registered trademark of Hewlett-Packard Company.

BOSX is a registered trademark of Bull S.A.

IRIX is a registered trademark of Siemens Nixdorf, A.G.

NeXTStep is a registered trademark of the NeXT Computer, Inc.

Purify, Quantify are registered trademarks of Pure Software Inc.

Windows is a registered trademark of Microsoft Corporation.

All other company or product names quoted are trademarks or registered trademarks of their respective trademark holders.

Who should read this manual

This manual describes how to use O₂DBAccess. This O₂ module enables to connect O₂ applications to relational databases on remote hosts, and to import and export data from and to such systems. O₂DBAccess provides class libraries (O₂C and C++) for these tasks. The manual also describes how to invoke SQL statements from O₂. An example program is presented.

Other documents available are outlined, click below.

See [O2 Documentation set](#).



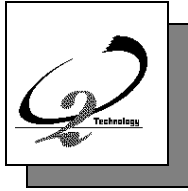


TABLE OF CONTENTS

This manual is divided into the following chapters:

- 1 - Introduction
- 2 - Utilization
- 3 - Classes
- 4 - Appendices



TABLE OF CONTENTS

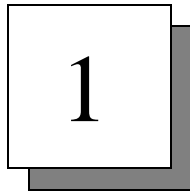
1	Introduction	9
	1.1 System Overview	10
	O2 Engine	12
	O2 Store	13
	O2DB Access	14
	1.2 Manual Overview	16
2	Programming Guide	17
	2.1 The o2dbaccess Library	18
	Classes	18
	2.2 Guidelines	19
	2.3 Accessing a Database	21
	Host connection and database log in	21
	Open Statement	22
	2.4 Preparing a Statement	23
	Associating an SQL statement	23
	Managing statements	23
	Storing a query result set	24
	2.5 Running a Statement	25
	2.6 Commit and Rollback	28
	2.7 Ending a Session	28
3	Class Library	29
	3.1 o2db_O2DBAccess	30
	server_error	31
	3.2 o2db_Connection	32
	connect	33
	disconnect	34
	logoff	35
	logon	36

TABLE OF CONTENTS

3.3	o2db_Session	38
	close	39
	commit	40
	open	41
	rollback	43
3.4	o2db_Statement	44
	add_cursor_variable	45
	add_input_variable	46
	associate	48
	operator >>	50
	operator <<	51
	o2db_sql_execute	53
4	Appendices	55
4.1	Example Application	56
	Defining a C++ schema	57
	Connecting to O2	58
	Connecting to the Relational database server	59
	Creating a table	61
	Populating the table	62
	Displaying the table content	64
	Querying the RDBMS	66
	Building the application	68
4.2	Configuration File	69
4.3	Possible Errors	71
	INDEX	77



TABLE OF CONTENTS



Introduction

Congratulations! You are now a user of O₂DBAccess!

O₂DBAccess is the O₂ module that enables you to communicate and work with relational databases on remote hosts

This chapter introduces the O₂ system and O₂DBAccess and outlines its various features and advantages. An overview of this User Manual is then given.

1.1 System Overview

The system architecture of O₂ is illustrated in [Figure 1.1](#).

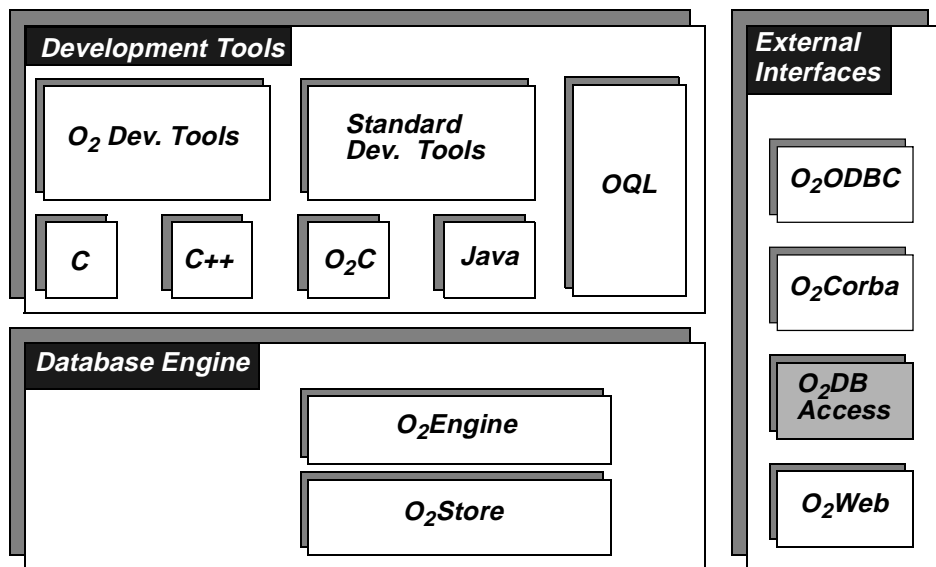


Figure 1.1: O₂ System Architecture

The O₂ system can be viewed as consisting of three components. The *Database Engine* provides all the features of a Database system and an object-oriented system. This engine is accessed with *Development Tools*, such as various programming languages, O₂ development tools and any standard development tool. Numerous *External Interfaces* are provided. All encompassing, O₂ is a versatile, portable, distributed, high-performance dynamic object-oriented database system.

Database Engine:

- O₂Store The database management system provides low level facilities, through O₂Store API, to access and manage a database: disk volumes, files, records, indices and transactions.
- O₂Engine The object database engine provides direct control of schemas, classes, objects and transactions, through O₂Engine API. It provides full text indexing and search capabilities with O₂Search and spatial indexing and retrieval capabilities with O₂Spatial. It includes a Notification manager for informing other clients connected to the same O₂ server that an event has occurred, a Version manager for handling multiple object versions and a Replication API for synchronizing multiple copies of an O₂ system.

System Overview

Programming Languages:

O₂ objects may be created and managed using the following programming languages, utilizing all the features available with O₂ (persistence, collection management, transaction management, OQL queries, etc.)

- C O₂ functions can be invoked by C programs.
- C++ ODMG compliant C++ binding.
- Java ODMG compliant Java binding.
- O₂C A powerful and elegant object-oriented fourth generation language specialized for easy development of object database applications.
- OQL ODMG standard, easy-to-use SQL-like object query language with special features for dealing with complex O₂ objects and methods.

O₂ Development Tools:

- O₂Graph Create, modify and edit any type of object graph.
- O₂Look Design and develop graphical user interfaces, provides interactive manipulation of complex and multimedia objects.
- O₂Kit Library of predefined classes and methods for faster development of user applications.
- O₂Tools Complete graphical programming environment to design and develop O₂ database applications.

Standard Development Tools:

All standard programming languages can be used with standard environments (e.g. Visual C++, Sun Sparcworks).

External Interfaces:

- O₂Corba Create an O₂/ Orbix server to access an O₂ database with CORBA.
- O₂DBAccess Connect O₂ applications to relational databases on remote hosts and invoke SQL statements.
- O₂ODBC Connect remote ODBC client applications to O₂ databases.
- O₂Web Create an O₂ World Wide Web server to access an O₂ database through the internet network.

O₂ Engine

O₂Engine has all the features of a database engine providing transparent management of data persistence, data sharing and data reliability, as well as all the features of an object-oriented system including the manipulation of complex objects with identity, classes, types, methods, multiple inheritance, overriding and late binding of methods.

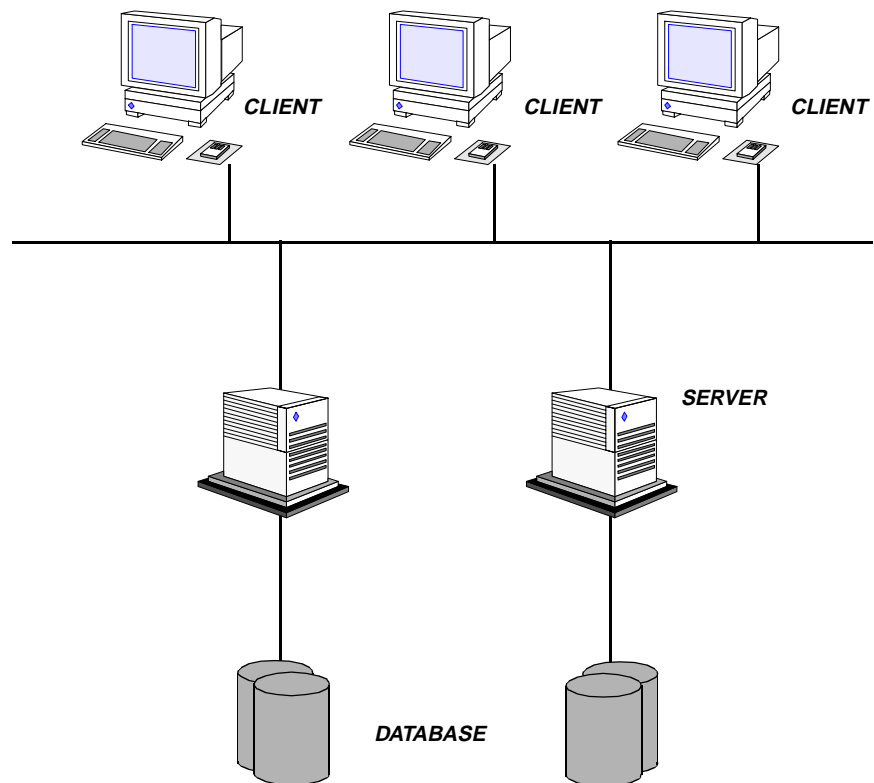


Figure 1.2: Client/server architecture

System Overview : O₂ Store

O₂ Store

The O₂Store physical storage management system offers you the following features:

- Transactional management of persistent structures.
- Client/ server architecture.
- Rollbacks and crash recovery.

