4D for OCI: Error Handling

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

Technical Note 05-45

Abstract

The purpose of this technical note is to describe the two error handling techniques available in 4D for OCI. Additionally this document will cover the basic concepts of error handling used in Oracle Call Interface (OCI) programming. A sample database is provided that demonstrates the error handling techniques.

Overview of 4D for OCI Error Handling

There are two techniques available for handling errors in 4D for OCI:

- **Standard error handling**
- **4D error handling**

The first technique, referred to as **Standard error handling**, involves manually checking the return code of each 4D for OCI function call and processing errors as needed. This technique tends to require that the developer write more code.

The second technique, referred to as **4D error handling**, is similar to handling errors in 4D with the ON ERR CALL command. The 4D for OCI plug-in includes a command called OCIOnErrCall:

OCIOnErrCall (methodName)

Parameter	Type
methodName	String

This command installs the specified project method as the method for catching (trapping) OCI errors. This project method is called the **error-handling method** or **error-catching method**. Using **4D error handling** eliminates the need for extra code to check for OCI errors. If an error occurs, 4D will call the **error-handling method** automatically. This technique tends to require less code.

Both of these error handling techniques will be discussed in more detail later in this document. First, it is important to understand how error handling works in OCI programming.

OCI Error Handling

OCI error handling involves two diagnostic tools:

- The return code of the OCI function call.
- An error **handle**, which can be used to access error **records** created by OCI.

The return code of every 4D for OCI function call is used to indicate whether or not the function executed successfully. The possible return codes are listed here (taken from the "Oracle Call Interface Programmer's Guide Release 8.1.6", December 1999):

OCI Return Code	Value	Description
OCI_SUCCESS	0	The function completed successfully.
OCI_SUCCESS_WITH_ INFO	1	The function completed successfully; a call to OCIErrorGet() will return additional diagnostic information. This may include warnings.
OCI_NO_DATA	100	The function completed, and there is no further data.
OCI_ERROR	-1	The function failed; a call to OCIErrorGet() will return additional information.
OCI_INVALID_HANDLE	-2	An invalid handle was passed as a parameter or a user callback is passed an invalid handle or invalid context. No further diagnostics are available.
OCI_NEED_DATA	99	The application must provide run-time data.
OCI_STILL_EXECUTING	-3123	The service context was established in non-blocking mode, and the current operation could not be completed immediately. The operation must be called again to complete. OCIErrorGet() returns ORA-03123 as the error code.
OCI_CONTINUE	-24200	This code is returned only from a callback function. It indicates that the callback function wants the OCI library to resume its normal processing.

However, in many cases, this return code will not provide sufficient information to determine what caused the error. Thus OCI programming also requires the use of error handles.

Nearly every 4D for OCI function requires an error handle as a parameter. This error handle is used to reference error records.

Error records are created by OCI in the event of certain database errors (e.g. OCI_ERROR). These records are used to store extra information beyond to return code of the 4D for OCI function. For example, an error record might have the error code of the error (e.g. 10003) and the error text of the error (e.g. "Invalid table name").

Note that a single error handle can point to multiple error records.

Thus, in order to correctly process errors with 4D for OCI, the error handle must be used to check the errors.

Standard Error Handling

Standard error handling involves two basic steps:

- Check the return code of each 4D for OCI function call.
- Check for errors as appropriate.

An example follows:

```
$l_returnValue:=OCIStmtPrepare    ($l_stmthp;$l_stmt_errhp;$t_SQL;Length($t_SQL))
If ($l_returnValue=OCI_SUCCESS )
        ` Continue OCI calls...
Else
        My_OCI_Error_Handler ($l_returnValue;$l_stmt_errhp)
End if
```

In this example the return code of the 4D for OCI command OCIStmtPrepare is checked for success with the variable $l_returnValue. Additionally an error handle ($l_stmt_errhp) is provided for the function call. If the function call is not successful (success is indicated by the constant OCI_SUCCESS or '0') a custom error handling method is called (My_OCI_Error_Handler).

This technique of testing for success and handling the error if needed should be repeated for every OCI function call in the database. Thus an If-Else block will be needed for every OCI function call.

The primary advantage to this technique is that the developer is in complete control of OCI error handling. The disadvantages are primarily: it requires more lines of code, potentially making the code harder to read; and it makes it more difficult to swap out error handling methods if desired.

