

# Oracle Warehouse Builder 10g

Embed Oracle Warehouse Builder  
in your applications using scripting

*An Oracle White Paper  
February 2004*

# Embed Oracle Warehouse Builder in your applications using scripting

Introduction .....	3
Defining OMB Scripting and Interface.....	3
The Reason Behind OMB .....	3
OMB the Language .....	4
Describing OMB*Plus .....	5
JDeveloper Integration .....	6
Business Benefits of OMB Scripting .....	6
Details and Examples .....	7
Starting with OMB Scripting.....	7
Creating and Updating Metadata .....	8
Automating Metadata Changes.....	12
Performing Generic Tasks.....	13
Conclusion.....	18

# Embed Oracle Warehouse Builder in your applications using scripting

## INTRODUCTION

Oracle Warehouse Builder is the only enterprise integration design tool that manages the full life cycle of Business Intelligence (BI) data and metadata for the Oracle database. As such the tool has extensive capabilities to deal with changes in the entire BI environment to allow an efficient and effective maintenance of both metadata and data.

As part of that offering Warehouse Builder has a language that can be used to manipulate the metadata in an efficient way. This language is called OMB (Oracle MetaBase) Scripting. With it comes a user interface called OMB\*Plus allowing you to work with the scripting language.

Using a scripting language enables you to automate tasks and to perform “click intensive” tasks in a more efficient manner by creating a script that perform the changes without user interaction. This way patching your application or mass updates to your applications can be done very efficiently. Repetitive tasks such as installations or deployment of code can be automated and scheduled.

This paper will discuss OMB Scripting and its architecture and will then show some of the power of this feature with detailed examples.

## DEFINING OMB SCRIPTING AND INTERFACE

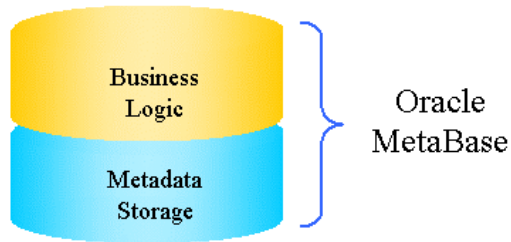
The scripting component of Warehouse Builder is a complete language and user interface that allows you to create commands performing actions on the Warehouse Builder metadata. The scripting environment consists of two components:

- OMB Scripting – Scripting is based on Java TCL. OMB scripting gives you access to the repository without using the Warehouse Builder user interface
- OMB\*Plus – The user interface for OMB Scripting

## The Reason Behind OMB

Essentially OMB Scripting and OMB\*Plus are to the Warehouse Builder repository what SQL and SQL\*Plus are to the Oracle database.

One of the key components of Warehouse Builder is the Oracle MetaBase (OMB which leads to the name for the scripting language and associated tool), which is the metadata management or business logic layer combined with the physical metadata storage layer.



**Figure 1 The Oracle MetaBase**

The MetaBase stores and manages the metadata that is created and used in Warehouse Builder. The typical access path to this metadata is via a graphical user interface, which makes complex tasks easy to understand and shields the complexity of granular activities from the user. This makes the Warehouse Builder user interface ideal for non-repetitive and complex tasks or for those getting started with the tool.

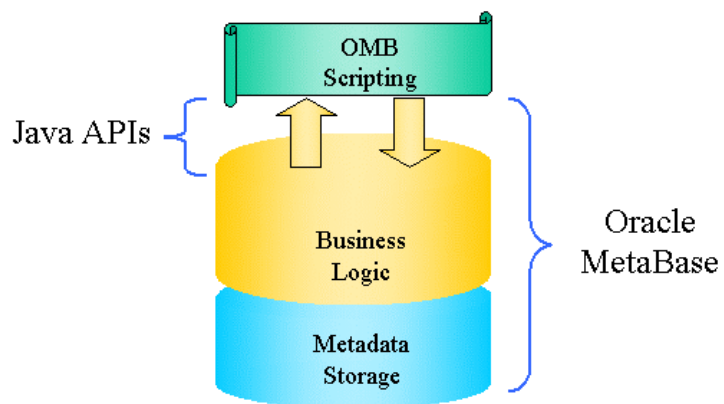
To explain this effect, you can compare the novice user of MS Excel who becomes more aware of the capabilities of Excel if he works with for a longer time. This user will find ways to make his work more productive, and he starts writing macros or little programs.

As a user becomes more familiar with Warehouse Builder and its concepts the need for a more efficient medium begins to arise. In other words, you are now an expert user looking to speed up certain tasks using “macros” or programs. This is exactly the reason OMB scripting is introduced.

### **OMB the Language**

Now that you understand why Warehouse Builder introduced a programming language, we will take a closer look at the language itself and the architecture behind it.

The OMB scripting language operates on the public Java APIs that are available on the MetaBase. The language masks the direct Java calls to the business logic with a 4GL language.



