Introduction to OCCI
and a bit more....

S. Ponce CERN IT DSS
Foreword

- **OCCI = ORACLE C++ Call Interface**
  - i.e. the ORACLE interface for C++ programs

- **Small layer on top of OCI (the C interface)**
  - See Andrea's talk
  - Note that OCI can also be used
    - But you will hopefully prefer OCCI

- **Object oriented**
  - Very close to Java API
    - See Slava's talk
Overall program

- OCCl for beginners
  - Basic use
  - PL/SQL
  - Queries and ResultSets
- Prepared statements
  - Input and output parameters
- Bulk access
  - Without object mode
  - With object mode
Basic OCCI Use

#include <occi.h>
using namespace oracle::occi;

...

Environment *env = Environment::createEnvironment
             (Environment::DEFAULT);
Connection *conn = env->createConnection(user, pwd, db);

...

Statement *stmt = conn.createStatement();
stmt->execute("INSERT INTO TestTable VALUES ('bla',123)");

...

conn->commit();
conn->terminateStatement(stmt);
Please add Error handling!
Basic OCCI Use - compilation

- Install Oracle
  - e.g. instantclient version
    - RPMs from ORACLE web site
    - .deb from RPMS with alien
  - Do not forget the devel package

- Configure your /etc/tnsnames.ora
  - Copy if from /afs/cern.ch/project/oracle/admin

```
g++ -o basic basic.cpp
   -I /usr/include/oracle/11.2/client64
   -L/usr/lib/oracle/11.2/client64/lib
   -l occi -l clntsh

g++ --version  \rightarrow  4.1.2, 4.4.6, 4.5.3, 4.6.2, 4.7.0
```
Basic OCCI Use

```cpp
#include <occi.h>
using namespace oracle::occi;

...

Environment *env = Environment::createEnvironment
    (Environment::DEFAULT);
Connection *conn = env->createConnection(user, pwd, db);

...

Statement *stmt = conn.createStatement();
stmt->execute("INSERT INTO TestTable VALUES ('bla',123)"");

...

conn->commit();
conn->terminateStatement(stmt);
```
#include <occi.h>
using namespace oracle::occi;

Environment *env = Environment::createEnvironment
  (Environment::DEFAULT);
Connection *conn = env->createConnection(user, pwd, db);

Statement *stmt = conn.createStatement();
stmt->execute("INSERT INTO TestTable VALUES ('bla',123)");

conn->commit();
conn->terminateStatement(stmt);
Basic OCCI Use

```
#include <occi.h>
using namespace oracle::occi;

Environment *env = Environment::createEnvironment (Environment::DEFAULT);
Connection *conn = env->createConnection(user, pwd, db);

Statement *stmt = conn.createStatement();
stmt->execute("INSERT INTO TestTable VALUES ('bla',123) ");

conn->commit();
conn->terminateStatement(stmt);
```
Basic OCCI Use

```
#include <occi.h>
using namespace oracle::occi;

...

Environment *env = Environment::createEnvironment
    (Environment::DEFAULT);
Connection *conn = env->createConnection(user, pwd, db);

...

Statement *stmt = conn.createStatement();
stmt->execute("INSERT INTO TestTable VALUES ('bla',123)");

...

conn->commit();
conn->terminateStatement(stmt);
```

Preparation

Create a connection

Interesting part

Run a statement

Validation/closing

Commit and close
• No difference with SQL statements
  – But for the ';' at the end....
    • PLS-00103: Encountered the symbol "end-of-file" when expecting …
    • Note that if you put one in SQL it's ORA-00911: invalid character

Statement *stmt = conn.createStatement();

ResultSet *rset = stmt.executeUpdate("BEGIN myProc(1,2,3); END;");

stmt->closeResultSet (rset);
Statement *stmt = conn.createStatement();

ResultSet *rset = stmt.executeQuery("SELECT name, value FROM TestTable");

while (rset->next()) {
    std::cout << "row name : " << rset->getString(1) << ", row value : " << rset->getInt(2) << std::endl;
}

stmt->closeResultSet (rset);
Prepared Statement

• Idea :
  – Avoid reparsing statements that are reused
  – Prevent SQL injections :
    ```sql
    execute('SELECT * FROM T WHERE id=' + idFromUser)
    ```
    What if idFromUser is “0; DELETE FROM T” ?

