OCI Mapper Debug 2004-1

By Josh Fletcher, Technical Support Engineer, 4D Inc.

Technical Note 06-12

Abstract

This Technical Note accompanies a version of the OCI Mapper component that features debug logging. The main focus of this Technical Note is to summarize the debugging features that have been added to the OCI Mapper. Installation procedures for the OCI Mapper Debug component are included.

This Technical Note is the second part of a two-part OCI Mapper update. The reason for the two-part release is the debugging code tends to make the OCI Mapper methods harder to read.

If you are not familiar with the OCI Mapper component, please see Technical Note 06-06, “OCI Mapper 2004-3”.

Introduction

The OCI Mapper is a framework that emulates the high-level commands found in 4D Oracle (a plug-in available for versions of 4D prior to 2004). It is provided in the form of a 4D component (including source code). Its methods were implemented with the native 4D language and 4D for OCI commands.

In this release a debug version of the OCI Mapper is provided. This version features the logging of OCI Mapper methods as well as 4D for OCI calls to a text file in order to facilitate advanced analysis of 4D for OCI programming problems. Additionally the debug logging functionality may be accessed from the developer’s own methods.

Developers who are already familiar with the OCI Mapper should find this Technical Note (and the new component) useful for debugging problems with the OCI Mapper as well as problems with their own 4D for OCI code.

4D for OCI developers may find the debug logging features useful as well for debugging their own code.

Note: it is not necessary to use any of the OCI Mapper methods in order the access the debug logging features provided in the new component. Thus the component can be useful for debugging 4D for OCI code outside the scope of the OCI Mapper methods.

In order to facilitate advanced debugging of the OCI Mapper framework, the OCI Mapper Debug component adds debugging code to all of the OCI Mapper methods. Additionally a total of 26 new methods have been added to the OCI Mapper to facilitate the debugging code.

The debugging code logs plain text messages to a text file.

OCI Mapper Debug 2004-1 in-depth

In this section you will find important design highlights of the OCI Mapper Debug component. This section is divided into two parts: general notes about the component design; and notes about specific methods.

Important General Notes

What Gets Logged?

For OCI Mapper methods, the start and end of each method are logged. The format of the log entry is like:

PID<*process number*> (*stack level*) *indentation* Method Start: *method name*

For example:

```
PID<1> (2) Method Start: OCI_Exist_Cursor
PID<1> (2) Method End: OCI_Exist_Cursor
```

The methods added to specifically support the debugging features are, of course, not logged.

For 4D for OCI calls inside the OCI Mapper methods a log entry is made before and after each 4D for OCI call. The format of the log entry is like:

PID<*process number*> (*stack level*) *indentation* Before: *function name*(*p1*;*p2*;...;*pn*)
PID<*process number*> (*stack level*) *indentation* After: *return value*:=*function name*(*p1*;*p2*;...;*pn*)

Where p1...pn are the parameters to the OCI function.

For example:

```
PID<1> (2) Before: OCIEnvCreate( 0; "OCI_HTYPE_ENV" )
PID<1> (2) After: 0:=OCIEnvCreate( 129735744; "OCI_HTYPE_ENV" )
```

Note that 4D for OCI constants are logged as strings instead of the integer constant value to aid readability.

The logging of OCI Mapper methods and 4D for OCI calls can be toggled on and off independently.

What Parameter Types Are Supported?

Currently the logging only supports integer and string parameters. When debugging 4D for OCI code these are often the two most important parameter types (handles are longints, SQL text is supported, etc.).

Other types of parameters will appear as "<n/a>" in the log file. However, the parameter logging functionality is easy to extend. See the notes for the method **OCIM_DBG_ParamToString**.

Stack Level

When logging debug messages the concept of a "stack frame" is used. This is referred to as the "stack level" in the component code. The stack level is indicated in the log file by both an integer value and indentation using spaces. Each time the start of a method is logged, the stack level for that process is increased. Thus subsequent log entries are indented further. When the end of the method is logged, the stack level is decreased.