O₂Store has the client/ server architecture shown in [Figure 1.2](#). The server process provides persistence, disk management, concurrency control, data recovery, and database security.

The features offered by O₂Engine and O₂Store are shown in [Figure 1.3](#) .

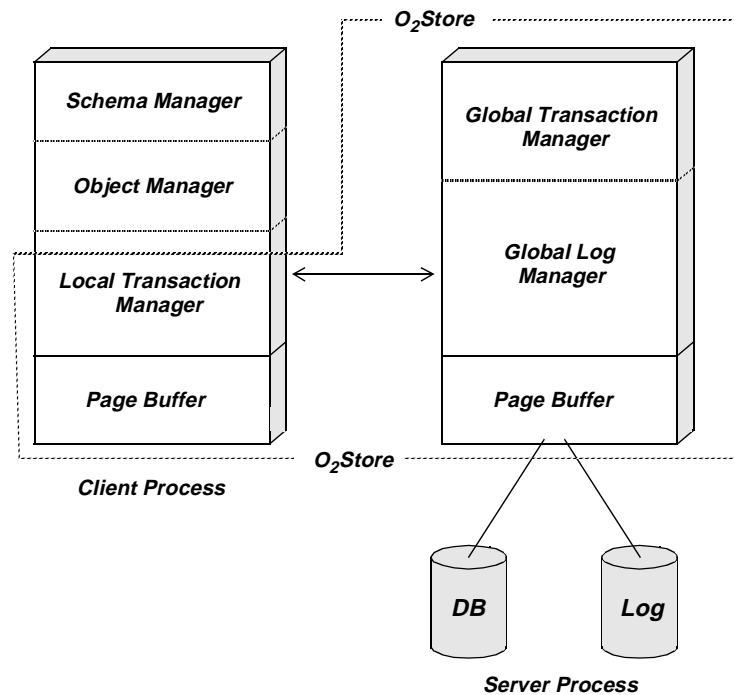


Figure 1.3: Global architecture showing O₂Store layer

O₂DB Access

O₂DBAccess is a set of C++ classes that enables O₂ applications to communicate and work with relational databases on remote hosts.

These classes allow you to carry out the following actions from your applications:

- Connect to a server and set up a session on a remote database.
- Run any SQL statement in the SQL syntax of that database.
- Fetch data as required from the database to the user application into O₂ objects.
- Mirror the commit and rollback facilities of some databases.
- Close the database session and terminate the connection to the host.

You can also retrieve error message text corresponding to database error codes.

O₂DBAccess is based on the SequeLink protocol as shown in [Figure 1.4](#). This is a software package that enables a client application to access simultaneously different relational databases residing on different servers that are connected to one or more types of local network.

SequeLink uniformly manages the different network protocols and the heterogeneity between platforms¹. With O₂DBAccess you can link to any platform currently supported by SequeLink.

1. For information about all possible network-host-database combinations, call O₂Line.

System Overview : O2DB Access

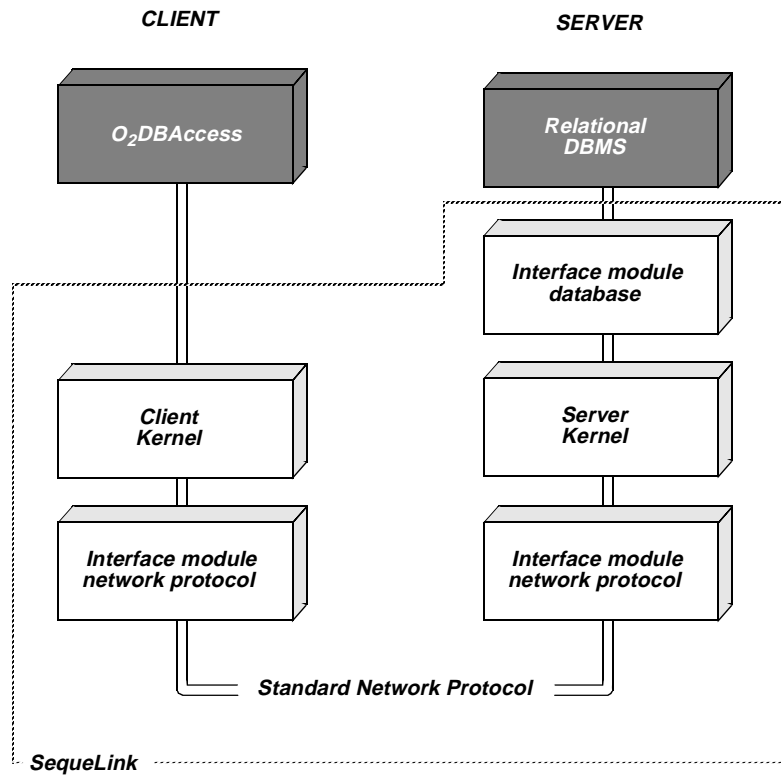


Figure 1.4: O₂DBAccess and SequeLink

1.2 Manual Overview

This manual is divided up into the following chapters:

- **Chapter 1 - Introduction**

A short introduction to the O₂ system, O₂Engine, O₂Store, and O₂DBAccess.

- **Chapter 2- Programming Guide**

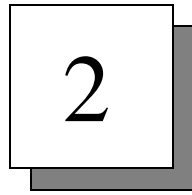
This chapter describes how to use O₂DBAccess. It covers accessing a database, preparing a statement, fetching data, commit and rollback, and running the statement.

- **Chapter 3 - Class Library**

This chapter details the various classes of the O₂DBAccess library: `o2db_O2DBAccess`, `o2db_Connection`, `o2db_Session`, and `o2db_Statement`.

- **Chapter 4 - Appendices**

This chapter includes an example program. It gives possible error codes and the configuration file.



Programming Guide

This chapter details how to use O₂DBAccess.

It is divided into the following chapters:

- [The o2dbaccess Library](#)
- [Guidelines](#)
- [Accessing a Database](#)
- [Preparing a Statement](#)
- [Running a Statement](#)
- [Commit and Rollback](#)
- [Ending a Session](#)

2.1 The o2dbaccess Library

O₂DBAccess is in fact a standard C++ library called `o2dbaccess` that you can use in any of your C++ application.

Classes

The `o2dbaccess` library, shown in [Figure 2.1](#), provides classes that enable you to communicate and work with the remote database.

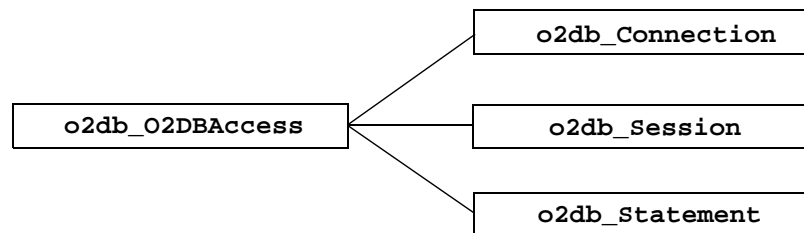


Figure 2.1: `o2dbaccess` library

The library classes are as follows:

- **`o2db_O2DBAccess`**

An `o2db_O2DBAccess` object defines the common resources of O₂DBAccess classes. It contains one function, `server_error`, which you use to obtain the RDBMS-detected error codes.

- **`o2db_Connection`**

`o2db_Connection` is a subclass of `o2db_O2DBAccess`. An `o2db_Connection` object defines and maintains a connection to a remote host.

- **`o2db_Session`**

`o2db_session` is a subclass of `o2db_O2DBAccess`. An `o2db_session` object defines and maintains a connection to a database server. It manages the transactions of the session.

- **`o2db_Statement`**

`o2db_statement` is a subclass of `o2db_O2DBAccess`. An `o2db_statement` object defines an access context to a database and contains information that is required to run an SQL statement.

Guidelines

2.2 Guidelines

To send an SQL statement for processing on remote database, you need to carry out the following steps:

1. Set up a connection to the host **server** machine.
2. Set up a session on the database by logging on.
3. Open a statement on the remote database server.
4. Prepare the statement to be run. This means putting information about the SQL query you want to execute in the statement and the C++ objects that will store the SQL result.
5. Run the statement.
6. If information has been inserted, updated or deleted, make the changes permanent, or undo them.
7. If you do not want to rerun the statement, close it.
8. Close the session by logging off the database.
9. End the connection by disconnecting from the host.

Each step corresponds to a particular function. [Table 2.1](#) shows these functions and its corresponding step.

The class of the function is given before the function name.

These functions constitute the basic set of functions that you need to use.

Table 2.1

Function set

Functions	Step
<code>o2db_Connect::connection</code>	Set up a connection
<code>o2db_Connection::logon</code>	Set up a session
<code>o2db_Session::open</code>	Open an access context

Table 2.1

Function set

Functions	Step
<code>o2db_Statement::associate</code> <code>o2db_Statement::add_cursor_variable</code> <code>o2db_Statement::add_input_variable</code> <code>o2db_Statement::>></code> <code>o2db_Statement::<<</code>	Prepare a statement
<code>o2db_Statement::execute</code>	Run a statement
<code>o2db_Session::commit</code>	Make the changes permanent
<code>o2db_Session::rollback</code>	Undo a transaction
<code>o2db_Session::close</code>	Close the context
<code>o2db_Connection::logoff</code>	End the session
<code>o2db_Connection::disconnect</code>	End the connection

All these various steps are detailed below.