Pros	Cons
• The developer is in control of all error handling	• More code • More difficult to use multiple error handlers

4D Error Handling

4D error handling involves a single line of code to install the **error-handling method**. For example:

```
OCIOnErrCall    ("My_OCI_Error_Handler")
```

Any errors generated by OCI function calls will automatically trigger a call to the **error-handling method**. Thus any 4D for OCI commands may be issued as in this example:

```
$l_returnValue:=OCIStmtPrepare    ($l_stmthp;$l_stmt_errhp;$t_SQL;Length($t_SQL))
```

No other code is necessary. If an error occurs in OCIStmtPrepare 4D for OCI will call the **error-handling method** as needed.

It is advisable, however, to at least test for successful execution of any 4D for OCI function calls as in this example:

```
$l_returnValue:=OCIStmtPrepare    ($l_stmthp;$l_stmt_errhp;$t_SQL;Length($t_SQL))
If ($l_returnValue=OCI_SUCCESS )
        ` Continue OCI calls...
End if
```

Using this technique, in the event of an error, none of the subsequent OCI function calls will be executed. This allows the code to exit gracefully in the event of an error.

The primary advantages of this technique are: fewer lines of code are required; and it makes it trivial to change error handlers. The disadvantage is the developer must rely on 4D for OCI to call their error handler.

Pros	Cons
• Less code • Easy to use multiple error handling methods	• The developer is not on complete control of error handling

The Custom OCI Error-Handling Method

It has been established that OCI programming relies on two diagnostic tools for error handling; the return code of OCI function calls and an error handle. The question is what must be done with this information? The code examples in this tech note have referred to a custom error handler called My_OCI_Error_Handler. This method will be explored here.

Parameters

The definition of My_OCI_Error_Handler is as follows:

```
My_OCI_Error_Handler (returnCode, errorHandle)
```

Parameter	Type
returnCode	Longint _
errorHandle	Longint _

Note that this custom error handler can be used with both error handling techniques.

For Standard error handling the parameters must be explicitly passed, for example:

```
$l_returnValue:=OCIStmtPrepare    ($l_stmthp;$l_stmt_errhp;$t_SQL;Length($t_SQL))
If ($l_returnValue=OCI_SUCCESS )
        ` Continue OCI calls...
Else
        My_OCI_Error_Handler ($l_returnValue;$l_stmt_errhp)
End if
```

In this example the return code is passed as the variable $l_returnValue and the error handle is passed as $l_stmt_errhp.

With 4D error handling the parameters are automatically passed by 4D for OCI.

The Return Code and Error Handle

To handle the error, the return code is useful for determining if there are error records available or not. A Case statement is useful here:

```
Case of
  : ($l_returnCode=OCI_ERROR )
  : ($l_returnCode=OCI_SUCCESS_WITH_INFO )
  : ($l_returnCode=OCI_NO_DATA )
 : ($l_returnCode=OCI_INVALID_HANDLE )
 : ($l_returnCode=OCI_NEED_DATA )
 : ($l_returnCode=OCI_STILL_EXECUTING )
 : ($l_returnCode=OCI_CONTINUE )
End case
```

In the case that there are no error records, a generic error message can be used. For example:

```
: ($l_returnCode=OCI_INVALID_HANDLE )
 ALERT("Error:   OCI_INVALID_HANDLE")
```

In the case that there are error records, the information in those records can be used to generate error messages. In the case of OCI_ERROR or OCI_SUCCESS_WITH_INFO there should be error records to check.

To check the error records the 4D for OCI command OCIErrorGet is used:

```
OCIErrorGet (hndlp; recordno; errcodep; bufp) _ status
```

Parameter	Type
hndlp	Longint _
recordno	Longint _
errcodep	Longint _
bufp	String _
status	Longint _

The first parameter to OCIErrorGet is the error handle. The second parameter is the record number to be retrieved (index starts at 1). The third is the numeric Oracle error code for the error record (e.g. for an ORA-00936 error the error code will be 936). The forth parameter is the actual error message text (e.g. "Invalid table name"). Here is an example of using the error handle to generate an error message:

```
$l_status:=OCIErrorGet   ($l_errorHandle;1;$l_errorCode;$t_errorText)
ALERT("Code:  "+String($l_errorCode)+"\nMessage:  "+$t_errorText)
```

In this example the error handle is passed as the variable $l_errorHandle. The record number is specified by the literal '1'. The error code will be returned in the variable $l_errorCode and the error message will be returned in the variable $t_errorText.

