The OCCI Object Model, Object Serialization and the Object Cache

Technical Whitepaper
Introduction

Oracle C++ Call Interface (OCCI) is a easy to use and powerful C++
interface to access the Oracle object-relational database. The OCCI Objects
interface is a seamless and optimized mechanism to access and manipulate
persistent data from the database as C++ objects. With the Objects interface
and navigational access paradigm, there is a single object model for the
application and the database. C++ application developers do not have to
contend with relational tables, normalization, SQL and the dreaded object-
relational impedance mismatch. Developers do not need to implement any
additional code for mapping the database objects to C++ class objects. This is
done by OCCI by compile-time generated code and runtime support.
Combining Oracle’s object-relational features, OCCI and C++, complex and
powerful object-oriented database applications can be developed.

This technical whitepaper aims to explain the underlying working of the
Objects interface in OCCI and is intended for advanced OCCI developers.
Readers are assumed to be familiar with OCCI and C++ object-oriented
programming.

This document uses the term database objects to refer to persistent data stored in
the database.

Steps in developing a Objects application with OCCI

- Design the object model
- Realize the object model by creating object types and object tables in
  the Oracle database
- Use the OTT tool to generate C++ classes corresponding to the object
types. A registration function is also generated by OTT
- Include the C++ class declarations generated by OTT in the
  application source code and use the classes as any other C++ class.
  Applications can choose to use either the navigational access model
  and/or the associative access model to retrieve and work with database
  objects.
- Compile and link with the OCCI library

Navigational and Associative access models

OCCI provides 2 different paradigms for accessing and manipulating objects
in the database. In the navigational access model, references to objects in the
database are fetched to the client and a client-side object cache transparently fetches the objects from the database when the references are dereferenced. Relationships between objects are modeled using references (as attributes) and the application can navigate and de-reference associated objects in the relationship. In an application using the navigational access model, the database becomes “invisible” and applications appear to be “pure” C++ with very minimal SQL.

In the **associative access** model, SQL queries and DML statements are executed to fetch and update objects in the database. There is no role of the object cache in the associative access model.

**Client-side Object Cache**

In OCCI’s navigational access model, a **client-side object cache** caches and manages database objects (represented as C++ objects) as and when references are de-referenced and objects are created/updated/deleted. When a reference is de-referenced by the application, the object cache searches for the referenced object in the cache and returns the cached object if present. If the object is not found in the cache, the object is fetched from database. The object cache does all this transparently. This dereference process is also termed **swizzling** or **pinning** in database terminology. When a transaction is committed by the OCCI application, the object cache **flushes** all the “dirty” (new/updated/deleted) objects to the database where the corresponding actions are reflected in the persistent store.

The OCCI object cache provides **transactional consistency** of objects – an object is cached for the duration of a transaction. After a current transaction is committed (or rolled back), the next de-reference deletes the old copy of the object from the cache (if present) and the object is refreshed from the database server.

**The object cache maintains a pointer to the C++ object instance representing the corresponding database object.**

**Object Cache Memory Management**

When a reference is dereferenced, the object cache **pins** the object in the cache and returns a pointer to the object. The object can then be accessed and manipulated by the application. A object is kept pinned in the cache till there are ‘active’ reference variables to the object. In a OCCI program, a reference is represented by a `Ref<T>` class instance and the lifetime of the `Ref<T>` variable determines the duration the corresponding object is kept pinned in the cache. Associated with a cached object is the **pin count** tracking the active number of reference variables to the object. The code snippet below gives an example:-
void foo()
{
    //References are represented by the OCCI Ref<T> class
    Ref<CEmployee> r1,r2;
    //execute a query and get initial references
    r1 = rs->getRef(1);
    string ename = r1->getEName();//object is pinned and pin count = 1
    r2 = r1;//one more reference to the same object, pin count = 2
    //when r1 and r2 go out of scope as the function returns, the
    //pin count of the object is decrement by 1 each and pin count
    //reaches 0. The object still remains in the cache but is now
    //eligible for garbage collection.
}

A object with a pin count of 0 is eligible for garbage collection. When the
memory usage of the object cache reaches the maximum size, the garbage
collector frees objects with a pin count of 0 using a LRU(least recently used)
algorithm until the memory usage of the cache drops down to the optimal size.
The maximum and optimal sizes of the cache can be configured by the
application by calling the setCacheMaxSize() and setCacheOptSize() methods of
the Environment class.