Note that this has nothing to do with the actual stack used by 4D. This is done only to make the log file easier to read. Here is an example:

```
PID<1> (0)Method Start: OD Execute SQL
PID<1> (1)      Method Start: OCI_Exist_Login
PID<1> (1)      Method End: OCI_Exist_Login
PID<1> (1)      Method Start: OD Create cursor
PID<1> (2)              Method Start: OCI_Exist_Login
PID<1> (2)              Method End: OCI_Exist_Login
PID<1> (2)              Before: OCIHandleAlloc( 129735744; 0; ...
PID<1> (2)              After: 0:=OCIHandleAlloc( 129735744; 129796272; ...
PID<1> (1)      Method End: OD Create cursor
PID<1> (0)Method End: OD Execute SQL
```

The stack level is only adjusted when logging 4D methods. There is no need to adjust it for OCI calls since there is nothing can be executed between the OCI call. For example, this would not make sense since you can not "step into" the OCI Call:

```
Before oci call
        log some stuff
After oci call
```

Debug Log Monitor Process

A method is provided in the component to be used as a process to monitor the debug log file in order to prevent hard disk saturation. If this process is

not used, the debug log file will grow continuously. The process is started by default.

Debug Log Files

The debug log files created by the component are plain text files.

To prevent hard disk saturation, the debug log files created by the component are capped at 500KB by default. Log files that exceed this size are "archived" as separate files (if the log monitor process is running). The maximum log file size may be changed by the developer.

Only the last 6 archives are kept, by default. The maximum number of archives to keep may be changed by the developer.

Semaphores are used to allow multiple processes to access the debug log file.

Important Method Notes

This section covers important information about specific methods in the OCI Mapper Debug 2004-1 component.

Method: **OCIM_DBG_INIT**

This method takes care of the initialization of the debug logging code and starts the log monitor. There is no need to call this method directly for applications that already use the OCI Mapper; it is automatically called by **OCI_TOOL_INITVAROCI**. However, if you do not plan to use the OCI Mapper methods you should place this method in your startup code.

Method: **OCIM_DBG_LOGFILE_NEW**

If you wish to modify the name of the debug log file, modify this method. An interface to modify the log file name is not provided in the OCI Mapper Debug component (other than directly modifying the interprocess variable, which is not recommended).

Method: **OCIM_DBG_ParamToString**

If you wish to add support for the logging of other parameter types besides integers and strings, modify this method. The input is a pointer to the parameter and the output should be a string representation of that parameter.

OCI Mapper Debug 2004-1 Command Reference

All of the 26 new methods added to the OCI Mapper component in this version are prefixed with "OCIM_DBG_" to indicate that they are methods related to debugging the OCI Mapper.

Note that the only methods documented here are the commands useful to the 4D developer. For debugging methods not covered here, refer to the comments in the code. For documentation on the non-debugging OCI Mapper methods refer to Technical Note 06-06 or the comments in the code.

Finally, all of the commands in the component contain header comments that describe the purpose of the command as well as list input/output parameters as appropriate.

OCIM_DBG_GetLogFileName
Returns the file name of the current log file.

OCIM_DBG_GetLogFilePath
Returns the path to the current log file.

OCIM_DBG_GetLogLimit
Returns the maximum number of archived log files kept.

OCIM_DBG_GetLogSizeLimit
Returns the maximum size a log file can reach before it is archived.

OCIM_DBG_LOGFILE_DELETECURRENT
Deletes the current log file, if it exists. This method may be executed at any time; a semaphore is used to protect log file access.

OCIM_DBG_LOGFILE_NEW
Creates a new log file name (note that the actual log file is not created until something is logged to it). Subsequent logging will use this new file. The format of the debug log file name is:

```
OCI Mapper Log MM-DD-YYYY hh_mm_ss.txt
```

Where "MM" is the month, "DD" is the day, "YYYY" is the year of the current date and "hh" is the hour, "mm" is the minute, and "ss" is the seconds for the current time.

This command can be called at any time; semaphores are used to protect access to the log file.

