TOQL C++ Binding Manual

TOOBIS Deliverable T36.TR.1

University of Athens

01 Pliroforiki

O_2 Technology

February 1998

Table of Contents

C++ Binding for TOQL

This document describes the classes that can be used in a C++ program, in order to interact with a TOOBIS database, so as to submit TOQL queries and retrieve their results. These classes are used in conjunction with classes **d_Session** and **d_Transaction**, which are described in ODMG C++ Binding Reference Manual [1].

1. Class d_tDatabase

Database operations are implemented as objects of the class **d_tDatabase**. This class defines operations for opening and closing databases. Class **d_tDatabase** inherits from **d_Database**, so the operations defined in this class can be used through an instance of **d_tDatabase**, except for operations for persistent root creation, destruction and renaming, along with operation **destroy**, for database destruction. These operations are trapped, and raise exceptions if used. In order to create persistent roots, one must use TODL.

In order to use class d_tDatabase, file **“toql_db.hxx”** should be included. The public interface of class **d_tDatabase** is as follows:

```
class d_tDatabase: public d_Database {
public:
   void open(const char *dbname, access_status status = read_write);
   void close(void);
   static d_tDatabase* get_current(void);
   static void set_current(d_tDatabase *cur);
   void destroy(const char *database_name);
   void create_persistent_root(const char *root_name,
                               const char *class_name,
                               const int mode = OL_VARIABLE);
   void destroy_persistent_root(const char *root_name);
   void set_object_name(const d_Ref_Any& theObject,
                        const char *theName,
                        const char *oldName = 0);
   void rename_object(const char *oldName, const char *newName);
};
```

In the following pages, the above methods are described.

1.1 d_tDatabase::open

Summary	Opens the database.
Syntax	**int open(const char *dbname, access_status status = read_write);**
Arguments	**dbname** is the name of the database to be open. **status** may take one of the following values: 1. *read_write*, in order to retrieve **and** modify objects in the database 2. *read_only*, in which case only object retrieval is allowed. 3. *exclusive*, so as to prevent other users from accessing the database.
Description	Opens the specified database in the requested mode. A database must be opened, before its objects may be accessed. When a database is opened, it is marked as the *current database*, i.e. the database against which queries are evaluated.
Returns	0 on success. If the database does not exist or metadata information is missing or invalid, the function returns 1.
Note	An application must open all the databases it wishes to connect to *before* beginning a transaction.
Example	The following lines of code declare a variable of type **d_tDatabase** and map it to the physical database *temporalData*. The database is opened in read-write mode (default).

```
d_tDatabase temporalDb;

temporalDb.open("temporalData");
```

1.2 d_tDatabase::close

Summary Closes the database.

Syntax **void close(void);**

Arguments None

Description Closes the database. Once a database is closed, its objects may no longer be accessed through TOQL. The *close* method performs the necessary cleanup and housekeeping activities required by the TOQL processor. Note that, contrary to the homonymous method of class **d_Database**, this method is *not* a no-operation.

If the current database is closed, no database is automatically selected as current; thus TOQL queries may not be submitted, until a new database is opened (using the **open** method), or an already open database is designated as current, using the **set_current** method.

If this method is invoked for a database that is not open, no action is performed.

Returns Nothing.

Example The following lines of code declare a variable of type **d_tDatabase** and map it to the physical database *temporalData*. Finally, the database is closed.

```
d_tDatabase temporalDb;
temporalDb.open("temporalData");
...
temporalDb.close();
```

1.3 d_tDatabase::get_current

Summary Returns the current temporal database.

Syntax **d_tDatabase *get_current(void);**

Arguments None

Description Returns the current temporal database. If no database is current, NULL is returned.

Returns The current temporal database.

Example The following lines of code declare two variables of types **d_tDatabase *** and **d_tDatabase** and assign the current database to the first one.

```
d_tDatabase *curDb, temporalDb;
curDb = temporalDb.get_current();
```

1.4 d_tDatabase::set_current

Summary Sets the current database.

Syntax **void set_current(d_tDatabase *cur);**

Arguments The database to be set as current. It must designate an open database.