Object Type Translator (OTT)

OTT is the command-line tool to generate C++ class declarations and
definitions for the object types in the database object model. For each object
type in the database, OTT generates a C++ class with member variables
Corresponding to the object type attributes. Methods for serialization & de-
serialization and object management are also generated in the C++ class.

Mapping of object attributes to OCCI/C++ types

The following table presents the mapping from the type of a attribute of a
object type to the datatype of the corresponding member variable in the OTT
generated C++ class:-

<table>
<thead>
<tr>
<th>Attribute type in Database</th>
<th>OCCI/C++ datatype of member variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR/NCHAR/VARCHAR2/NVARCHAR2</td>
<td>C++ STL string</td>
</tr>
<tr>
<td>CLOB/NCLOB</td>
<td>Clob</td>
</tr>
<tr>
<td>BLOB</td>
<td>Blob</td>
</tr>
<tr>
<td>BFILE</td>
<td>Bfile</td>
</tr>
<tr>
<td>NUMBER</td>
<td>Number</td>
</tr>
<tr>
<td>BINARY_FLOAT</td>
<td>BFloat</td>
</tr>
<tr>
<td>BINARY_DOUBLE</td>
<td>BDouble</td>
</tr>
<tr>
<td>DATE</td>
<td>Date class</td>
</tr>
</tbody>
</table>
Object Serialization and de-serialization

Objects are stored in object tables in the Oracle database as a byte stream. This byte stream is typically termed the object image. The OCCI application retrieves and works only with C++ class instances. The process of converting from the object image to the clientside C++ object is called de-serialization (or marshalling). The process of converting a clientside application C++ object instance to an object image is called serialization (or unmarshalling).

Serialization is necessary when flushing a dirty object (new/modified/deleted) to the database and de-serialization is done when retrieving an object from the database.

OCCI Object Model

The application C++ classes generated by OTT derive from OCCI’s PObject class. The PObject class is “known” to the OCCI framework and hence the framework can access and work with the application’s C++ classes using C++ polymorphism. Object construction, initialization and management are provided by PObject class methods. For example, after a object is retrieved from the database and it’s attribute(s) modified at the client, the PObject::markModified() needs to be called to indicate to the object cache that the object has been modified and needs to be flushed to the database at transaction commit. Overloaded operator new methods are provided on PObject to construct new objects in the database.

OCCI AnyData class

The OCCI AnyData class provides an easy and natural interface for accessing and for constructing an object image stream. The getXXX() methods of AnyData are called to read/access the next attribute from the object image (during de-serialization) and the setXXX() methods are called to set/write the next.
**attribute** in the object image (during serialization). It is the caller’s responsibility to ensure the appropriate `getXXX()` or `setXXX()` method for the next attribute’s datatype is called. Hence, the caller needs to have some kind of object type metadata during serialization/de-serialization. In OTT generated C++ classes, the serialization and de-serialization methods have the code generated to call the appropriate `getXXX()`/`setXXX()` methods in the same sequence as the attributes in the object type.