OCIM_DBG_LogMapperOff
Turn off the logging of OCI Mapper method calls.

OCIM_DBG_LogMapperOn
Turn on the logging of OCI Mapper method calls.

OCIM_DBG_LogOCIOff
Turn off the logging of 4D for OCI function calls in the OCI Mapper methods.

OCIM_DBG_LogOCIOn
Turn on the logging of 4D for OCI function calls in the OCI Mapper methods.

OCIM_DBG_Log_MessageNoPad
Logs a message to the log file with no stack level. The message format is:

> PID<*Process Number*> *message*/r/n

OCIM_DBG_Log_MethEnd
Logs a message to the log file to indicate that the end of a 4D method has been reached. Decreases the stack level for the current process.

OCIM_DBG_Log_MethStart
Logs a message to the log file to indicate that the start of a 4D method has been reached. Increases the stack level for the current process.

OCIM_DBG_Log_OCIMethAfter
Logs a message to the log file to indicate that a 4D for OCI function call has completed. The format of the message is:

> After: *OCIReturnVal*:=*OCIMethodName*(*p1*;*p2*;...;*pn*)

Where p1...pn were the parameters to the OCI function.

OCIM_DBG_Log_OCIMethBefore
Logs a message to the log file to indicate that a 4D for OCI function is about to be called. The format of the message is:

> Before: *OCIMethodName*(*p1*;*p2*;...;*pn*)

Where p1...pn are the parameters to the OCI function.

OCIM_DBG_MonitorLogStart
Creates a process that will monitor the debug log file and archive it if needed.

OCIM_DBG_MonitorLogStop
Sets an interprocess variable so that the debug log monitoring process can be stopped.

OCIM_DBG_SetLogFilePath
Sets the path where the log file will be saved. Default path is the OS temporary folder. This command can be called at any time; semaphores are used to protect access to the log file.

OCIM_DBG_SetLogLimit
Sets the maximum number of archived log files to keep. Default is 6.

OCIM_DBG_SetLogSizeLimit
Sets the maximum size that the log file can reach before it is archived. Default is 500KB.

The following OCI Mapper Debug debugging methods are not covered here, but more information can be found in the source code for the methods:

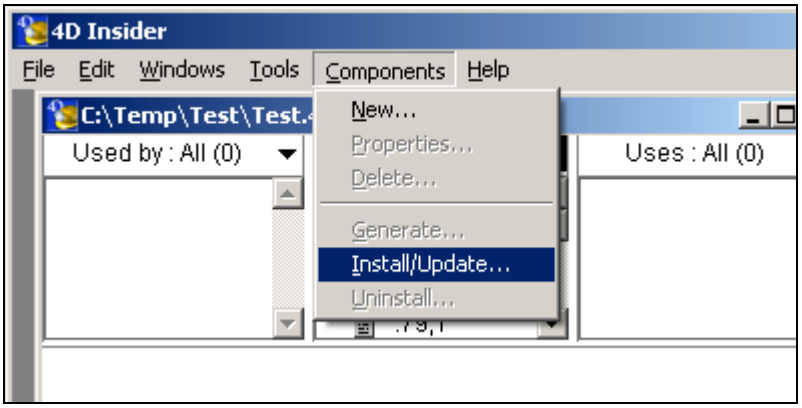
OCIM_DBG_FormatLogFilePath
OCIM_DBG_INIT
OCIM_DBG_LOGFILE_ARCHIVE
OCIM_DBG_Log_Message
OCIM_DBG_MonitorLog
OCIM_DBG_ParamToString

Installation Procedures

Note that 4D 2004 and 4D for OCI 2004 are required for the OCI Mapper Debug 2004-1.