For a full description of each specific function refer to Chapter 3.

Accessing a Database

2.3 Accessing a Database

You must first connect to the remote host and log onto the database.

Host connection and database log in

With O₂DBAccess, you do this by creating a `o2db_Connection` object. You must call the `connect` function on this object to connect to the remote host. You then call the `logon` function on this object to log onto the database and begin a session. The `logon` function returns a pointer to an `o2db_Session` object.

Once connected and logged on, you can run as many SQL statements as you want.

For example:

```
#include "o2db_o2db_access.h"

int create_session()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    o2db_Session *session;

    status = connection.connect("oracle", "user", "pwd");

    session = connection.logon("dbuser", "dbpasswd", &status);

    if ( status == o2dbE_SERVER) {
        msg = connection.server_error(&code);
        cerr << "Error: " << code << "    msg: " << msg << endl;
    } else if (status != O2DB_OK)
        cerr << "An error occurs: " << status << endl;

    ...
}
```

When your transactions with the database are finished, you use the `logoff` and `disconnect` functions to respectively end the session and remote host connection.

Refer to [Section 3.2](#) for a full description of all these `o2db_Connection` functions.

Open Statement

Once connected to the remote host and logged on to the database, you must now open a statement in which you can run an SQL statement. You do this using the `open` function from your `o2db_session` object (result of the `logon` function). The statement contains the statement itself and any additional information that may be needed to run the statement. This function returns an object of class `o2db_statement`.

A `o2db_statement` object remains open until you explicitly close it using the `close` function (of your `o2db_session` object), or until you end the database session using the `logout` function. However, you do not need to close a statement if you want to use it for a different statement. You simply re-use it.

For example:

```
#include "o2db_o2db_access.h"

int open()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    status = connection.connect("oracle", "user", "pwd");
    session = connection.logon( "dbuser", "dbpasswd",
                               &status);
    statement = session->open(&status);
    if (status == o2dbE_SERVER) {
        msg = connection.server_error(&code);
        cerr << "Error: " << code << "    msg: " << msg << endl;
    } else if (status != O2DB_OK)
        cerr << "An error occurs: " << status << endl;
    ...
}
```

`o2db_statement` and how to manage statements are fully explained in [Section 2.4](#) below; [Section 3.3](#) gives a description of the `open` function.

Note

The statement is local to the logon session in which it is used and the number of statements you can open at the same time is restricted to 100. However, you rarely need more than 15 and the external database or yourself can impose a lower limit.

2.4 Preparing a Statement

The next step after accessing the database is to prepare the SQL statements you want to run.

Associating an SQL statement

You must firstly define the SQL statement you want to run using the `associate` function of the `o2db_statement` class.

The `associate` function also sends the statement to the RDBMS database for validation. It is at this point that any SQL syntax errors are trapped. If any are found, you can get the database error codes by calling the `server_error` function on the `o2db_statement` object (see [Section 3.1](#) for details of this function and [Section 4.3](#) for a list of possible errors).

If the SQL statement does not need a result object (i.e. it is not a select statement) and it contains no parameter markers, O₂DBAccess needs no more information.

In this case, you can run your statement immediately using the `o2db_sql_execute` function. See [Section 2.5](#) for more details.

In the other cases, i.e. if the statement is a select statement and/ or if it contains parameter markers, you must provide O₂DBAccess with more information. This is explained in the remainder of this section.

Managing statements

When you want to run a select statement or a statement that contains parameter markers or both, you need to provide more information before the statement is run.

The information needed includes the result object and its projection list, which you store in the statement using the `add_cursor_variable` function or the '>>' operator of the class `o2db_statement`.

You also need to store the input parameters of the statement using the `add_input_variable` function or the '<<' operator of the class `o2db_statement`.

You can manage your statements in the following three different ways:

1. Use one statement for one specific SQL statement and close it as soon as it has been run.
2. Open a statement, use it for one SQL statement, and then reuse it for another SQL statement by simply associating it to a new SQL statement.

You can do this as many times as you want until you want to close the statement.

3. Open a statement for a particular SQL statement that you want to run several times. You keep the statement open and associated to the same SQL statement until you no longer wish to rerun the statement.

While the statement is open, you can rerun it with new values for any of its parameters using the `o2db_sql_execute` function. The new values of the parameters are given using the `add_input_variable` function (or the '<<' operator).

Note

Re-associating a statement resets the parameter list of the statement as well as the result information.

Storing a query result set

Storing the result set of a query function involves the following steps:

1. You must firstly define the C++ classes where you want the retrieved data to be stored.
2. You must then give the relevant transfer information. This means associating a part of the result set (i.e. the name of a column) to a part of the C++ storage class (i.e. the name of a C++ attribute).

You do this by defining data buffers using the `add_cursor_variable` function or the '>>' operator of the class `o2db_statement`.

You can then run the statement using the `o2db_sql_execute` function.

See the example in the next section.

2.5 Running a Statement

If the SQL statement does not need a result object (i.e. it is not a select statement), you can run it directly using the `o2db_sql_execute` function with only your statement as a parameter.

If the statement is a select statement, you must provide the relevant transfer information as detailed in [Section 2.4](#), and then call the `o2db_sql_execute` function with the statement and the C++ object that will store the result of the query as parameters.

For example, suppose you set up a single connection, a single session in which you want to run three SQL statements: a `CREATE` statement, an `INSERT` statement with parameters that you want to run twice, and a `SELECT` statement; you call the following sequence of functions to process each step.

```
int test()
{
    d_Set<d_Ref<Person> > the_persons("ThePersons");
    d_Iterator<d_Ref<Person> > iter;
    d_Ref<Person> person;

    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    int status, code;

    status = connection.connect("oracle", "user", "passwd");
    if (status != O2DB_OK)
        return status;

    session = connection.logon("oracleuser",
                              "oracleuserpasswd", &status);
    if (status != O2DB_OK)
        return status;

    statement = session->open(&status);
    if (status != O2DB_OK) return status;

    // (1) A statement without parameters:
    // -----
    status = statement->associate("CREATE TABLE person \
                                (id_person SMALLINT null, \
                                 name VARCHAR(16) null, \
                                 firstname VARCHAR(16) null)");
    if (status != O2DB_OK) return status;
```

```
status = o2db_sql_execute(statement);
if (status != O2DB_OK)
    return status;

// (2) A statement with 3 input parameters:
//-----
status = statement->associate( "INSERT INTO \
                               person(id_person, name, \
                                   firstname) \
                               VALUES (?, ?, ?)");

// First execution
*statement << 1 << "Doe" << "John";
status = o2db_sql_execute(statement);
if (status != O2DB_OK)
    return status;

// Second execution with different parameters
*statement << 2 << "Smith" << "Paul";
status = o2db_sql_execute(statement);
if (status != O2DB_OK)
    return status;

// (3) A statement returning a result
//-----
status = statement->associate("SELECT name, firstname \
                              FROM person");

// The class Person has at least 2 attributes:
// pname and pfirstname
*statement >> "pname" >> "pfirstname";
status = o2db_sql_execute(statement, the_persons);
if (status != O2DB_OK)
    return status;

// Now the C++ set contains the selected information
iter = the_persons.create_iterator();
while (iter.next(person)) {
    cout << "\t*** [ "
         << person -> pname << " "
         << person -> pfirstname << " ]" << endl;
}
```

Running a Statement : Storing a query result set

```
// Ending the session and disconnecting
status = session->close(statement);
if (status != O2DB_OK)
    return status;

status = connection.logoff(session);
if (status != O2DB_OK)
    return status;

connection.disconnect();
}
```

2.6 Commit and Rollback

The `commit` and `rollback` functions mirror a feature of some databases in allowing you to systematically and explicitly make permanent or unroll a series of related database actions at strategic points in a session.

2.7 Ending a Session

When all your transactions with the database have been carried out, the functions `close`, `logoff` and `disconnect` close the statement and end the session and connection respectively..

```
#include "o2db_o2db_access.h"

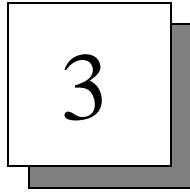
int end_session()
{
    int status;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;

    ...

    statement = session->open(&status);

    ...

    status = session->close(statement);
    connection.logoff(session);
    connection.disconnect();
    return status;
}
```



Class Library

CLASS SET AND THEIR MEMBER FUNCTIONS

O₂DBAccess is a set of C++ classes that enables O₂ applications to communicate and work with relational databases on remote hosts.

This chapter details all these classes and their respective member functions.

It is divided into the following sections:

- [o2db_O2DBAccess](#) - This is the super class of all the other classes of the package.
- [o2db_Connection](#) - This class manages the connection to a remote host.
- [o2db_Session](#) - This class manages the connection to a database server.
- [o2db_Statement](#) - This class manages all the resources belonging to an SQL statement.

3.1 **o2db_O2DBAccess**

`o2db_O2DBAccess` is the super class of the classes `o2db_Connection`, `o2db_Session`, and `o2db_Statement`. It manages common resources and owns the following public member function:

- `server_error`

o2db_O2DBAccess

server_error

Summary Gets the last error code from the database server.