Description Sets the current database to be the one specified by the parameter. Once the current database has been changed, queries are directed to the new current database, rather than to the previously current.

If the designated database is not open, no action is taken and the current database is not modified.

Returns Nothing.

Example The following lines of code declare two variables of types **d_tDatabase**. Both databases are mapped to physical databases, using the **open** method, and then the first one is set as **current**, using the `set_current` method.

```
d_tDatabase Db1, Db2;
Db1.open("aTemporalDb"); // current = aTemporalDb
Db2.open("bTemporalDb"); // current = bTemporalDb
Db2.set_current(&Db1); // current = aTemporalDb
```

1.5 d_tDatabase::destroy

Summary The functionality defined in class **d_Database** for this method is to destroy a database. In class **d_tDatabase** this method is redefined and raises an exception if used.

Syntax **void destroy(const char *database_name);**

Arguments **database_name** is the name of the database to destroy.

Description This method may not be used for database destruction, and raises an exception if used.

Returns Raises an exception.

1.6 d_tDatabase::create_persistent_root

Summary The functionality defined in class **d_Database** for this method is to create a persistent root, i.e. a database entry point. In class **d_tDatabase** this method is redefined and raises an exception if used.

Syntax

```
void create_persistent_root(const
              char *root_name,
              const char *class_name,
              const int mode = OL_VARIABLE);
```

Arguments **root_name** is the name of the persistent root to create. **class_name** is the name of the class. **mode** specifies the desired mode for the persistent root.

Description This method may not be used for persistent root creation, and raises an exception if used. Persistent roots may be created only via TODL definitions (extents).

Returns Raises an exception.

1.7 d_tDatabase::destroy_persistent_root

Summary	The functionality defined in class **d_Database** for this method is to destroy a persistent root. In class **d_tDatabase** this method is redefined and raises an exception if used.
Syntax	**void destroy_persistent_root(const char *root_name);**
Arguments	**root_name** is the name of the persistent root to destroy.
Description	This method may not be used for persistent root destruction, and raises an exception if used.
Returns	Raises an exception.

1.8 d_tDatabase::set_object_name

Summary	The functionality defined in class **d_Database** for this method is to assign a persistent name to an object. In class **d_tDatabase** this method is redefined and raises an exception if used.
Syntax	**void set_object_name(const d_Ref_Any& theObject, const char *theName, const char *oldName = 0);**
Arguments	**theObject** is the name of the object to which a name is to be assigned; **theName** is the name to be assigned to the object; **oldName** should specify the current object name, if any, or should be omitted for currently unnamed objects.
Description	This method may not be used for persistent name assignment, and raises an exception if used.
Returns	Raises an exception.

1.9 d_tDatabase::rename_object

Summary	The functionality defined in class **d_Database** for this method is to rename an object. In class **d_tDatabase** this method is redefined and raises an exception if used.
Syntax	**void set_object_name(const char *oldName, const char *newName);**
Arguments	**oldName** is the current name of the object; **newName** is the new name for the object.
Description	This method may not be used for object renaming, and raises an exception if used.
Returns	Raises an exception.

2. Class d_TOQL_Query

This chapter describes the class **d_TOQL_Query**, which can be used to execute a TOQL query from a C++ application. The following functions and class public interface are described:

- **d:TOQL_Query::d_TOQL_Query**
- **d_TOQL_Query::operator=**
- **d_TOQL_Query::operator<<**
- **d_toql_execute**
- **d_toql_iterate**
- **TOQL_error_message**

To use TOQL, the file "toql_qry.hxx" should be included. The public interface for class d_TOQL_Query is as follows:

```
class d_TOQL_Query: {
public:
    d_TOQL_Query(void);
    d_TOQL_Query(const char *s);
    d_TOQL_Query(const d_String &s);
    d_TOQL_Query(const d_TOQL_Query &q);
    d_TOQL_Query &operator=(const char *s);
    d_TOQL_Query& operator=(const d_TOQL_Query q);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, const char *p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, const d_String
                                   &p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, char p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, unsigned char p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, short p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, unsigned short
                                   p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, int p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, unsigned int p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, long p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, unsigned long p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, float p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, double p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, const Interval
                                   &p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, const Instant
                                   &p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, const Period &p);
    friend d_TOQL_Query &operator<<(d_TOQL_Query &q, const Period_Set
                                   &p);
};
template<class T> int d_toql_execute(d_TOQL_Query &q, T &result);
template<class T> int d_toql_iterate(d_TOQL_Query &q, T &result);
const char *TOQL_error_message(void);
```