Installing the OCI Mapper Debug component into a new database

1. Open the database with 4D Insider.
2. Open the **Components** menu and select **Install/Update...**:



3. Browse for the "OCI Mapper Debug 2004-1.4CP" file and click the **Open** button.

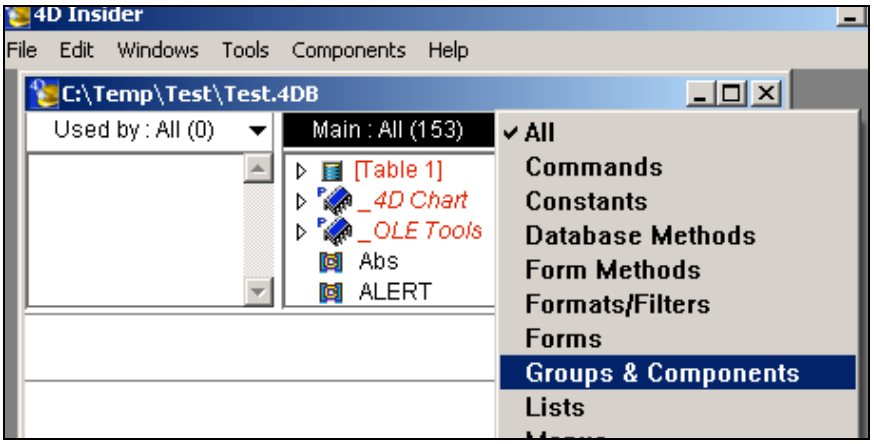
4. 4D Insider installs the OCI Mapper Debug component. Be sure to copy the "OCI_Mapper" table:

5. Quit 4D Insider.
6. Open the database with 4D.
7. Insert the method **OCI_TOOL_INITVAROCI** or **OCIM_DBG_INIT** in the **On Startup** database method.
8. Quit 4D.
9. Finally be sure that the 4D for OCI plug-in is installed in the database.

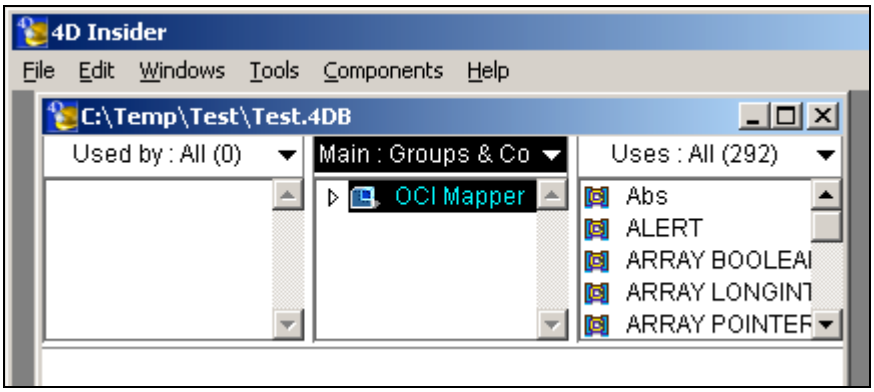
Installing the component into a database that already uses the OCI Mapper

Note: you cannot install the OCI Mapper Debug component without first uninstalling the "regular" OCI Mapper component from a database that already uses the OCI Mapper.

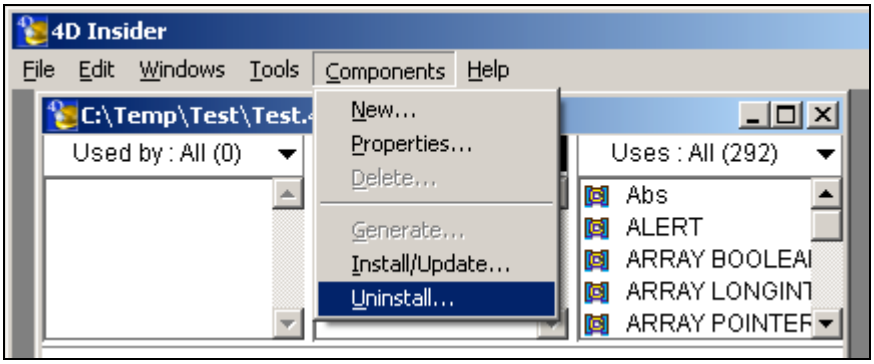
1. Open the database with 4D Insider.
2. First uninstall the "regular" version of the OCI Mapper. Open the **Main** pop-up menu and select **Groups & Components**:



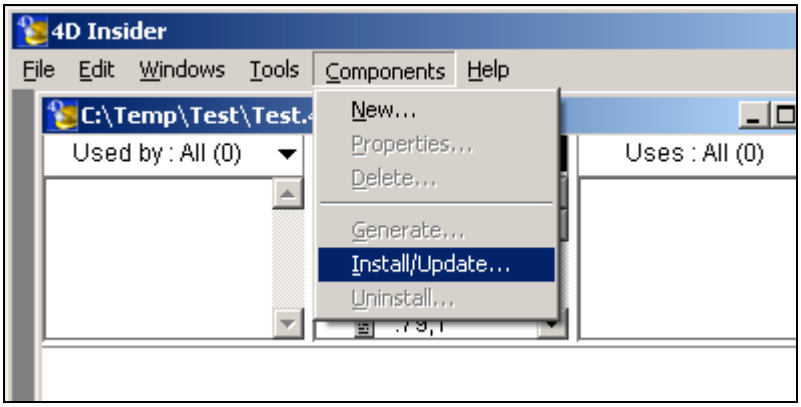
3. Select the **OCI Mapper** component:



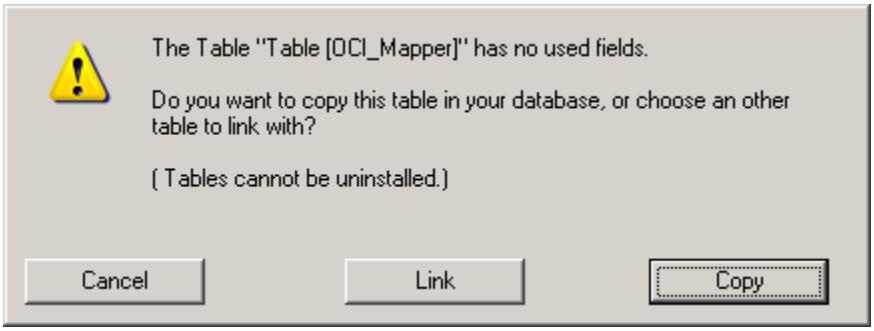
4. Open the **Components** menu and select **Uninstall...**:



5. Click the **OK** button to uninstall the "regular" OCI Mapper.
6. Open the **Components** menu and select **Install/Update...**:



7. Browse for the "OCI Mapper Debug 2004-1.4CP" file and click the **Open** button.
8. 4D Insider installs the OCI Mapper Debug component. Be sure to copy the "OCI_Mapper" table:



9. Quit 4D Insider.

Using the OCI Mapper Debug Component

If this is a new database in which the OCI Mapper has never been installed, be sure to add **OCI_TOOL_INITVAROCI** to your startup code. You **must** execute this method before you call any other OCI Mapper methods. If you only plan to use the debug logging features of the component you may call **OCIM_DBG_INIT** in your startup code instead of **OCI_TOOL_INITVAROCI**.

If this is a database that already used the OCI Mapper you should already have **OCI_TOOL_INITVAROCI** in your startup code. The debug log monitor process will already be started and the logging is turned on by default when you start your database.

To stop the debug log file monitor process use **OCIM_DBG_MonitorLogStop**. **Beware that the log file will grow continuously if you do this**. To start it again use **OCIM_DBG_MonitorLogStart**.

If you wish to disable the logging of OCI Mapper method calls, execute **OCIM_DBG_LogMapperOff**. Use **OCIM_DBG_LogMapperOn** to turn the logging back on.

If you wish to disable the logging of 4D for OCI calls in the OCI Mapper, execute **OCIM_DBG_LogOCIOff**. Use **OCIM_DBG_LogOCIOn** to turn the logging back on.

There are several options that can be set in regards to the debug log file:

- The location of the log file.
- The maximum number of archived log files to keep.
- The maximum size the log file can reach before archiving.

The location of the log file can be set with **OCIM_DBG_SetLogFilePath**. The default setting is the OS temporary folder.