Remember that there can be more than one error record per error handle so it is important to loop through them. This can be accomplished by using an index variable and incrementing it. Note that it is not possible to get the total number of error records via OCI. Instead the application must continue calling the OCIErrorGet function until its return value is OCI_NO_DATA (or not OCI_SUCCESS, which is safer in case the call to OCIErrorGet fails in some way). When the index passed to OCIErrorGet is greater than the number of error records, OCIErrorGet will return OCI_NO_DATA.

Putting It All Together

Here is the implementation of My_OCI_Error_Handler:

```
 ` ------------------------------------------------------
 ` Method: My_OCI_Error_Handler
 ` Description
 ` Error handling method that can be used with OCIOnErrCall().
 ` For error processing all this mehod does is display the errors in an alert.
 `
 ` Parameters
 ` ------------------------------------------------------------------------------
 ` C_LONGINT($1) = Return code from the function that reported the error
 `         (e.g. OCI_ERROR, OCI_INVALID_HANDLE, etc.)
 ` C_LONGINT($2) = OCI error handle (the error handle contains error
 `         "records" that have the actual error data in them.
 ` ------------------------------------------------------------------------------
C_LONGINT($1;$l_returnCode)
C_LONGINT($2;$l_errorHandle)

$l_returnCode:=$1
$l_errorHandle:=$2

 ` This long integer is used to index through the records diagnostic records
 ` in the error handle. In OCI it is possible for a single error handle to
 ` have multiple error records.
C_LONGINT($l_errRecNum)
 ` The error records start at 1.
$l_errRecNum:=1
```

```
 ` Stores return code from any OCI calls in this method.
C_LONGINT($l_status)

 ` The error code of the error in the error record. E.g. for an
 ` ORA-00936 error the error code will be 936.
C_LONGINT($l_errorCode)

 ` The error message text.
C_TEXT($t_errorText)

 ` This is just used in the ALERT message...
C_TEXT($t_methodName)
$t_methodName:=Current method name

 ` A loop is used because there can be more than one error record...
Repeat
  Case of
   : ($l_returnCode=OCI_ERROR )
     ` To use OCIErrorGet the address of the error handle is needed
     ` and the error record number to load (starts at 1). The third
     ` and fourth parameters are output variables in which OCI will
     ` place values.
    $l_status:=OCIErrorGet ($l_errorHandle;$l_errRecNum;$l_errorCode;$t_errorText)

     ` If this is false, there are no more error records or another
     ` error occured.
    If ($l_status=OCI_SUCCESS )
      If ($t_errorText="")
       ALERT($t_methodName+"\nError: OCI_ERROR\nCode: "+String($l_errorCode)+"\nMessage:
No further information.")
      Else
       ALERT($t_methodName+"\nError: OCI_ERROR\nCode: "+String($l_errorCode)+"\nMessage:
"+$t_errorText)
      End if
    End if

  : ($l_returnCode=OCI_SUCCESS_WITH_INFO )
    ` OCI_SUCCESS_WITH_INFO is used to indicate that, while your
    ` OCI operation executed successfully, there is extra
    ` information you may want.
    $l_status:=OCIErrorGet ($l_errorHandle;$l_errRecNum;$l_errorCode;$t_errorText)
    If ($l_status=OCI_SUCCESS )
      ALERT($t_methodName+"\nCode: "+String($l_errorCode)+"\nInfo: "+$t_errorText)
    End if

   : ($l_returnCode=OCI_NO_DATA )
    ALERT($t_methodName+"\nError: OCI_NO_DATA")
    $l_status:=OCI_NO_DATA

   : ($l_returnCode=OCI_INVALID_HANDLE )
    ALERT($t_methodName+"\nError: OCI_INVALID_HANDLE")
    $l_status:=OCI_NO_DATA
```

```
    : ($l_returnCode=OCI_NEED_DATA )
      ALERT($t_methodName+"\nError:    OCI_NEED_DATA")
      $l_status:=OCI_NO_DATA

    : ($l_returnCode=OCI_STILL_EXECUTING )
      ALERT($t_methodName+"\nError:    OCI_STILL_EXECUTING")
      $l_status:=OCI_NO_DATA

    : ($l_returnCode=OCI_CONTINUE )
      ALERT($t_methodName+"\nError:    OCI_CONTINUE")
      $l_status:=OCI_NO_DATA
    End case

    ` Increment to the next error record. You must do this in order
    ` to exit the loop. There is no way to tell if there are more
    ` error records until you call OCIErrorGet and receive a result
    ` of OCI_NO_DATA.
   $l_errRecNum:=$l_errRecNum+1
  Until ($l_status#OCI_SUCCESS )
```