<table>
<thead>
<tr>
<th>Attribute type in Database</th>
<th>AnyData class method for de-serialization / serialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR/NCHAR/VARCHAR2/NVARCHAR2</td>
<td><code>getString/setString</code></td>
</tr>
<tr>
<td>CLOB/NCLOB</td>
<td><code>getClob/setClob</code></td>
</tr>
<tr>
<td>BLOB</td>
<td><code>getBlob/setBlob</code></td>
</tr>
<tr>
<td>BFILE</td>
<td><code>getBfile/setBfile</code></td>
</tr>
<tr>
<td>NUMBER</td>
<td><code>getNumber/setNumber</code></td>
</tr>
<tr>
<td>BINARY_FLOAT</td>
<td><code>getBFloat/setBFloat</code></td>
</tr>
<tr>
<td>BINARY_DOUBLE</td>
<td><code>getBDouble/setBDouble</code></td>
</tr>
<tr>
<td>DATE</td>
<td><code>getDate/setDate</code></td>
</tr>
<tr>
<td>TIMESTAMP/TIMESTAMP WITH TIME ZONE / TIMESTAMP WITH LOCAL TIME ZONE</td>
<td><code>getTimestamp/setTimestamp</code></td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td><code>getIntervalDS/setIntervalDS</code></td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td><code>getIntervalVM/setIntervalVM</code></td>
</tr>
<tr>
<td>REF object type</td>
<td><code>getRef/setRef</code> (Takes a <code>RefAny</code>)</td>
</tr>
<tr>
<td>object type (embedded object)</td>
<td><code>getObject/setObject</code></td>
</tr>
<tr>
<td>VARRAY/NESTED TABLE of type</td>
<td><code>getVector/setVector with the appropriate vector&lt;T&gt;</code>. The <code>getVector/setVector</code> methods are strictly not member methods of the <code>AnyData</code> class and are global functions that take the <code>AnyData</code> instance as the first parameter.</td>
</tr>
</tbody>
</table>

Object Serialization and de-serialization in OCCI

In the OCCI object model, it is each class’s responsibility to serialize and de-serialize objects of its own type. This is achieved by the OTT generated `readSQL` and `writeSQL` methods in each C++ class. There are 2 versions of the `readSQL` and `writeSQL` methods: static and member. In the OTT generated registration function, the static methods are “registered” with the OCCI framework, thereby mapping an object type to its C++ class and then to its static `readSQL/writeSQL` methods.
The OCCI framework calls the static `readSQL` method with some context when a object is to be de-serialized (from a object image). The `readSQL` method constructs a new C++ object instance and calls the member `readSQL` method on the new instance. The member `readSQL` method reads the attributes from the image using the `AnyData` class and populates the new object’s member variables. The new object instance is returned back to the framework where it is added to the object cache.

On the same lines, the OCCI framework calls the static `writeSQL` method with the `dirty` object to be serialized (into a object image) and some context. The static `writeSQL` calls the member `writeSQL` method. The member `writeSQL` method writes the values of the member variables (attributes) and forms the object image using the `AnyData` class methods.

The above design is “recursive” : embedded object members are serialized/de-serialized similarly. The `AnyData::setObject()` and `AnyData::getObject()` methods directly call the embedded object’s member `writeSQL()` and `readSQL()` methods respectively (C++ polymorphism using virtual methods) and the embedded object attribute is written to the object image or read from the image. For a attribute of type : collection of objects, the `setVector/getVector` functions call the `writeSQL/readSQL` method for each object in the collection.

A couple of examples are presented to explain the OTT generated C++ classes and serialization/de-serialization.

Example 1 :-

```sql
create or replace type TAddress as object
(
    Line1    Varchar2(100),
    Line2    Varchar2(100),
    Line3    Varchar2(100)
)

create or replace type TEmployee;
/

create or replace type TDepartment as object
(
    DName   Varchar2(100),
    DMgr    REF TEmployee
)
/

create or replace type TPhoneList as varray(10) of varchar2(20)
create or replace type TAward as object
(
    Citation    Varchar2(100),
    AwardDate   Date
)

create or replace type TAwardList as varray(100) of TAward;
```
create or replace type TEmployee as object
{
  EmpId       Number(10),
  EName       VARCHAR2(100),
  Dept        Ref TDepartment,
  Addr        TAddress,
  PhoneNos    TPhoneList,
  Awards      TAwardList
}

The classes generated by OTT are CAddress, CAward, CDepartment and CEmployee.

class CAddress : public oracle::occi::PObject {
private:
  OCCI_STD_NAMESPACE::string LINE1;
  OCCI_STD_NAMESPACE::string LINE2;
  OCCI_STD_NAMESPACE::string LINE3;
}