The maximum number of archives can be set with **OCIM_DBG_SetLogLimit**. Note that this is only enforced if the debug log monitor process is running. The default is 6.

The maximum size of the log file can be set with **OCIM_DBG_SetLogSizeLimit**. Note that this is only enforced if the debug log monitor process is running. The default is 500KB.

Logging Your Own 4D Methods

If you would like to log the start and end of your own 4D methods do the following:

At the beginning of your method add a line like:

```
OCIM_DBG_Log_MethStart (Current method name)
```

At the end of your method add a line like:

```
OCIM_DBG_Log_MethEnd (Current method name)
```

For example, given the 4D project method "MyMethod" the log entry might look like:

```
PID<1> (0)Method Start: MyMethod
PID<1> (0)Method End: MyMethod
```

Logging Your Own 4D for OCI Calls

If you would like to log the start and end of your own 4D for OCI calls do the following:

Given a 4D project method that is about to call the 4D for OCI function **OCIStmtExecute** as:

```
$status:=OCIStmtExecute ($Svchp;$Stmthp;$Errhp;1;0;0;0;OCI_DEFAULT )
```

Before the 4D for OCI function call add a code block like:

```
C_TEXT($ociConstant)
C_LONGINT($zero;$one)
$ociConstant:="OCI_DEFAULT"
$zero:=0
$one:=1
OCIM_DBG_Log_OCIMethBefore ("OCIStmtExecute";->$Svchp;->$Stmthp;->$Errhp;->$one;-
>$zero;
->$zero;->$zero;->$ociConstant)
```

Note how the 4D for OCI constant OCI_DEFAULT is represented as a string. This is not necessary, you could just use the constant's value, but it makes the log file

easier to read. Also notice that literals must be represented with a variable since the parameters to **OCIM_DBG_Log_OCIMethBefore** must be pointers.

After the 4D for OCI function call add a code block like:

```
OCIM_DBG_Log_OCIMethAfter ($status;"OCIStmtExecute";->$Svchp;->$Stmthp;->$Errhp;->$one;->$zero;
->$zero;->$zero;->$ociConstant)
```

Notice that the first parameter to **OCIM_DBG_Log_OCIMethAfter** is the return value of the 4D for OCI function. All 4D for OCI commands have a return value with the exception of **OCIOnErrCall**.

The log entry for this code might look like:

```
PID<1> (2) Before: OCIStmtExecute( 129758392; 129795236; 129796272; 1; 0; 0; 0;
"OCI_DEFAULT" )
PID<1> (2) After: 0:=OCIStmtExecute( 129758392; 129795236; 129796272; 1; 0; 0; 0;
"OCI_DEFAULT" )
```

A Final Word

There may be features of the OCI Mapper Debug component that you do not like or that do not highlight a problem that you are trying to debug. If this is the case, remember that you have the source code! Please feel free to alter the design as it suits your database. A great attempt has been made to document the source code of the OCIM_DBG methods to make it easy should the developer choose to change the design. Hopefully you will find the debug log file management code useful in any database.

Useful Resources

Useful 4D Resources

4D for OCI Documentation:
http://www.4d.com/products/downloads_4d.html
(Included with the 4D 2004 All-in-One Installer package)

Useful Oracle Resources

Oracle Call Interface Programmer's Guide, 10*g* Release 2 (10.2):
http://download-west.oracle.com/docs/cd/B19306_01/appdev.102/b14250/toc.htm

Oracle Call Interface Programmer's Guide, Release 8.1.6
http://download-west.oracle.com/docs/cd/A87862_01/NT817CLI/index.htm

Conclusion

The OCI Mapper provides a good example of how to build a framework for 4D for OCI. However it also provides a level of abstraction from “pure” 4D for OCI programming. This abstraction can make debugging the 4D for OCI code a more difficult task. Thus the OCI Mapper Debug component can be a great help in debugging the OCI Mapper code. Furthermore, the OCI Mapper Debug component provides a good example of how to build a debugging “harness” for 4D for OCI programming.