Syntax `char* server_error(int *code);`

Arguments `code` A pointer on an integer that in return will contain the error code.

Description This member function is used by an O₂DBAccess programmer to obtain extra information about a database server related error. It can be applied to an instance of the `o2db_O2DBAccess` class or one of its subclasses after an error of type `o2dbE_SERVER` occurred. The function puts in the integer pointed by `code` the database server error code raised by the failure. The string returned by this function contains a textual description of the error, when applicable.

As this string is stored in a static area, it is only available until the next call to `server_error` occurs. This string must not be freed by the user.

Returns A string with the textual description of the error.

Example

```
#include "o2db_o2db_access.h"

int get_emp(o2db_Session *session)
{
    int code, status;
    char *msg;
    o2db_Statement statement = session->open(&status);
    status = statement->associate("SELECT * from emp");
    if (status == o2dbE_SERVER) {
        msg = statement->server_error(&code);
        cerr << "error: " << code << " msg: " << msg << endl;
    }
    ...
}
```

3.2 `o2db_Connection`

An object of this class defines and maintains the connection to a remote host. The information needed to set up a link between the O₂ application and the host system is found in a configuration file. It is made up of a set of network and host specific parameters. Refer to [Section 4.2](#) for a full description of the configuration file.

This section describes the following member functions of the `o2db_Connection` class:

- `connect`
- `disconnect`
- `logoff`
- `logon`

o2db_Connection : connect

connect

Summary	Connects to a remote host.						
Syntax	<code>int connect(char *c_name, char *username, char *password);</code>						
Arguments	<table><tr><td><code>c_name</code></td><td>This is a string containing a valid entry in the O2DBAccess configuration file (see Section 4.2).</td></tr><tr><td><code>username</code></td><td>This is a string containing the user name on the remote host.</td></tr><tr><td><code>password</code></td><td>This is a string containing the password of the user on the remote host.</td></tr></table>	<code>c_name</code>	This is a string containing a valid entry in the O2DBAccess configuration file (see Section 4.2).	<code>username</code>	This is a string containing the user name on the remote host.	<code>password</code>	This is a string containing the password of the user on the remote host.
<code>c_name</code>	This is a string containing a valid entry in the O2DBAccess configuration file (see Section 4.2).						
<code>username</code>	This is a string containing the user name on the remote host.						
<code>password</code>	This is a string containing the password of the user on the remote host.						
Description	This function sets up a link between the workstation and a remote SequeLink server.						
Returns	o2DB_OK if the connection succeeded, or one of the following error codes if the connection failed: <ul style="list-style-type: none">• o2dbE_STILL_CONNECT• o2dbE_RC_NOTFOUND• o2dbE_FILE_NOTFOUND• o2dbE_INVALID_RC• o2dbE_UNKN_NETWORK• o2dbE_OPEN_FILE• o2dbE_SERVER						

Example

```
#include "o2db_o2db_access.h"

int connect()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    status = connection.connect("oracle", "user", "pwd");
    if ( status != O2DB_OK) {
        cerr << "Connection error: " << status << endl;
    }
    ...
}
```

disconnect

- Summary** Disconnects from a remote host.
- Syntax** `void disconnect(void);`
- Arguments** None.
- Description** The `disconnect` member function disconnects the link between the application and the remote SequeLink server. All the current sessions and statements are released by this function.
- Returns** Nothing

Example

```
#include "o2db_o2db_access.h"

int dbaccess()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    status = connection.connect("oracle", "user", "pwd");
    ...
    connection.disconnect();
    return status;
}
```

o2db_Connection : logoff

logoff

Summary	Ends a database server session.
Syntax	<code>int logoff(o2db_Session *session);</code>
Arguments	<code>session</code> A pointer on the session to close.
Description	The <code>logoff</code> member function ends an existing session on a database server. All the statements created during the session are destroyed and the database server resources are released. The pointers to <code>o2db_statement</code> objects are no longer valid after this call.
Returns	A status code whose value is <code>O2DB_OK</code> if the operation was successful, or one of the following error codes if the operation failed: <ul style="list-style-type: none">• <code>o2dbE_NOT_CONNECT</code>• <code>o2dbE_NOT_MEMBER</code>• <code>o2dbE_SQLNK</code>• <code>o2dbE_SERVER</code>

Example

```
#include "o2db_o2db_access.h"

int dbaccess()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    o2db_Session *session;
    status = connection.connect("oracle", "user", "pwd");
    session = connection.logon( "dbuser", "dbpasswd",
                               &status);

    ...
    status = connection.logoff(session);
    if (status == o2dbE_SERVER) {
        msg = connection.server_error(&code);
        cerr << "error: " << code << "    msg: " << msg << endl;
    } else if (status != O2DB_OK)
        cerr << "An error occurs: " << status << endl;

    ...
    connection.disconnect();
    return status;
}
```

logon

Summary	Logs onto a database server.						
Syntax	<pre>o2db_session *logon(char *logon1, char *logon2, int *retcode);</pre>						
Arguments	<table><tr><td><code>logon1</code></td><td>A parameter needed to log onto the database server.</td></tr><tr><td><code>logon2</code></td><td>A parameter needed to log onto the database server.</td></tr><tr><td><code>retcode</code></td><td>A pointer on an integer that will contain an error code on failure.</td></tr></table>	<code>logon1</code>	A parameter needed to log onto the database server.	<code>logon2</code>	A parameter needed to log onto the database server.	<code>retcode</code>	A pointer on an integer that will contain an error code on failure.
<code>logon1</code>	A parameter needed to log onto the database server.						
<code>logon2</code>	A parameter needed to log onto the database server.						
<code>retcode</code>	A pointer on an integer that will contain an error code on failure.						
Description	<p>The <code>logon</code> member starts a new session on the database server. The <code>logon1</code> and <code>logon2</code> parameters are database dependent but usually contains a database user and its password. For more information, refer to the documentation <i>Using SequelLink with your Database and Server</i>. The parameter length must be less than 256 characters.</p> <p><code>retcode</code> is a pointer to an integer that will be set to an error code in case of failure. The possible error codes returned in <code>retcode</code> are as follows:</p> <ul style="list-style-type: none">• <code>o2dbE_NOT_CONNECT</code>• <code>o2dbE_TOOLONG</code>• <code>o2dbE_SQLNK</code>• <code>o2dbE_SERVER</code>						
Returns	A pointer to an object of the class <code>o2db_session</code> , or NULL if the login failed.						

o2db_Connection : logon

Example

```
#include "o2db_o2db_access.h"

int create_session()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    o2db_Session *session;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon("dbuser", "dbpasswd", &status);
    if ( status == o2dbE_SERVER) {
        msg = connection.server_error(&code);
        cerr << "Error: " << code << "    msg: " << msg << endl;
    } else if (status != O2DB_OK)
        cerr << "An error occurs: " << status << endl;

    ...
    connection.disconnect();
    return status;
}
```

3.3 `o2db_Session`

An object of the `o2db_session` class is created by the `logon` function of the `o2db_Connection` class and manages the connection to a database server.

This class has the following member functions:

- `close`
- `commit`
- `open`
- `rollback`

o2db_Session : close

close

Summary	Closes a statement.
Syntax	<code>int close(o2db_Statement *statement);</code>
Arguments	<code>statement</code> A pointer on a statement returned by the member function <code>open</code> .
Description	This function closes an already opened statement and releases all resources associated with this statement.
Returns	A status code whose value is <code>O2DB_OK</code> if the operation was successful, or one of the following error codes if the operation failed: <ul style="list-style-type: none">• <code>o2dbE_NOT_LOGON</code>• <code>o2dbE_NOT_MEMBER</code>• <code>o2dbE_SQLNK</code>• <code>o2dbE_SERVER</code>

Example

```
#include "o2db_o2db_access.h"

int dbaccess()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon( "dbuser", "dbpasswd",
                                &status);
    ...
    statement = session->open(&status);
    ...
    status = session->close(statement);
    ...
    connection.disconnect();
    return status;
}
```

commit

Summary	Performs a commit on the database server.
Syntax	<code>int commit(void);</code>
Arguments	None
Description	This function performs a commit on the database server, thus validating what has been done in the current session.
Returns	A status code whose value is <code>O2DB_OK</code> if the operation was successful, or one of the following error codes if the operation failed: <ul style="list-style-type: none">• <code>o2dbE_NOT_LOGON</code>• <code>o2dbE_SQLNK</code>• <code>o2dbE_SERVER</code>

Example

```
#include "o2db_o2db_access.h"

int dbaccess()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon("dbuser", "dbpasswd",
                               &status);
    ...
    statement = session->open(&status);
    ...
    status = session->commit();
    ...
    status = session->close(statement);
    ...
    connection.disconnect();
    return status
}
```

o2db_Session : open

open

Summary	Creates a new statement.
Syntax	<code>o2db_statement *open(int *retcode);</code>
Arguments	retcode A pointer on an integer that will contain an error code on failure.
Description	<p>This function opens a new statement that will allow a programmer to execute SQL statements. A pointer on the created statement is returned.</p> <p>retcode is a pointer on an integer that will be set to an error code in case of failure. The possible error codes returned in retcode are as follows:</p> <ul style="list-style-type: none">• <code>o2dbE_NOT_LOGON</code>• <code>o2dbE_NO_MORE_STATEMENTS</code>• <code>o2dbE_SQLNK</code>• <code>o2dbE_SERVER</code>
Returns	A pointer on an object of the class <code>o2db_statement</code> or NULL if the operation failed.