**Figure 2 Scripting on top of the MetaBase APIs**

Each of the OMB commands is pushed through the API layer into the MetaBase and produces the requested results and feedback.

The language itself has two components:

- Metadata Definition Language (MDL) – which allows you to define object structures in the MetaBase. With MDL you are working at the meta metadata level defining constructs to hold design metadata. This can be compared to DDL in the relational database
- Metadata Manipulation Language (MML) – which allows you to change the content of metadata objects. MML allows you to create definitions for tables, mappings, process flows etc. You can compare this with DML in the relational database

The scripting language is a set of Warehouse Builder extensions on top of Java TCL<sup>1</sup>. Java TCL brings a cross platform, easy to use language, which allows you to transport your scripts across environments. You can develop your OMB scripts on Windows and then run them on the production UNIX machine without changes. This openness and portability is in line with the overall Warehouse Builder strategy and makes TCL such a valuable platform. By embedding TCL commands within OMB applications you can even combine the power of both languages for your application.

### Describing OMB\*Plus

OMB\*Plus is a utility that comes with the Warehouse Builder client installation and appears, on Windows environments, in your Warehouse Builder program group. The function of OMB\*Plus is to create an environment that gives convenient access to the OMB language.

Within OMB\*Plus you receive syntax help and feedback on your commands, as you seen in the example below.



```
OWB OMB PLUS
OMB+> OMBHELP OMBCC

PURPOSE
OMBCC - Change Context command allows users to change the current context
to the desired location in OWB tree. The target context can be specified
either as an absolute path starting from the root ('/') or as a relative
path starting from the current context. Also, the path can contain '..',
which allows to navigate "up" to the parent context.

PREREQUISITES
Must be connected to a OWB repository.

SYNTAX
parseChangeContextCommand = OMBCC "QUOTED_STRING"

SEE ALSO
OMBDC
OMBDC

OMB+> OMBDC
/
OMB+> OMBCC 'NEW_FEATURES'
Context changed.
OMB+>
OMB+>
OMB+>
```

Figure 3 The OMB\*Plus Interface

Of course, it is also possible to create your scripts using a common text editor and run these within OMB\*Plus.

<sup>1</sup> For more information on TCL take a look at: <http://www.tcl.tk/>

## JDeveloper Integration

Another way to write OMB scripts and commands is within Oracle JDeveloper. You can register a Warehouse Builder extension component within JDeveloper that allows you to work with OMB scripting.

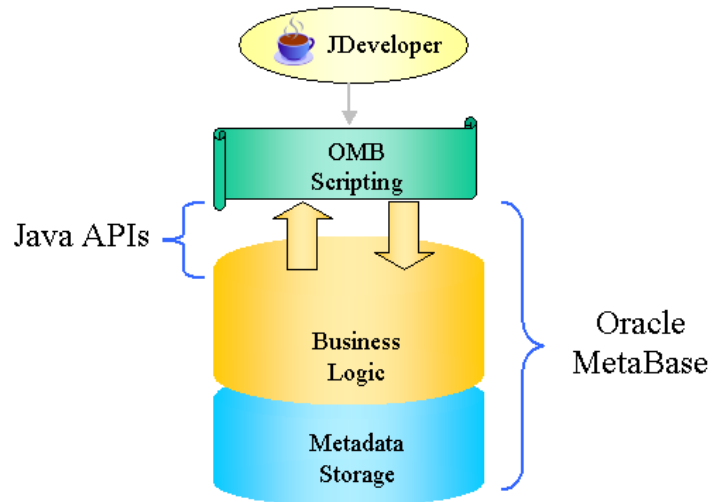


Figure 4 JDeveloper integration

Using the JDeveloper editor you get the following benefits within OMB scripting:

- Syntax Highlighting – implements color coding for OMB syntax making large scripts easier to read and understand
- Keyword Auto Completion – completes an OMB key word and increases productivity by reducing the need to type complete commands

## Business Benefits of OMB Scripting

Adding a scripting language to an already powerful tool like Warehouse Builder delivers specific business benefits to high end users. These benefits are:

- Increased Productivity – from two major aspects, task automation and mass update capabilities
- Portability – develop on one environment and run without changes on another (for example from Windows to UNIX)
- Application Embedding – save time by using the Warehouse Builder MetaBase instead of rebuilding a metadata store

With these benefits advanced users can make Warehouse Builder a highly productive and stable development platform.

## DETAILS AND EXAMPLES

After discussing the high level usages and implications of OMB scripting we will provide more details and examples in this section. As with any programming language, examples greatly enhance the understanding of a language and its usages.

In this chapter we will discuss a variety of commands. The goal is to allow you to follow basic examples to become familiar with the concepts of OMB scripting. To find all the detailed syntax and functionality, please refer to the OMB scripting documentation that comes with Warehouse Builder.