2.1 d_TOQL_Query::d_TOQL_Query

Summary	Creates an object of the class **d_TOQL_Query**
Syntax	**d_TOQL_Query(void);** **d_TOQL_Query(const char *s);** **d_TOQL_Query(const d_String &s);** **d_TOQL_Query(const d_TOQL_Query &q);**
Arguments	**s** is a string, or *char* * which contains the text of the query. **q** is an object of the class **d_TOQL_Query**, to be copied into the newly created object.
Description	When the constructor is used with no parameters, a query with empty query text is created. The query text may be changed afterwards, using the **=** operator. If a string, or a or *char* * is used as a parameter to the constructor, the query text is set to the current value of the parameter. When the constructor is called with a parameter of type **d_TOQL_Query**, both the query text *and* the arguments of the parameter are copied into the newly created object.
Returns	The query object.
Example	The following lines of code declare a variable of type **d_TOQL_Query** with some text and afterwards create an exact copy of this variable into a second one.

```
d_TOQL_Query q1(“select Name from Products”);
d_TOQL_Query q2(q1);
```

2.2 d_TOQL_Query::operator=

Summary	Changes the query text of a query object or copies another query object into it.
Syntax	**d_TOQL_Query& operator=(const char *s);** **d_TOQL_Query& operator=(const d_TOQL_Query &q);**
Arguments	**s** is a *char ** which contains the text of the query. **q** is an object of the class **d_TOQL_Query**, to be copied into the left-hand-side object of the assignment.
Description	If the right-hand-side of the assignment is of type *char **, the query text is set to the value of the parameter and any arguments currently specified for the query are cleared. If the right-hand-side of the assignment is a parameter of type **d_TOQL_Query**, both the query text *and* the arguments of the parameter are coppied into the query object.
Returns	The query object.
Example	The following lines of code declare a variable of type **d_TOQL_Query** and use the **=** operator to assign some query text to it.

```
d_TOQL_Query q1;
q1 = "select Name from Products";
```

2.3 operator<<

Summary Used to pass the value of the specified argument as a parameter to the query object.

Syntax

```
d_TOQL_Query &operator<<(d_TOQL_Query &q, const char *p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, const d_String &p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, char p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, unsigned char p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, short p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, unsigned short p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, int p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, unsigned int p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, long p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, unsigned long p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, float p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, double p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, const Interval &p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, const Instant &p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, const Period &p);

d_TOQL_Query &operator<<(d_TOQL_Query &q, const Period_Set &p);
```

Arguments **q** is an object of the class **d_TOQL_Query**, which will be supplied with the parameter.

p is the parameter.

Description The value of the parameter is appended to the parameter list of the query. The parameters must be given with this operator in the order corresponding to $1, $2, ..., $n.

Returns The query object.

Note The **<<** operator is not supported for types **d_Date**, **d_Time**, **d_Timestamp** and **d_Interval**. The TOOBIS classes **Interval**, **Instant**, **Period** and **Period_Set** should be used instead. Additionally, the **<<** operator is not supported for types **const d_Ref<T>** and **const d_Collection<T>**.

Example The following lines of code declare a variable of type **d_TOQL_Query** with some query text, and supplies it with two arguments:

```
d_TOQL_Query q1("select $1 * e.salary as NewSalary\
                from Employees as e\
                where e.name = $2");
int raise; char emp_name[20];
...
q1 << raise << emp_name;
```

2.4 d_toql_execute

Summary Executes a TOQL query against the current database.

Syntax **int d_toql_execute(d_TOQL_Query &q, T &result);**

Arguments **q** is an object of the class **d_TOQL_Query**, which holds the query to be executed.

result is a variable into which the query result will be stored. The type of the variable must match the type of the query result.