• Practically :
  – Define statement with placeholders
  – Call them several times with different values

• The way to go
  – NEVER use non prepared statements
Prepared Queries

Statement *stmt = conn->createStatement
("SELECT name, value FROM TestTable
WHERE value=:1");

for (int value = 0 ; value < 10; value++) {
    stmt->setInt(1, value);
    ResultSet *rset = stmt->executeQuery();
    while (rset->next()) {
        cout << rset->getString(1) << " : " << rset->getInt(2) << endl;
    }
    stmt->closeResultSet (rset);
}
CREATE PROCEDURE testOutParams
    (name IN VARCHAR2, value OUT INTEGER,
     description OUT VARCHAR2, data OUT SYS_REFCURSOR);

Statement *stmt = conn->createStatement
    ("BEGIN testOutParams(:1,:2,:3,:4); END;"");
stmt->registerOutParam(2, OCCIINT);
stmt->registerOutParam(3, OCCISTRING, 200);
stmt->registerOutParam(4, OCCICURSOR);

stmt->setString(1, name);
stmt->executeUpdate();

int value = stmt->getInt(2);
std::string description = stmt->getString(3);
ResultSet *rset = stmt->getCursor(4);
Commit and autocommit

```cpp
Statement *stmt = conn.createStatement
    ("UPDATE TestTable SET value=:1 WHERE name=:2");

for (...) {
    stmt->setInt(1, value);
    stmt->setString(2, name);
    stmt->execute();
}
stmt.commit();

OR

stmt->setAutoCommit(true);
for (...) {
    stmt->setInt(1, value);
    stmt->setString(2, name);
    stmt.execute();
}
```
Bulk access

• Idea:
  – Avoid network looping with the db
  – Send/Receive all rows in one go

• Practically:
  – Send/receive buffers/vectors of data
CREATE PACKAGE mypack AS
  TYPE numList IS TABLE OF NUMBER INDEX BY binary_integer;
END;
CREATE PROCEDURE testBulk (myvalues IN mypack.numList);

Statement *stmt = conn->createStatement
  ("BEGIN testBULK(:1); END;");

unsigned char (*buffer)[21]= (... calloc(nb, 21);
ub2 *lens=(ub2 *)malloc (sizeof(ub2)*nb);
ub4 unused = nb;

for (... counter= ... value = ...) {
  Bytes b = Number(value).toBytes();
  b.getBytes(buffer[counter],b.length());
  lens[i] = b.length();
}

stmt->setDataBufferArray(1, buffer, OCCI_SQLT_NUM,
  Nb, &unused, 21, lens);
stmt->executeUpdate();
free(....)
create type numList as table of integer;
create procedure testBulk (id in numList);

environment *env = environment::createEnvironment (environment::OBJECT);
...
statement *stmt = conn.createStatement ("BEGIN testBULK(:1); END;");

std::vector<Number> values;
for (... value = ...)
    { values.push_back(Number(value)); }  

setVector(stmt, 1, values, "NUMLIST");
stmt->executeUpdate();

Warning: case sensitive and case ids decided by ORACLE!
Why not using object mode?

- Brings a whole ORACLE library dealing with object in the game
  - This library has been problematic for performance
- BUT
  - The problems were with real object code
  - So we are right now testing the use of setVector “alone”
- Conclusion
  - it's too early to conclude
That's it

Enjoy OCCI programming
That's it

Enjoy OCCI programming

Well, actually, that's not all of it, there is much more...
What I'm not covering

- Clobs, Blobs
  - How to use large (binary) data
- Connection pooling
  - How to share a set of connections
- Introspection
  - How to avoid the numbers for parameters
- Advanced object usage
  - How to store/retrieve objects from the DB
- Much more....

See Oracle Documentation