### Starting with OMB Scripting

As a first step you will have to connect to a repository from OMB\*Plus. After starting the OMB\*Plus tool issue the following command to connect to a repository:

```
OMBCONNECT owbrep/owbrep@localhost:1521:orcl92
```

Note that the OMB specific commands are case sensitive and that all commands require uppercase entry. Some OMB commands can be shortened to avoid too much typing. For example, OMBCONNECT can be shortened to OMBCONN. The general rule is that after the OMB component you will need the first four characters in the shortened version.

Using OMBDCC you can display the current position, or context, in the repository hierarchy. The repository hierarchy (the project tree in the graphical user interface) is comparable to a directory structure on an operating system. After connecting you are at the root, which is symbolized with ' / '.

A difference between the repository and a file system is the name of your position. In OMB scripting this is called your "context". Your context is determined by the location you are at within the Warehouse Builder tree. At initial login the root is your context. To change your context to a project issue the command:

```
OMBCC 'MY_PROJECT'
```

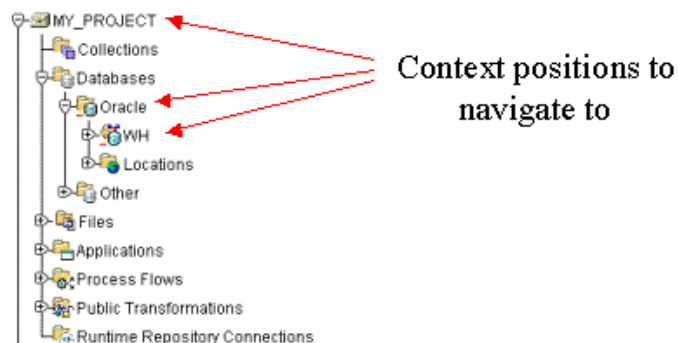


Figure 5 A Graphical Representation of the Context

To see the content of a context, use the `OMBLIST` command. The `OMBLIST` command will list all objects of the specified type. You can then (unless you are at module level already) change context to one of the listed objects. To disconnect OMB\*Plus from the repository, use the lower case `TCL exit` command.

Make sure to issue an `OMBCOMMIT` before leaving OMB\*Plus as the tool does not do an implicit commit.

### Creating and Updating Metadata

An important task in Warehouse Builder is the creation of schema objects. We will first look at how to create new objects in a repository then show how an object can be modified.

To work on the repository, we connected using the `OMBCONNECT` command as shown above. You can then create a new project in this repository, however we will work on the available `MY_PROJECT`.

Change context to the project:

```
OMBCC 'MY_PROJECT'
```

A difference between SQL\*Plus is that there is no need for a `' ; '` to signal execution, return will execute the command. To avoid execution of multiple line scripts at the first line, it is required to add a `' \ '` to signal the continuation of the script.

Within Warehouse Builder we are going to start at the following position in the tree:

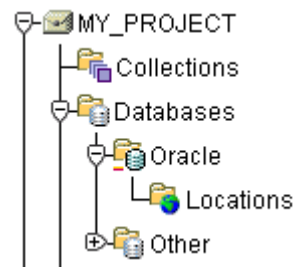


Figure 6 Tree for the empty `MY_PROJECT`

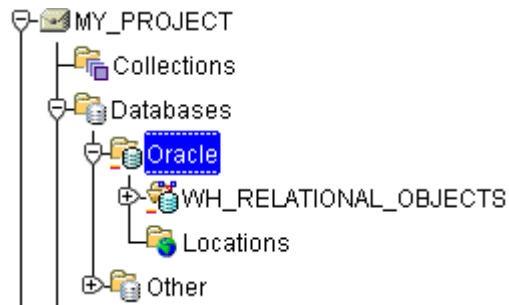
Objects are created within a module, in this case we have to create a new Oracle database target module:

```
OMBCREATE ORACLE_MODULE 'WH_RELATIONAL_OBJECTS' \
SET PROPERTIES (DESCRIPTION) VALUES ('Target Module for
testing relational objects')
```

Issue a commit to store your created objects:

OMBCOMMIT

This will lead to the following situation within the tree structure in Warehouse Builder showing the new module WH\_RELATIONAL\_OBJECTS, created:



**Figure 7 MY\_PROJECT with an Oracle Module**

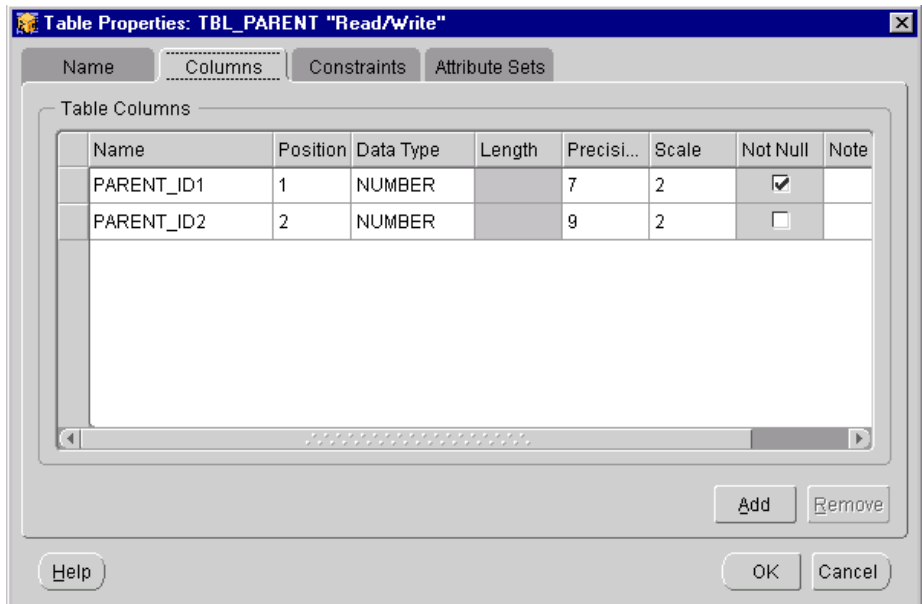
Within this new module we will create and manipulate objects. In this case we will create a table and then add a column to it. To create a table, run the following statement to change context to the module we just created:

```
OMBCC 'WH_RELATIONAL_OBJECTS'
```

Now run the table metadata create statement:

```
OMBCREATE TABLE 'TBL_PARENT' \
  SET PROPERTIES (DESCRIPTION, BUSINESS_NAME) \
  VALUES ('This table contains UK for referencing in \
    other objects.' \
    , 'Master Table') \
  ADD COLUMN 'PARENT_ID1' \
  SET PROPERTIES (DATATYPE, PRECISION, SCALE, \
    NOT_NULL) VALUES ('NUMBER', 7, 2, 1) \
  ADD COLUMN 'PARENT_ID2' \
  SET PROPERTIES (DATATYPE, PRECISION, SCALE) \
  VALUES ('NUMBER', 9, 2) \
  ADD PRIMARY_KEY 'TBLPARENT_PK1' \
  SET REFERENCE COLUMNS ('PARENT_ID1')
```

A table, TBL\_PARENT is created with the columns PARENT\_ID1 and PARENT\_ID2.



**Figure 8 UI version of TBL\_PARENT**

As a second step, we will add another table, TBL\_CHILD to the module with a foreign key relation to the TBL\_PARENT table:

```

OMBCREATE TABLE 'TBL_CHILD' \
  SET PROPERTIES (DESCRIPTION) \
  VALUES ('This table connects to TBL_PARENT.') \
  ADD COLUMN 'ID1' \
  SET PROPERTIES (DATATYPE, DESCRIPTION) \
  VALUES ('NUMBER', 'ID1 is the first column \
    of the PK') \
  ADD COLUMN 'COL2' \
  SET PROPERTIES (DATATYPE, NOT_NULL, DESCRIPTION) \
  VALUES ('NUMBER', 1, 'COL2 is the second \
    column of the PK') \
  ADD COLUMN 'VAR2' \
  SET PROPERTIES (DATATYPE, LENGTH) \
  VALUES ('VARCHAR2', 100) \
  ADD COLUMN 'NBR' \
  SET PROPERTIES (DATATYPE, PRECISION, SCALE, \
    NOT_NULL) \
  VALUES ('NUMBER', 10, 5, 0) \
  ADD PRIMARY_KEY 'TBL_PK1' \
  SET REFERENCE COLUMNS ('ID1', 'COL2') \
  ADD UNIQUE_KEY 'TBL_UK1' \
  SET REFERENCE COLUMNS ('ID1') \
  ADD FOREIGN_KEY 'TBL_FK' \

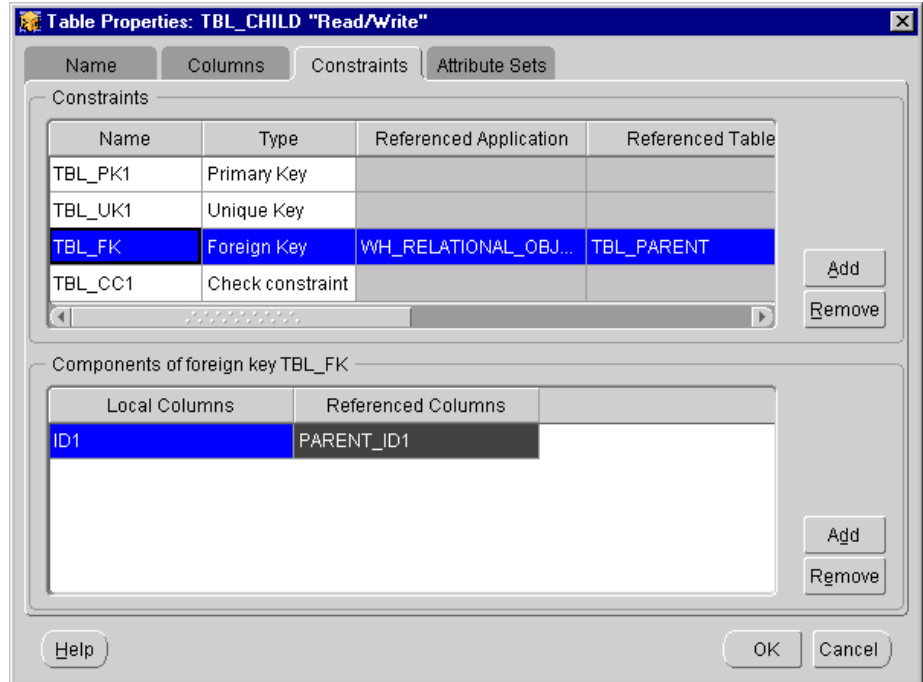
```

```

SET REFERENCE COLUMNS ('ID1') \
SET REFERENCE PRIMARY_KEY ('TBLPARENT_PK1') \
  OF TABLE '../WH_RELATIONAL_OBJECTS/TBL_PARENT' \
  ADD CHECK_CONSTRAINT 'TBL_CC1' \
SET PROPERTIES (CHECK_CONDITION) \
  VALUES ('NBR >= 0')