Attribute                C++ class member variable AnyData method called in readSQL/writeSQL Comments
LINE1                   string           (getString/setString
LINE2                   string            getString/setString
LINE3                   string            getString/setString

class CDepartment : public oracle::occi::PObject {
private:
  OCCI_STD_NAMESPACE::string DNAME;
  oracle::occi::Ref< CEmployee > DMGR;
}

Attribute                C++ class member variable AnyData method called in readSQL/writeSQL Comments
DNAME                    string            getString/setString
DMGR                     Ref<CEmployee>   getRef/setRef  Reference type

class CAward : public oracle::occi::PObject {
private:
  OCCI_STD_NAMESPACE::string CITATION;
  oracle::occi::Date AWARDDATE;
}

Attribute                C++ class member variable AnyData method called in readSQL/writeSQL Comments
CITATION                 string            getString/setString
AWARDDATE                Date              getDate/GetDate

class CEmployee : public oracle::occi::PObject {
private:

The OCCI Object Model, Object Serialization and the Object Cache Page 8
Example 2 :-

The OTT generated code is annotated with underlined comments to explain the working of readSQL and writeSQL methods.

Object Type :-

create or replace type Employee as object

{
    Id Number(10),
    Name Varchar2(100),
    Salary Number(10,2)
};

OTT Generated C++ class

class CEmployee : public oracle::occi::PObject {

private:

    oracle::occi::Number ID;
    OCCI_STD_NAMESPACE::string NAME;
    oracle::occi::Number SALARY;

public:

    static void *readSQL(void *ctxOCCI_);
    virtual void readSQL(oracle::occi::AnyData& streamOCCI_);
    static void writeSQL(void *objOCCI_, void *ctxOCCI_);
virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

readSQL and writeSQL methods

void *CEmployee::readSQL(void *ctxOCCI_) static
{
    CEmployee *objOCCI_ = new(ctxOCCI_) CEmployee(ctxOCCI_); construct new instance
    oracle::occi::AnyData streamOCCI_(ctxOCCI_);
    try
    {
        if (streamOCCI_.isNull())
            objOCCI_->setNull();
        else
            objOCCI_->readSQL(streamOCCI_); call the member method
    }
    catch (oracle::occi::SQLException& excep)
    {
        delete objOCCI_;
        excep.setErrorCtx(ctxOCCI_);
        return (void *)NULL;
    }
    return (void *)objOCCI_; returned to OCCI framework for the object cache
}

void CEmployee::readSQL(oracle::occi::AnyData& streamOCCI_)
{ the getXXX() calls are in the same sequence as the attributes of the object type
    ID = streamOCCI_.getNumber(); read the number attribute
    NAME = streamOCCI_.getString(); read the varchar2 attribute
    SALARY = streamOCCI_.getNumber(); read the number attribute
}

void CEmployee::writeSQL(void *objectOCCI_, void *ctxOCCI_) static
{
    CEmployee *objOCCI_ = (CEmployee *) objectOCCI_; object to be flushed
    oracle::occi::AnyData streamOCCI_(ctxOCCI_);
    try
    {
        if (objOCCI_->isNull())
            streamOCCI_.setNull();
        else
            objOCCI_->writeSQL(streamOCCI_); call the member method
    }
    catch (oracle::occi::SQLException& excep)
    { excep.setErrorCtx(ctxOCCI_);
        return;
    }
}

void CEmployee::writeSQL(oracle::occi::AnyData& streamOCCI_)
{ the setXXX() calls are in the same sequence as the attributes of the object type
    streamOCCI_.setNumber(ID); write the number attribute
    streamOCCI_.setString(NAME); write the varchar2 attribute
    streamOCCI_.setNumber(SALARY); write the number attribute
}
The OCCI Object Model, Object Serialization and the Object Cache.
April 2007

Author: Shankar Iyer

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Copyright © 2007, Oracle. All rights reserved.
This document is provided for information purposes only and the contents hereof are subject to change
without notice. This document is not warranted to be error-free, nor subject to any other warranties or
conditions, whether expressed orally or implied in law, including implied warranties and conditions of
merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this
document and no contractual obligations are formed either directly or indirectly by this document. This
document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for
any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks
of their respective owners.