Example

```
#include "o2db_o2db_access.h"

int dbaccess()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon( "dbuser", "dbpasswd",
                                &status);

    ...
    statement = session->open(&status);
    ...
    status = session->commit();
    ...
    status = session->close(statement);
    ...
    connection.disconnect();
    return status;
}
```

o2db_Session : rollback

rollback

Summary	Performs a rollback on the database server.
Syntax	<code>int rollback(void);</code>
Arguments	None.
Description	This function performs a rollback on the database server, thus invalidating what has been done in the current session.
Returns	A status code whose value is <code>O2DB_OK</code> if the operation was successful, or one of the following error codes if the operation failed: <ul style="list-style-type: none">• <code>o2dbE_NOT_LOGON</code>• <code>o2dbE_SQLNK</code>• <code>o2dbE_SERVER</code>

Example

```
#include "o2db_o2db_access.h"

int dbaccess()
{
    int code, status;
    char *msg;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon("dbuser", "dbpasswd", &status);
    ...
    statement = session->open(&status);
    ...
    status = session->rollback();
    ...
    status = session->close(statement);
    ...
    connection.disconnect();
    return status;
}
```

3.4 `o2db_Statement`

An object of the `o2db_statement` class is created by the `open` function of the `o2db_session` class and manages all the resources related to SQL statements.

This class has the following member functions:

- `add_cursor_variable`
- `add_input_variable`
- `associate`
- `operator >>`
- `operator <<`

And the following function:

- `o2db_sql_execute`

o2db_Statement : add_cursor_variable

add_cursor_variable

- Summary** Specifies a C++ attribute name to store an SQL result set column.
- Syntax** `int add_cursor_variable(char *s);`
- Arguments** `s` A string representing a C++ attribute name.
- Description** When a select statement is performed, the result set has to be stored in a C++ object or in a collection of C++ objects. This function must be called once for each column of the result set returned by the SQL statement, in order to specify in which attribute of a C++ object the values of a column will be put.
- The calls to this function must be done in the order the columns of the result set occur in the SQL statement.
- Returns** O2DB_OK.
- Example**

```
#include "o2db_o2db_access.h"

int get_persons()
{
    int code, status;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    d_Set<d_Ref<Person> > persons;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon("dbuser", "dbpasswd", &status);
    ...
    statement = session->open(&status);
    ...
    status = statement->associate("SELECT name,age FROM emp");
    // assuming class Person {
    //     char *name_att;
    //     int age_att;
    // }
    statement->add_cursor_variable("name_att");
    statement->add_cursor_variable("age_att");
    ...
    status = session->close(statement);
    ...
    connection.disconnect();
    return status;
}
```

add_input_variable

Summary	Evaluates an input variable of an SQL statement.
Syntax	<pre>int add_input_variable(const char *s); int add_input_variable(char c); int add_input_variable(unsigned char uc); int add_input_variable(short s); int add_input_variable(unsigned short us); int add_input_variable(int i); int add_input_variable(unsigned int ui); int add_input_variable(long l); int add_input_variable(unsigned long ul); int add_input_variable(float f); int add_input_variable(double d); int add_input_variable(const d_Bits &s);</pre>
Arguments	xx A C++ value to assign to the input variable. xx can be any kind of C++ type, according to the type expected by the database server.
Description	<p>This function is used by a programmer who defined an SQL statement containing input variables. These variables are written using '?' in the SQL statement. It is necessary to give a value for these variables before running the statement.</p> <p>This function associates a value of any type with an input variable. The type used must be consistent with the expected type in the database server. This function must be called once for each input variable defined in the statement, in the order of the '?' markers appearing in the SQL statement.</p>
Returns	<p>A status code whose value is <code>O2DB_OK</code> if the operation was successful, or one of the following error codes if the operation failed:</p> <ul style="list-style-type: none"> • <code>o2dbe_NOT_OPEN</code> • <code>o2dbe_NOSTMT</code> • <code>o2dbe_INTERNAL</code> • <code>o2dbe_NOT_SUPPORTED</code> • <code>o2dbe_NILREF</code> • <code>o2dbe_RANGE</code> • <code>o2dbe_MISMATCH</code> • <code>o2dbe_SQLNK</code> • <code>o2dbe_SERVER</code>

o2db_Statement : add_input_variable

Example

```
#include "o2db_o2db_access.h"

int get_persons()
{
    int code, status;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    d_Set<d_Ref<Person> > persons;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon("dbuser", "dbpasswd", &status);
    ...
    statement = session->open(&status);
    ...
    status = statement->associate("SELECT name,age FROM emp
                                WHERE age<?");
    ...
    statement->add_cursor_variable("name_att");
    statement->add_cursor_variable("age_att");
    statement->add_input_variable(20);
    ...
    status = session->close(statement);
    connection.disconnect();
    return status;
}
```

associate

Summary	Defines an SQL statement.
Syntax	<code>int associate(char *stmt);</code>
Arguments	<code>stmt</code> The text of the SQL statement that will be executed.
Description	<p>This function is used to define an SQL statement. It can be any valid SQL sentence in which input variables are replaced by question marks ('?').</p> <p>The actual values for the input variables are given using the function <code>add_input_variable</code> or the operator <code><<</code>. Calling <code>associate</code> on a statement invalidates all previous variable definitions for this statement.</p>
Returns	<p>A status code whose value is <code>O2DB_OK</code> if the operation was successful, or one of the following error codes if the operation failed:</p> <ul style="list-style-type: none">• <code>o2dbE_NOT_OPEN</code>• <code>o2dbE_NOSTMT</code>• <code>o2dbE_TOOLONG</code>• <code>o2dbE_SQLNK</code>• <code>o2dbE_SERVER</code>

o2db_Statement : associate

Example

```
#include "o2db_o2db_access.h"

int get_persons()
{
    int code, status;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    d_Set<d_Ref<Person> > persons;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon("dbuser", "dbpasswd",
                               &status);

    ...
    statement = session->open(&status);
    ...
    status = statement->associate("SELECT name,age FROM emp
                                  WHERE age<?");
    ...
    status = session->close(statement);
    ...
    connection.disconnect();
    return status;
}
```

operator >>

Summary	Adds a cursor variable.	
Syntax	<code>o2db_Statement & operator >>(o2db_Statement &q , const char *s);</code>	
Arguments	q	A reference to an object of the class <code>o2db_Statement</code> that manages the SQL statement for which the variable is defined.
	s	A string representing a C++ attribute name.
Description	This operator acts in exactly the same way as the member function <code>add_cursor_variable</code> . It is a more direct interface to it.	
Returns	A reference to the statement.	
Example		

```
int get_persons()
{
    int code, status;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    d_Set<d_Ref<Person> > persons;
    d_Iterator<d_Ref<Person> > iter;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon("dbuser", "dbpasswd", &status);
    ...
    statement = session->open(&status);
    ...
    status = statement->associate("SELECT name,age FROM emp
                                WHERE age<?");
    *statement >> "name_att" >> "age_att";
    statement->add_input_variable(20);
    status = o2db_sql_execute(*statement, persons);
    ...
    status = session->close(statement);
    connection.disconnect();
    iter = persons.create_iterator();
    while (iter.next(p)) {
        cout << "Name : " << p.name_att << endl;
        cout << "Age : " << p.age_att << endl;
        cout << endl;
    }
    return status;
}
```

o2db_Statement : operator <<

operator <<

Summary	Adds an input variable.	
Syntax	<pre>o2db_Statement &operator << (o2db_Statement &q, const char *s); o2db_Statement &operator << (o2db_Statement &q, char c); o2db_Statement &operator << (o2db_Statement &q, unsigned char uc); o2db_Statement &operator << (o2db_Statement &q, short s); o2db_Statement &operator << (o2db_Statement &q, unsigned short us); o2db_Statement &operator << (o2db_Statement &q, int i); o2db_Statement &operator << (o2db_Statement &q, unsigned int ui); o2db_Statement &operator << (o2db_Statement &q, long l); o2db_Statement &operator << (o2db_Statement &q, unsigned long ul); o2db_Statement &operator << (o2db_Statement &q, float f); o2db_Statement &operator << (o2db_Statement &q, double d); o2db_Statement &operator << (o2db_Statement &q, const d_Bits &s);</pre>	
Arguments	q	A reference to an object of the class <code>o2db_Statement</code> that manages the SQL statement for which the variable is defined.
	xx	A C++ value to associate to the input variable. xx can be any kind of C++ type, according to the type expected by the database server.
Description	This operator acts in exactly the same way as the member function <code>add_input_variable</code> . It is a more direct interface to it	
Returns	A reference to the statement.	