```

This TBL\_CHILD is linked to the TBL\_PARENT as we can see in the properties of the table.



**Figure 9 Foreign Key between TBL\_PARENT and TBL\_CHILD**

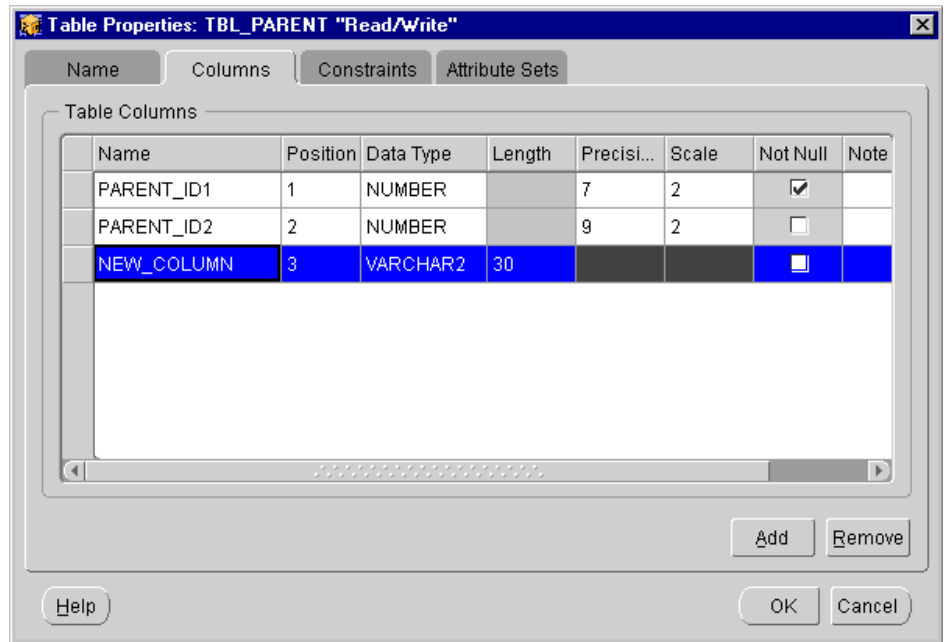
After creating these tables, we realized that TBL\_PARENT needs some additional columns. You can use the following script to add the NEW\_COLUMN column. This addition is then shown in the properties sheet for the table.

```

OMBALTER TABLE 'TBL_PARENT' \
ADD COLUMN 'NEW_COLUMN' \
SET PROPERTIES (DATATYPE, LENGTH) \
  VALUES ('VARCHAR2', 30)

OMBCOMMIT

```



**Figure 10 A new column added**

Using scripts like the ones shown above makes it possible to create and manipulate all objects within the Warehouse Builder repository.

### Automating Metadata Changes

While creating objects via a scripting interface is a powerful feature, automation is much more powerful. In the following example we will update all tables we just created with one script. This is exceptionally powerful if you need to update your design with a set of columns like creation date. Instead of using the graphical user interface and touching all tables you can now run a simple script. This reduces the amount of work drastically.