Description This function parses and evaluates the query in the **d_TOQL_Query** parameter and stores the outcome into the parameter **result**. If the query text references parameters (*$1*, *$2* and so on), these must be provided to the query before execution. Each execution resets the query parameters.

Before a query is submitted, a session to the DBMS must have been established and a database should have been opened.

Returns 0 if the query is correct; otherwise it returns a non-zero value.

An exception may be raised if an error occurs during the execution of the query or the instantiation of the result.

Example The following lines of code declare a variable of type **d_TOQL_Query**, and use the function **d_toql_execute** to evaluate it and store the results into a program variable.

```
d_TOQL_Query q1(“select Name from Products”);
d_Bag<d_String> result;
d_toql_execute(q1, result);
```

2.5 d_toql_iterate

Summary Executes a TOQL query against the current database, and returns an iterator for the result.

Syntax **int d_toql_iterate(d_TOQL_Query &q, d_Iterator<T>& result);**

Arguments **q** is an object of the class **d_TOQL_Query**, which holds the query to be executed.

result is an iterator which will be used to iterate over the query result. The query must return a *bag* and the iterator's base type (T) must match the type of the bag's elements.

Description This function parses and evaluates the query in the **d_TOQL_Query** parameter. The result is not returned in a program variable, as in **d_toql_execute**, but rather an iterator which will point to each element of the result in turn is instantiated properly. If the query text references parameters (*$1*, *$2* and so on), these must be provided to the query before execution. Each execution resets the query parameters.

Before a query is submitted, a session to the DBMS must have been established and a database should have been opened.

Returns 0 if the query is correct; otherwise it returns a non-zero value.

Example The following lines of code declare a variable of type **d_TOQL_Query**, and use the function **d_toql_iterate** to evaluate it and define an iterator over the result.

```
d_Iterator<d_String> p_name;
d_TOQL_Query q1("select Name from Products");
d_toql_execute(q1, p_name);
```

2.6 TOQL_error_message

Summary Retrieves a string containing the error message, corresponding to the evaluation of the last TOQL query.

Syntax **const char *TOQL_error_message(void);**

Arguments None.

Description This function retrieves a textual description of the error that (may have) occured during the execution of the last TOQL query. The text is reset at each TOQL query execution.

Returns The error text.

Example The following lines of code declare a variable of type **d_TOQL_Query**, and use the function **d_toql_execute** to evaluate it. Since the query text is not a valid TOQL query, execution will fail and the function TOQL_error_message is used to retrieve the error text and display it to the user.

```
d_TOQL_Query q1(“erroneous query text”);
d_Bag<d_String> result;
d_toql_execute(q1, result);
cout << TOQL_error_message();
```

3. An example program

Standard C++ includes

```
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <stdlib.h>
```

O2 system includes

```
#include <rt_odb_Session.hxx>
#include <rt_odb_Transaction.hxx>
```

TOQL includes

```
#include <toql_db.hxx>
#include <toql_qry.hxx>
```

Application includes

```
#include <Factories.hxx>
```

Variables for database connection

```
d_Session session;
d_tDatabase database;
d_Transaction transaction;
```

```
int main(int argc, char *argv[])
{
```

Connect to O2

```
if (session.begin(argc, argv, “di”,
      “pileus”, “/opt/o2”))
{
      cerr << “Cannot start o2” << endl;
      exit(1);
}
```

Open database

```
if (database.open(“temporal_db”) != 0)
{
      cerr << “Cannot open db” << endl;
      session.end();
      exit(1);
}
```

Begin transaction

```
transaction.begin();
```

Variables for querying

```
d_TOQL_Query toql_query(“select f from Factories as f”);
d_bag<Factory> result;
```

Query database and process results

```
if (d_toql_execute(toql_query, result) != 0)
{
      cerr << “Error: “ << TOQL_error_message() << endl;
      exit(1);
}
else
      PrintFactories(result);
```

Disconnect

```
database.close();
session.end();
```

```
return 0;
}
```