Example

```
int get_persons()
{
    int code, status;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    d_Set<d_Ref<Person> > persons;
    d_Iterator<d_Ref<Person> > iter;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon("dbuser", "dbpasswd", &status);
    ...
    statement = session->open(&status);
    ...
    statement->associate("SELECT name,age FROM emp
                        WHERE age<?");
    *statement >> "name_att" >> "age_att";
    *statement << 20;
    status = o2db_sql_execute(*statement, persons);
    ...
    status = session->close(statement);
    ...
    connection.disconnect();
    iter = persons.create_iterator();
    while (iter.next(p)) {
        cout << "Name : " << p.name_att << endl;
        cout << "Age : " << p.age_att << endl;
        cout << endl;
    }
    return status;
}
```

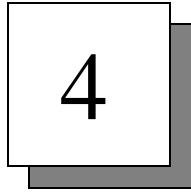
o2db_Statement : o2db_sql_execute

o2db_sql_execute

Summary	Executes an SQL statement.						
Syntax	<pre>int o2db_sql_execute(o2db_Statement & q); int o2db_sql_execute(o2db_Statement & q, o2_col_root & coll); int o2db_sql_execute(o2db_Statement & q, d_Ref<T> & o);</pre>						
Arguments	<table><tr><td>q</td><td>A reference to an object of the class <code>o2db_Statement</code> that manages the SQL statement to be executed.</td></tr><tr><td>coll</td><td>A reference to any O₂ collection in which the result of the statement must inserted.</td></tr><tr><td>o</td><td>A reference to an O₂ object in which the result (for a single line result set) must be put.</td></tr></table>	q	A reference to an object of the class <code>o2db_Statement</code> that manages the SQL statement to be executed.	coll	A reference to any O ₂ collection in which the result of the statement must inserted.	o	A reference to an O ₂ object in which the result (for a single line result set) must be put.
q	A reference to an object of the class <code>o2db_Statement</code> that manages the SQL statement to be executed.						
coll	A reference to any O ₂ collection in which the result of the statement must inserted.						
o	A reference to an O ₂ object in which the result (for a single line result set) must be put.						
Description	This function executes a previously defined SQL statement. The first function returns immediatly after the statement has been run (used for a statement without result) and the other two functions put the result of the statement into the target object. The target object can be a collection of object when the programmer expects to have a result set with multiple rows or can be an object when the programmer knows that the statement will return a single row result set.						
Returns	A status code whose value is <code>O2DB_OK</code> if the operation was successful, or one of the following error codes if the operation failed: <ul style="list-style-type: none">• <code>o2dbE_NOT_OPEN</code>• <code>o2dbE_NOSTMT</code>• <code>o2dbE_NOT_SELECT</code>• <code>o2dbE_DEFINED</code>• <code>o2dbE_MISMATCH</code>• <code>o2dbE_NOT_SUPPORTED</code>• <code>o2dbE_NOT_UNIQUE</code>• <code>o2dbE_INVALID_NAME</code>• <code>o2dbE_NOT_EXECUTED</code>• <code>o2dbE_NILREF</code>• <code>o2dbE_SQLNK</code>• <code>o2dbE_SERVER</code>						

Example

```
int get_persons()
{
    int code, status;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;
    d_Set<d_Ref<Person> > persons;
    d_Iterator<d_Ref<Person> > iter;
    status = connection.connect("oracle", "user", "pwd");
    ...
    session = connection.logon("dbuser", "dbpasswd", &status);
    ...
    statement = session->open(&status);
    ...
    statement->associate("SELECT name,age FROM emp
                        WHERE age<?");
    statement->add_cursor_variable("name_att");
    statement->add_cursor_variable("age_att");
    statement->add_input_variable(20);
    status = o2db_sql_execute(*statement, persons);
    ...
    status = session->close(statement);
    connection.disconnect();
    iter = persons.create_iterator();
    while (iter.next(p)) {
        cout << "Name : " << p.name_att << endl;
        cout << "Age  : " << p.age_att  << endl;
        cout << endl;
    }
    return status;
}
```



Appendices

This chapter contains the following appendices:

- [Example Application](#)
- [Configuration File](#)
- [Possible Errors](#)

4.1 Example Application

This example illustrates the various steps you must go through in order to use O₂DBAccess inside an ODMG C++ application.

This section is divided up into the following sections:

- [Defining a C++ schema](#)
- [Connecting to O2](#)
- [Connecting to the Relational database server](#)
- [Creating a table](#)
- [Populating the table](#)
- [Displaying the table content](#)
- [Querying the RDBMS](#)
- [Building the application](#)

Example Application : Defining a C++ schema

Defining a C++ schema

You must first define the C++ classes that will be used to receive the results of the SQL statements run in the application. In this example, we define a class `Person`. The C++ classes defined here must be imported in O₂ as persistence capable classes.

To be able to store an SQL result set, you will need to define collections for these classes.

```
#include "o2util_CC.hxx"

class Person {
public:
    d_Short id_person;
    char *name;
    char *firstname;
    int age;
    d_Char employed;
    d_Long zip_code;
};
```

Note

You make a C++ class persistence capable using `o2import` or a directive in the `o2makegen` configuration file (see the Section "Building the application").

Connecting to O₂

The main function of the application is an ODMG C++ function that opens a session with the O₂ server, opens a database, starts an O₂ transaction and creates a persistent root. It then calls the function `manage_sql`, which handles all the `dbaccess` part and finally commits the O₂ transaction and disconnects from the O₂ server.

```
main(int argc , char *argv[])
{
    char *o2_home = getenv("O2HOME");
    char *o2_system = getenv("O2SYSTEM");
    char *o2_server = getenv("O2SERVER");

    d_Session session;
    d_Database database;
    d_Transaction transaction;

    if (! o2_home) {
        cerr << "Cannot retrieve O2HOME" << endl;
        exit(1);
    }
    if (! o2_system) {
        cerr << "Cannot retrieve O2SYSTEM" << endl;
        exit(1);
    }
    if (! o2_server) {
        cerr << "Cannot retrieve O2SERVER" << endl;
        exit(1);
    }

    if (session.begin(argc, argv, o2_system, o2_server, o2_home))
    {
        cerr << "Cannot connect to o2" << endl;
        exit(1);
    }

    database.open("o2dbaccess_b");
    transaction.begin();
    database.create_persistent_root( "ThePersons",
                                    "d_Set<d_Ref<Person> >");

    manage_sql();

    transaction.commit();
    database.close();
    session.end();
}
```

Connecting to the Relational database server

This function is the entry point of the O₂DBAccess related code. It establishes a connection with the remote database server, opens a database, and creates a statement.

At this point, the functions that issue the queries are called, and finally, all the resources opened on the remote database server are closed.

```
void manage_sql()
{
    int retcode, svr_error;
    char *msg;
    o2db_Connection connection;
    o2db_Session *session;
    o2db_Statement *statement;

    // connection with the Sequelink server.

    retcode = connection.connect( "oracle", "user",
                                  "userpasswd");

    if (retcode != O2DB_OK)
        cerr << "Cannot open a connection with the
Sequelink server" << endl;

    // open an session

    session = connection.logon( "oracleuser",
                                "oracleuserpasswd",
                                &retcode);

    if (retcode != O2DB_OK) {
        if (retcode == o2dbE_SERVER) {
            msg = connection.server_error(&svr_error);
            cerr << "error: " << svr_error << " msg: " <<
msg << endl;
        } else cerr << "dbaccess error: " << retcode;
        return;
    }
}
```

```
// create a statement

statement = session->open(&retcode);
if (retcode != O2DB_OK) {
    if (retcode == o2dbE_SERVER) {
        msg = connection.server_error(&svr_error);
        cerr << "error: " << svr_error << "    msg: "
<< msg << endl;
    } else cerr << "dbaccess error: " << retcode;
    return;
}

// call the queries

person_table_creation(statement);
person_table_initialization(statement);
person_table_display(statement);

retcode = connection.logoff(session);
if (retcode != O2DB_OK) {
    if (retcode == o2dbE_SERVER) {
        msg = connection.server_error(&svr_error);
        cerr << "error: " << svr_error << "    msg: "
<< msg << endl;
    } else cerr << "dbaccess error: " << retcode;
    return;
}

retcode = connection.disconnect();
if (retcode != O2DB_OK) {
    if (retcode == o2dbE_SERVER) {
        msg = connection.server_error(&svr_error);
        cerr << "error: " << svr_error << "    msg: "
<< msg << endl;
    } else cerr << "dbaccess error: " << retcode;
    return;
}
}
```

Example Application : Creating a table

Creating a table

This function creates a new table on the RDBMS. The SQL `CREATE` statement is associated to the `O2DBAccess` statement and as there is no more information to give, the statement is executed.

```
void person_table_creation(o2db_Statement * statement)
{
    int retcode, svr_error;
    char *msg;

    retcode = statement->associate("CREATE TABLE person \
                                   ( id_person SMALLINT null, \
                                   name VARCHAR(16) null, \
                                   firstname VARCHAR(16) null, \
                                   age INTEGER null, \
                                   employed INTEGER default 0, \
                                   zip_code NUMERIC null)");

    if (retcode != O2DB_OK) {
        if (retcode == o2dbE_SERVER) {
            msg = statement->server_error(&svr_error);
            cerr << "error: " << svr_error << "   msg: " << msg
<< endl;
        } else cerr << "dbaccess error: " << retcode;
        return;
    }

    retcode = o2db_sql_execute(*statement);
    if (retcode != O2DB_OK) {
        if (retcode == o2dbE_SERVER) {
            msg = statement->server_error(&svr_error);
            cerr << "error: " << svr_error << "   msg: " << msg
<< endl;
        } else cerr << "dbaccess error: " << retcode;
        return;
    }
}
```

Populating the table

The following function populates the previously created table.

The `INSERT SQL` statement is associated to the `O2DBAccess` statement. It contains parameters markers (denoted with '?'), that must be bound with the actual values of the parameters.

In this example, this is performed using the '<<' operator. It could also have been performed using the `add_input_variable` member function.