Let's add 4 columns to all tables within the module we were working in.

```
foreach tableName [OMBLIST TABLES] {

OMBALTER TABLE '$tableName' \
ADD COLUMN 'CREATED_BY' \
SET PROPERTIES (DATATYPE, LENGTH) \
VALUES ('VARCHAR2', 30) \
ADD COLUMN 'DATE_CREATED' \
SET PROPERTIES (DATATYPE) VALUES ('DATE') \
ADD COLUMN 'MODIFIED_BY' \
SET PROPERTIES (DATATYPE, LENGTH) \
VALUES ('VARCHAR2', 30) \
ADD COLUMN 'DATE_MODIFIED' \
```

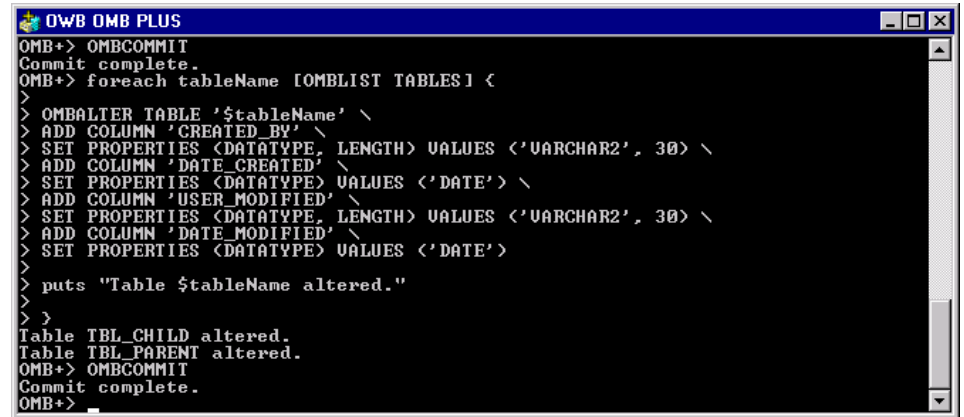
```

SET PROPERTIES (DATATYPE) VALUES ('DATE')

puts "Table $tableName altered."
}

```

This script first gets the list of tables in the current context and then loops through this script adding four columns to each of the tables in the list.



```

OWB OMB PLUS
OMB+> OMBCOMMIT
Commit complete.
OMB+> foreach tableName [OMBLIST TABLES] {
>
> OMBALTER TABLE '$tableName' \
> ADD COLUMN 'CREATED_BY' \
> SET PROPERTIES (DATATYPE, LENGTH) VALUES ('VARCHAR2', 30) \
> ADD COLUMN 'DATE_CREATED' \
> SET PROPERTIES (DATATYPE) VALUES ('DATE') \
> ADD COLUMN 'USER_MODIFIED' \
> SET PROPERTIES (DATATYPE, LENGTH) VALUES ('VARCHAR2', 30) \
> ADD COLUMN 'DATE_MODIFIED' \
> SET PROPERTIES (DATATYPE) VALUES ('DATE')
>
> puts "Table $tableName altered."
>
>
>
Table TBL_CHILD altered.
Table TBL_PARENT altered.
OMB+> OMBCOMMIT
Commit complete.
OMB+>

```

Figure 11 Command line feedback in OMB\*Plus

OMB\*Plus displays a text in the window when the table has been altered to confirm the change.

If you want to run a script that resides on the operating system from within OMB\*Plus, you must use the following command:

```
source c:\owb\my_script.tcl
```

This command will go to the qualified path and execute the specified file. So for the example in the next section, you can create a scripts with all the individual commands and run that from OMB\*Plus. You can also start OMB\*Plus and directly launch a script. After executing the script OMB\*Plus will close. You do this by just adding the script name and path to the execution command for the OMB\*Plus batch file:

```
ombplus.bat c:\owb\my_script.tcl
```

Make sure the script does all required tasks such as connecting to the repository and changing to the required context. On UNIX ensure to launch the shell script (ombplus.sh) in this way.

### Performing Generic Tasks

Apart from metadata modifications it is possible to automate certain generic tasks within Warehouse Builder. These tasks can then be scheduled. The following example will show a script to export the project metadata using a script. Note that in this example the directory MDL on the C-drive must exist.

```

OMBEXPORT MDL_FILE 'c:\mdl\exp_demo.mdl' \
PROJECT 'MY_PROJECT' \
COMPONENTS (ORACLE_MODULE \
            'WH_RELATIONAL_OBJECTS', \
            TABLE 'TBL_CHILD', TABLE 'TBL_PARENT') \
OUTPUT LOG 'c:\mdl\exp_demo.log'

```

The example shown here exports specific objects, however it is also possible to export a complete module or even project using a script. As you can see you must escape the \ character in the windows path with a second \. If you do not do that it will be interpreted as a line continuation character. Therefore use \\ to symbolize a directory.

This script can now be scheduled (using the process flow editor in Warehouse Builder) and run, for example, every night.

After exporting our metadata for back-up, we want to deploy the information in the project. For this we have to create a deployment action plan which we then deploy to the database. The following example shows you how to create the TBL\_PARENT. Since we start with an empty project without any previous setup, we have to go through the normal steps done on the client application.

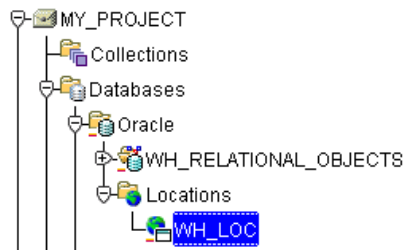
In project context, execute the following scripts to achieve a first time registration. Once these steps are done, you can run the later steps (the actual deployment) without repeating these first scripts.

Create a location:

```

OMBCREATE LOCATION 'WH_LOC' \
SET PROPERTIES (TYPE, VERSION, \
DESCRIPTION, BUSINESS_NAME) \
VALUES ('Oracle Database', '9.2', \
        'This is a location', 'Location')

```



**Figure 12** UI representation of a location WH\_LOC

Assign the location to the module created earlier:

```

OMBALTER ORACLE_MODULE 'WH_RELATIONAL_OBJECTS' \

```

```
SET REFERENCE \
LOCATION 'WH_LOC'
```

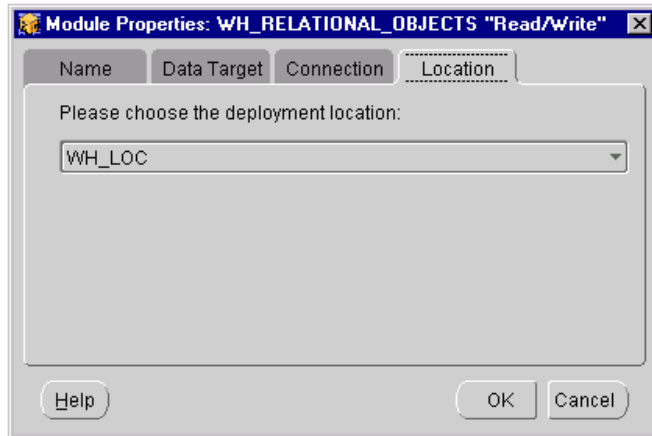


Figure 13 Assigned WH\_LOC location in the UI

Create a new runtime repository connection, which connects the design repository to the runtime repository:

```
OMBCREATE RUNTIME_REPOSITORY_CONNECTION \
'NEW_RR_CONNECTION' \
SET PROPERTIES (DESCRIPTION, BUSINESS_NAME, \
HOST, PORT, SERVICE_NAME, \
CONNECT_AS_USER, RUNTIME_REPOSITORY_OWNER) \
VALUES ('Runtime Repository', \
'RR_MY_PROJECT', 'localhost', \
1521, 'orcl92', 'OWBRU', 'OWBRR')
```

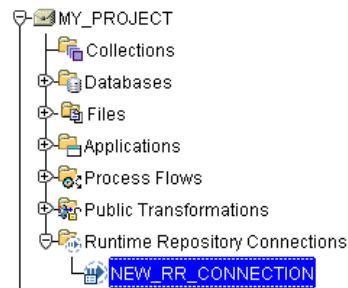


Figure 14 UI representation of Runtime Repository Connection

Connect from OMB\*Plus to the runtime environment (in the user interface this will launch the deployment manager):

```
OMBCONNECT RUNTIME 'NEW_RR_CONNECTION' USE PASSWORD
```

'owbru'

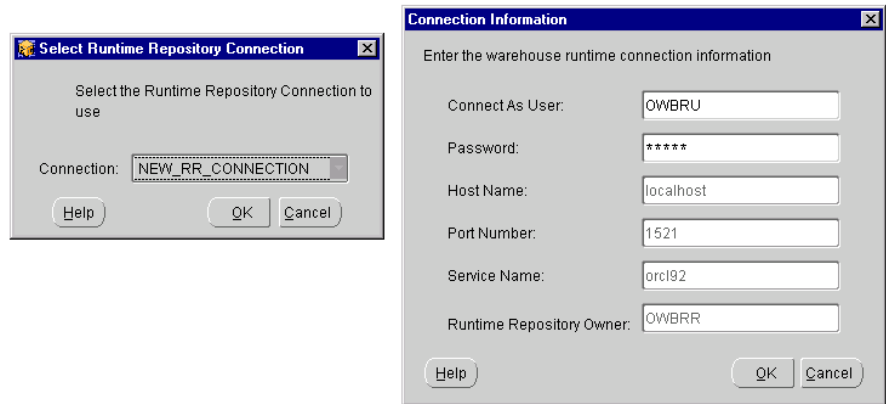


Figure 15 Connecting to the Runtime Platform via the UI

Register the location within the runtime platform:

```
OMBREGISTER LOCATION 'WH_LOC' \  
  SET PROPERTIES (Host, Port, Service, \  
                 Schema, Password) \  
  VALUES ('localhost', 1521, 'orcl92', \  
          'owb_tgt', 'owb_tgt')
```