The `person_data` table used in the example is not detailed and could be replaced by any input source.

After the parameters are bound, the statement is executed. In this example, note that the same associated statement is used several times to insert several rows in the person table, each time with a new binding of the actual parameters.

```
void person_table_initialisation(o2db_Statement *
statement)
{
    int retcode, svr_error;
    char *msg;

    retcode = statement->associate("INSERT INTO \
                                   person (id_person, \
                                           name, \
                                           firstname, \
                                           age, \
                                           employed, \
                                           zip_code) \
                                   VALUES (? , ? , ? , ? , ? , ? , ?)");
    if (retcode != O2DB_OK) {
        if (retcode == o2dbE_SERVER) {
            msg = statement->server_error(&svr_error);
            cerr << "error: " << svr_error << "    msg: "
<< msg << endl;
        } else cerr << "dbaccess error: " << retcode;
        return;
    }
}
```

Example Application : Populating the table

```
for (int i = 0 ; i<NB_PERSON ; i++) {
    *statement << person_data[i].id_person;
    *statement << person_data[i].name;
    *statement << person_data[i].firstname;
    *statement << person_data[i].age;
    *statement << person_data[i].employed;
    *statement << person_data[i].zip_code;

    retcode = o2db_sql_execute(*statement);
    if (retcode != O2DB_OK) {
        if (retcode == o2dbE_SERVER) {
            msg = statement->server_error(&svr_error);
            cerr << "error: " << svr_error
                << "    msg: " << msg << endl;
        } else cerr << "dbaccess error: " << retcode;
    }
    return;
}
}
```

Displaying the table content

This function shows how to handle a `SELECT SQL` statement. This kind of statement returns a result set that needs to be stored in C++ objects.

A collection of C++ objects of the persistence capable class `Person` is created and is given as parameter to the `o2db_sql_execute` function. After the statement execution, this collection will contain the result set of the SQL query. In this example, the collection "ThePersons" is persistent, but results can be stored in temporary collections too.

Note that no cursor variables are defined in this function to specify to which C++ attributes the table columns are associated. In this case, all the C++ attributes of the class given to the `o2db_sql_execute` function are implicitly associated to the result set in the order they appear in the C++ class definition. You must check that the SQL query returns a valid data set for the given C++ object.

After the statement is executed, an iterator is created on the collection and is used to display each retrieved person

Example Application

```
void person_table_display(o2db_Statement * statement)
{
    int retcode, svr_error;
    char *msg;
    d_Set<d_Ref<Person> > the_persons("ThePersons");
    d_Iterator<d_Ref<Person> > iter;
    d_Ref<Person> person;

    retcode = statement->associate( "SELECT * \
                                   FROM person");
    if (retcode != O2DB_OK) {
        if (retcode == o2dbE_SERVER) {
            msg = statement->server_error(&svr_error);
            cerr << "error: " << svr_error << "   msg: "
<< msg << endl;
        } else cerr << "dbaccess error: " << retcode;
        return;
    }

    retcode = o2db_sql_execute(*statement, the_persons);
    if (retcode != O2DB_OK) {
        if (retcode == o2dbE_SERVER) {
            msg = statement->server_error(&svr_error);
            cerr << "error: " << svr_error
                << "   msg: " << msg << endl;
        } else cerr << "dbaccess error: " << retcode;
        return;
    }
    iter = the_persons.create_iterator();
    while (iter.next(person)) {
        cout << "\t*** [ "
            << person -> id_person << " "
            << person -> name << " "
            << person -> firstname << " "
            << person -> age << " "
            << person -> employed << " "
            << person -> zip_code << " ]" << endl;
    }
}
```

Querying the RDBMS

This query is quite similar to the previous one except that cursor variables are explicitly defined (because the number and order of appearance of the attributes of the given C++ class are not the same as the columns of the result set).

This example also shows how to combine input variables and cursor variables in the same statement. Moreover the result is now put in a temporary list.

The function firstly associates the SQL statement to the O₂DBAccess statement, gives the value '35' as the actual parameter for the marker used in the SQL statement, and defines the cursor variables.

In this case the column "name" of the result set is associated to the attribute "name" of the C++ class (`*statement >> "name"`) and the column "firstname" of the result set is associated to the attribute "firstname" of the C++ class (`*statement >> ... >> "firstname"`).

Note

Even if it is the case in this function, the C++ attribute associated to a column of the SQL result set does not need to have the same name as the column. For instance, we could have associated an attribute called "person_name" to the column "name" of the result set

Example Application

```
void person_table_query(o2db_Statement * statement)
{
    int retcode, svr_error;
    char *msg;
    d_List<d_Ref<Person> > young_people;
    d_Iterator<d_Ref<Person> > iter;
    d_Ref<Person> person;

    retcode = statement->associate("SELECT name, firstname\
                                   FROM person \
                                   WHERE age < ? \
                                   ORDER BY age");

    if (retcode != O2DB_OK) {
        if (retcode == o2dbE_SERVER) {
            msg = statement->server_error(&svr_error);
            cerr << "error: " << svr_error
                 << "    msg: " << msg << endl;
        } else cerr << "dbaccess error: " << retcode;
        return;
    }
    // binds the input variable
    *statement << 35;

    // binds the cursor variables.
    *statement >> "name" >> "firstname";
    retcode = o2db_sql_execute(*statement, young_people);
    if (retcode != O2DB_OK) {
        if (retcode == o2dbE_SERVER) {
            msg = statement->server_error(&svr_error);
            cerr << "error: " << svr_error
                 << "    msg: " << msg << endl;
        } else cerr << "dbaccess error: " << retcode;
        return;
    }

    iter = young_people.create_iterator();
    while (iter.next(person)) {
        cout << "\t*** [ "
             << person -> name << " "
             << person -> firstname << " ]" << endl;
    }
}
```

Building the application

You build the application in exactly the same way as you build a classical ODMG C++ application.

To use O₂DBAccess you just need to link-edit your application with the O₂DBAccess library and with the SequeLink client libraries and to add a directive to look for SequeLink client include files.

The following code gives an `o2makegen` configuration file that can be used to create an application (see the "C++ Binding Guide Manual").

Note

Before using a generated application, it is mandatory to create an O₂ schema and base using the `o2dsa` program.

```
O2System: $O2SYSTEM
O2Server: $O2SERVER
O2Schema: o2dbaccess_s

+UseConfirmClasses

O2Home: $O2HOME

ImpFiles: Person.hxx
[Person.hxx]ImpClasses: Person
ImpSet: Person

Sources: example.cc
ProgramLib: o2dbaccess SqlnkCor SqlnkNet SqlnkDef
ProgramLibDir: $O2HOME/thirdparty/SequeLink/lib/IRIX
Include: $O2HOME/thirdparty/SequeLink/include
ProgramName: example
ProgramObjs: example.o
```

Configuration File

4.2 Configuration File

The configuration file `o2dbaccess.cf` contains the information needed to link up your O₂ application and a remote host, in the form of a set of network and host-specific parameters.

You can change the file name by specifying a new name and its path in the environment variable `O2DBACCESS`. This variable must contain the full path of the file.

In order of priority, the file is first taken from the working directory, then `$HOME` and then the O₂ installation directory. It is an ASCII file where each line corresponds to a named link description. It contains one entry for each named link with comments beginning with `#`. The line format is:

```
c_name:network:host:port:service:lu_name:mode:max_statements:type_checking
```

Each field is described below.

<i>c_name</i>	Link name. Specify a link name each time you invoke the <code>o2db_Connection::connect</code> member function.
<i>network</i>	Type of network protocol used. Possible values: TCP (MacTCP on Macintosh), ADSP, APPC, DECnet, NetBIOS and AppleTalk.
<i>host</i>	RDBMS remote host/ node/ zone name.
<i>port</i>	Port on which the Sequelink server is listening. Default is 2000. If you want to use the default port number, use a <code>'*'</code> .
<i>service</i>	SequeLink (database) service name that you want to connect to. You can find this name in the servermap file on the remote host. See SequeLink manual.
<i>lu_name</i>	Physical LU Name in APPC network protocol (optional). If it is not applicable, use a <code>'*'</code> .
<i>mode</i>	APPC mode in APPC network protocol (optional). If it is not applicable, use a <code>'*'</code> .
<i>max_statements</i>	Integer (from 0 to 100) specifying maximum number of statements that can be opened at the same time during a session. If 0, the default value (15) is used.
<i>type_checking</i>	Boolean specifying whether to enable (<code>true</code>) or disable (<code>false</code>) type checking in the <code>o2db_statement::add_cursor_variable</code> and <code>o2db_statement::add_input_variable</code> member functions. Default value is <code>false</code> .

Warning !

The type checking is not supported for all RDBMS. An example configuration file is as follows:

```
supra on salome:TCP:salome:LSPSUPRA2:*:*:15:false
db2 on sgph11:DECnet:sgph11:MVSDB2:*:*:0:true
oracle:TCP:o2tech:oracle_services:*:*:0:false
```

4.3 Possible Errors

Each member function returns an internal error code.

This section describes these error codes. You can obtain RDBMS-detected error codes using the `server_error` member function. See [Section 3.1](#) for a description of this member function.

- **-1001**

Code: `o2dbE_SERVER (-1001)`

Call: All.

Cause: RDBMS-detected error has occurred.

Action: Consult error code and message text calling the `o2db_O2DBAccess::server_error` member function.