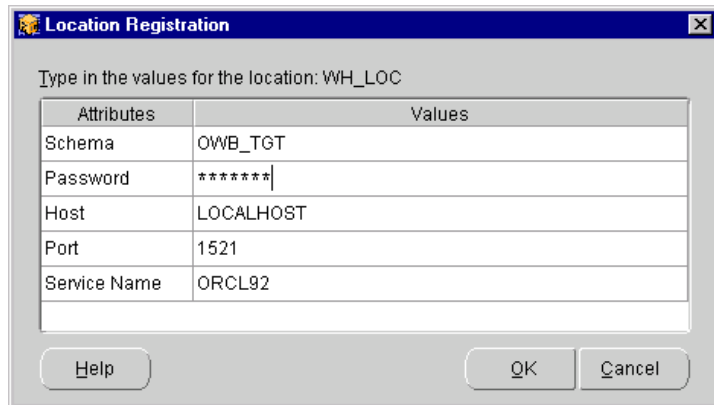


Figure 16 Registering a location within the UI

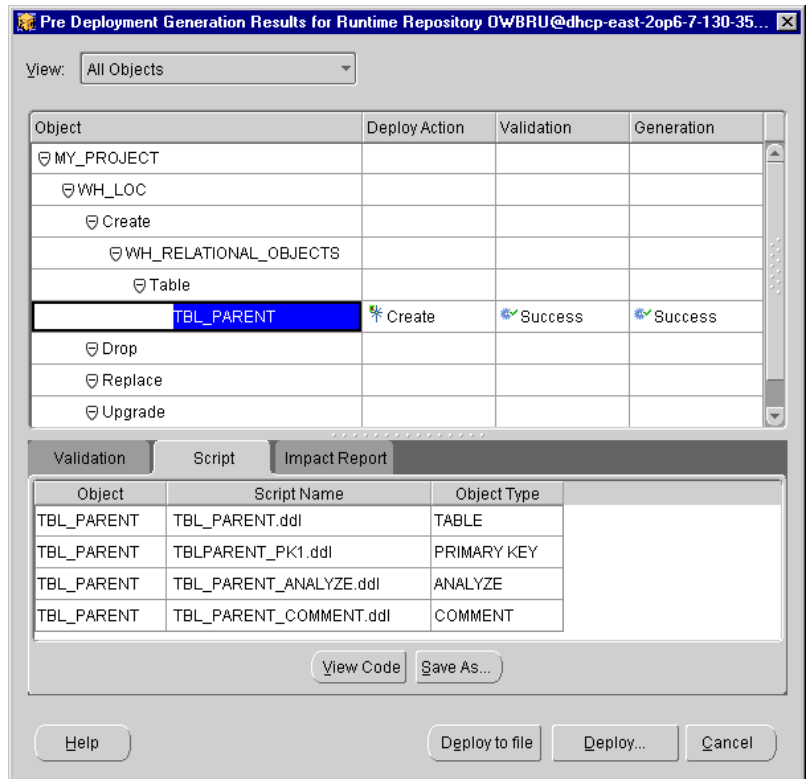
You are now all set to start the actual deployment by first creating a deployment action plan, which selects the objects you want to deploy, and the action to be performed.

Create deployment action plan:

```
OMBCREATE TRANSIENT DEPLOYMENT_ACTION_PLAN \  

```

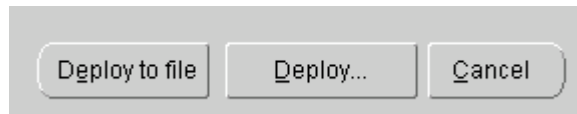
```
'DEPLOY_PLAN' ADD ACTION 'TABLE_DEPLOY' \
SET PROPERTIES (OPERATION) \
VALUES ('CREATE') SET REFERENCE TABLE \
/MY_PROJECT/WH_RELATIONAL_OBJECTS/TBL_PARENT'
```



**Figure 17 Viewing the deployment action plan in the UI**

Execute the deployment action plan to deploy the chosen objects into the database, in ORACLE\_MODULE context (OMBCC 'WH\_RELATIONAL\_OBJECTS') :

```
OMBDEPLOY DEPLOYMENT_ACTION_PLAN 'DEPLOY_PLAN'
```



**Figure 18 Executing the plan with the deploy buttons**

These last two steps can then be repeated at will or on schedule. Note that the deployment action plan is only valid during the current session to the repository.

This completes the examples. To find out more about the various scripting commands please look at the Warehouse Builder Scripting Reference that comes with the documentation for the product.

## **CONCLUSION**

A scripting language is a very powerful tool. The biggest benefits scripting brings to Warehouse Builder are:

- Automation of repetitive tasks
- More efficient mass change capabilities
- Off-line execution of activities

These benefits greatly enhance the productivity of Warehouse Builder, allowing developers to make maximum use of their resources. Being able to execute scripts instead of working on the UI (no matter how good that UI is) will always lead to faster implementation times. Therefore the addition of scripting to Warehouse Builder is a tremendous step forward.



Embed Oracle Warehouse Builder in your applications using scripting  
February 2004  
Author: Jean-Pierre Dijcks

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](http://www.oracle.com)

Oracle is a registered trademark of Oracle Corporation. Various product and service names referenced herein may be trademarks of Oracle Corporation. All other product and service names mentioned may be trademarks of their respective owners.

Copyright © 2004 Oracle Corporation  
All rights reserved.