- **-2001**

Code: `o2dbE_SQLNK (-2001)`

Call: All.

Cause: A SequeLink error has occurred. Should usually only be issued on a `o2db_Connection::connect` member function call.

Action: Check the link parameters, user name and password. The connect failure reason is in `<network>srv.log` file on the remote host.

- **-3001**

Code: `o2dbE_STILL_CONNECT (-3001)`

Call: `o2db_Connection::connect`

Cause: You are still connected to a host.

Action: The `o2db_Connection::connect` member function was called but not the `o2db_Connection::disconnect` member function.

- **-3002**

Code: `o2dbE_NOT_CONNECT (-3002)`

Call: `o2db_Connection::logon, o2db_Connection::logoff`

Cause: Not connected.

Action: Check completion of the previous `connect` member function call.

- **-3003**

Code: `o2dbE_NOT_MEMBER (-3003)`

Call: `o2db_Connection::logoff, o2db_Session::close`

Cause: The Session/ Statement object has not been created by the receiver.

Action: Check the member function call syntax.

- **-3004**

Code: `o2dbE_NOT_LOGON (-3004)`

Call: `o2db_Session::open, o2db_Session::close, o2db_Session::commit, o2db_Session::rollback`

Cause: Not logged on.

Action: Check completion of the previous `o2db_Connection::logon` member function call.

- **-3005**

Code: `o2dbE_TOOLONG (-3005)`

Call: `o2db_Connection::logon, o2db_Statement::associate`

Cause: The parameters are too long.

Action: Decrease parameter length.

- **-3006**

Code: `o2dbE_NOT_OPEN (-3006)`

Call: `o2db_Statement::associate, o2db_Statement::add_input_variable, o2db_sql_execute`

Cause: The statement has not been opened.

Action: Check the completion of the previous `o2db_Session::open` member function call.

- **-3008**

Code: `o2dbE_NOSTMT (-3008)`

Call: `o2db_Statement::associate, o2db_Statement::add_input_variable, o2db_sql_execute`

Cause: The SQL statement is empty.

Action: Check the member function call syntax.

Possible Errors : Building the application

- **-3009**

Code: `o2dbE_FILE_NOTFOUND (-3009)`
Call: `o2db_Connection::connect`
Cause: The configuration file was not found.
Action: Check that the configuration file exists.

- **-3010**

Code: `o2dbE_RC_NOTFOUND (-3010)`
Call: `o2db_Connection::connect`
Cause: No description of the link parameters for this link name.
Action: Check configuration file contents and the member function call syntax.

- **-3011**

Code: `o2dbE_INVALID_RC (-3011)`
Call: `o2db_Connection::connect`
Cause: Invalid line in the configuration file.
Action: Check the configuration file contents.

- **-3012**

Code: `o2dbE_UNKN_NETWORK (-3012)`
Call: `o2db_Connection::connect`
Cause: Unknown network protocol.
Action: Check configuration file contents.

- **-3013**

Code: `o2dbE_OPEN_FILE (-3013)`
Call: `o2db_Connection::connect`
Cause: The configuration file cannot be opened.
Action: Check that the configuration file exists and check its access rights.

- **-3014**
 - Code: `o2dbE_NOT_SELECT (-3014)`
 - Call: `o2db_sql_execute`
 - Cause: The SQL statement is not a select statement.
 - Action: Check the syntax of the SQL statement.

- **-3015**
 - Code: `o2dbE_RANGE (-3015)`
 - Call: `o2db_Statement::add_input_variable`
 - Cause: The order number is out of range.
 - Action: Check the order number.

- **-3016**
 - Code: `o2dbE_DEFINED (-3016)`
 - Call: `o2db_sql_execute`
 - Cause: The parameter or the result object is yet defined.
 - Action: Check the order number.

- **-3017**
 - Code: `o2dbE_MISMATCH (-3017)`
 - Call: `o2db_Statement::add_input_variable,`
`o2db_sql_execute`
 - Cause: Type checking failed.
 - Action: Check the result or parameter type.

- **-3018**
 - Code: `o2dbE_NOT_SUPPORTED (-3018)`
 - Call: `o2db_Statement::add_input_variable,`
`o2db_sql_execute`
 - Cause: One of the atomic types used is not supported in the current version of O₂DBAccess.
 - Action: Check the type of result object.

Possible Errors : Building the application

- **-3021**

Code: `o2dbE_INVALID_NAME (-3021)`
Call: `o2db_sql_execute`
Cause: One of the names is not an attribute name.
Action: Check the list of attribute names.

- **-3022**

Code: `o2dbE_NOT_EXECUTED (-3022)`
Call: `o2db_sql_execute`
Cause: The statement has not been executed.
Action: Check the completion of the previous `o2db_sql_execute` member function call.

- **-3023**

Code: `o2dbE_NO_MORE_STATEMENTS (-3023)`
Call: `o2db_Statement::open`
Cause: An attempt was made to exceed the maximum number open statements allowed.
Action: Close some statements.

- **-3024**

Code: `o2dbE_NILREF (-3024)`
Call: `o2db_Statement::add_input_variable,`
`o2db_sql_execute`
Cause: The result object or a parameter is `nil`.
Action: Check the member function call syntax.

- **-3025**

Code: `o2dbE_NOMEM (-3025)`
Call: All.
Cause: Not enough memory.
Action: Close some statements.

- **-4001**

Code: `o2dbe_INTERNAL (-4001)`

Call: All.

Cause: Internal error. It should not normally be issued.

Action: Contact your local technical support.

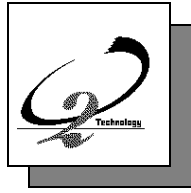
- **-5001**

Code: `o2dbw_NOT_UNIQUE (-5001)`

Call: `o2db_sql_execute`

Cause: This a warning. There is more than one row to fetch whereas the result type is not a collection.

Action: Nothing. Only the first row has been fetched.



INDEX



A

`add_cursor_variable`
Function [23–24, 45](#)

`add_input_variable`
Function [23–24, 46](#)

Application
Building [68](#)
Example [56–68](#)

Architecture
Client/ server [13](#)
O₂ [10](#)

`associate`
Function [23, 48](#)

C

C [11](#)

C++
Interface [11](#)

C++ Schema
Definition [57](#)

`c_name` [69](#)

Class [18](#)
`o2db_Connection` [21, 32](#)
`o2db_O2DBAccess` [30](#)
`o2db_Session` [21, 38](#)
`o2db_Statement` [44](#)

Client/ server architecture [13](#)

`close`
Function [22, 39](#)

`commit`
Function [28, 40](#)

Configuration
File [68, 69](#)

`connect`
Function [21, 33](#)

D

Data
Access [21](#)

Database
Access [21–22](#)
Log off [21](#)
Log on [21](#)

`disconnect`
Function [21, 28, 34](#)

E

Environment variable [69](#)

Error codes [23, 71–76](#)

Example application [56–68](#)

F

File
Configuration [69](#)

INDEX

Function 19

`add_cursor_variable` 23–24, 45
`add_input_variable` 23–24, 46
`associate` 23, 48
`close` 22, 39
`commit` 28, 40
`connect` 21, 33
`disconnect` 21, 28, 34
`logoff` 21, 28, 35
`logon` 21, 36
`o2db_sql_execute` 25, 53
`open` 22, 41
`operator <<` 51
`operator >>` 50
`rollback` 28, 43
`server_error` 31

H

Host

 Connection 21, 58
 Disconnect 21, 28

`host` 69

I

Internal error codes 71

J

Java 11

L

Library

 Definition 18

logoff

 Function 21, 28, 35

logon

 Function 21

`lu_name` 69

M

Managing statements 23

`max_statements` 69

`mode` 69

N

`network` 69

O

O₂

 Architecture 10

O₂C 11

O₂Corba 11



INDEX

`o2db_Connection`
 Class [18, 32](#)
 Creation [21](#)
 Functions [21, 32–37](#)

`o2db_O2DBAccess`
 Class [18, 30](#)
 Functions [31](#)

`o2db_Session`
 Class [18, 21–22, 38](#)
 Functions [38–43](#)

`o2db_sql_execute`
 Function [25, 53](#)

`o2db_Statement`
 Class [18, 22, 44](#)
 Functions [44–54](#)

`O2DBACCESS` [69](#)

`O2DBAccess` [11](#)

`o2dbaccess`
 Schema [18](#)

`o2dbaccess.cf` [69](#)

`O2Engine` [10](#)

`O2Graph` [11](#)

`O2Kit` [11](#)

`O2Look` [11](#)

`O2ODBC` [11](#)

`O2Store` [10](#)
 Overview [13](#)

`O2Tools` [11](#)

`O2Web` [11](#)

`open`
 Function [22, 41](#)

`operator <<`
 Function [51](#)

`operator >>`
 Function [50](#)

`OQL` [11](#)

P

`port` [69](#)

R

`RDBMS`
 Querying [66](#)

`rollback`
 Function [28, 43](#)

S

`server_error`
 Error codes [71](#)
 Function [23, 31](#)

`service` [69](#)

`Session`
 Close [22, 28](#)
 Open [22](#)

`Statement`
 Close [22](#)
 Management [23](#)
 Maximum number [22](#)
 Open [22](#)
 Preparation [23](#)
 Running [25](#)

`System`
 Architecture [10](#)

T

`Table`
 Creating [61](#)
 Displaying content [64](#)
 Populating [62](#)

`type_checking` [69](#)