Using the Sample Database

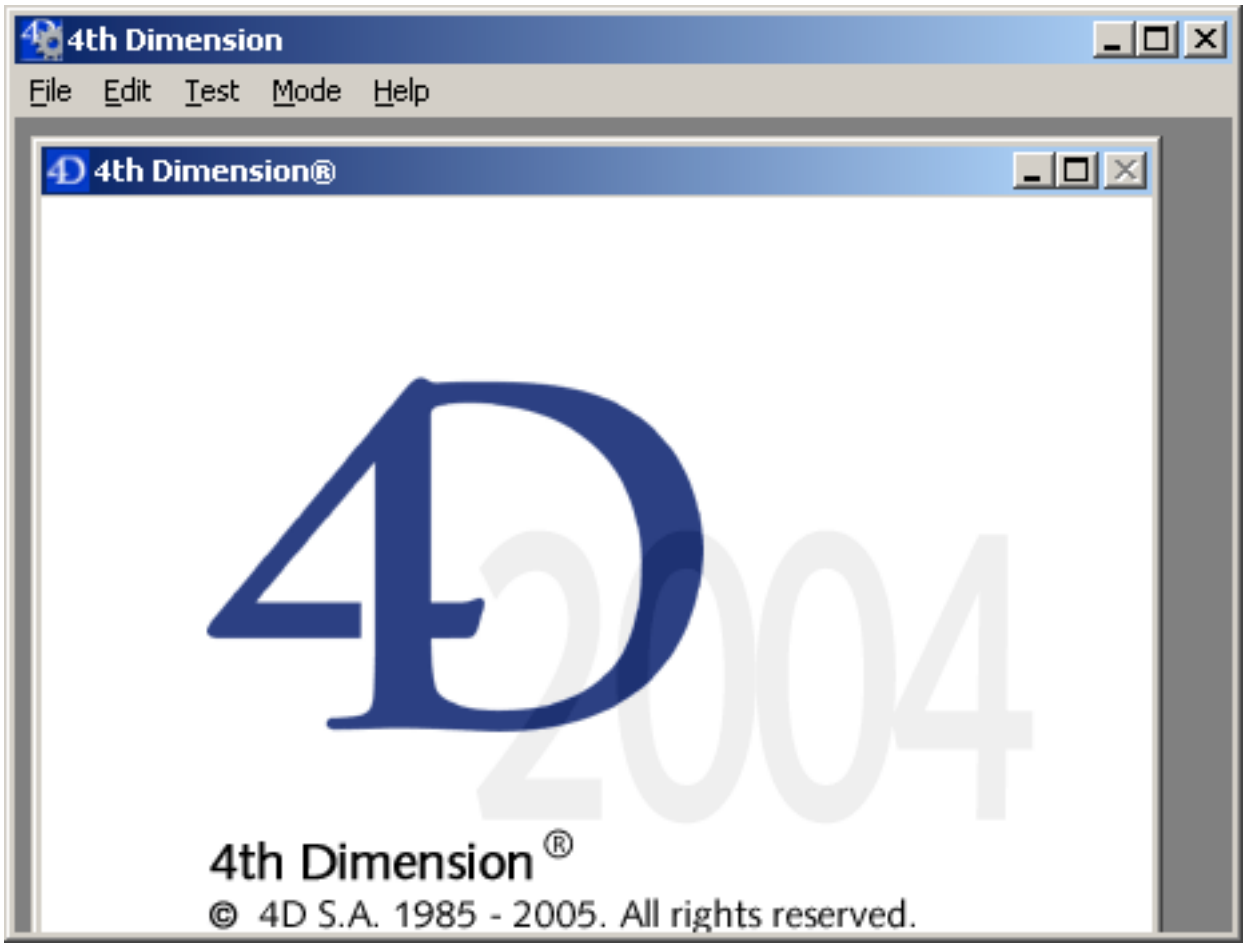
The sample database provides the following features:

- A dialog form that can be used to connect to an Oracle database and execute two SQL statements (one bad, one good). Error handling can be toggled from Standard error handling to 4D error handling.
- A custom OCI error handler project method.
- A sample project method that demonstrates Standard error hadling.
- A sample project method that demonstrates 4D error handling.

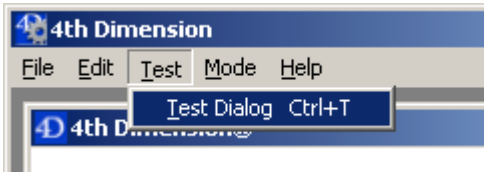
To use the sample database:

- Open “4D for OCI Error Handling.4DB”
- Create a new data file when prompted

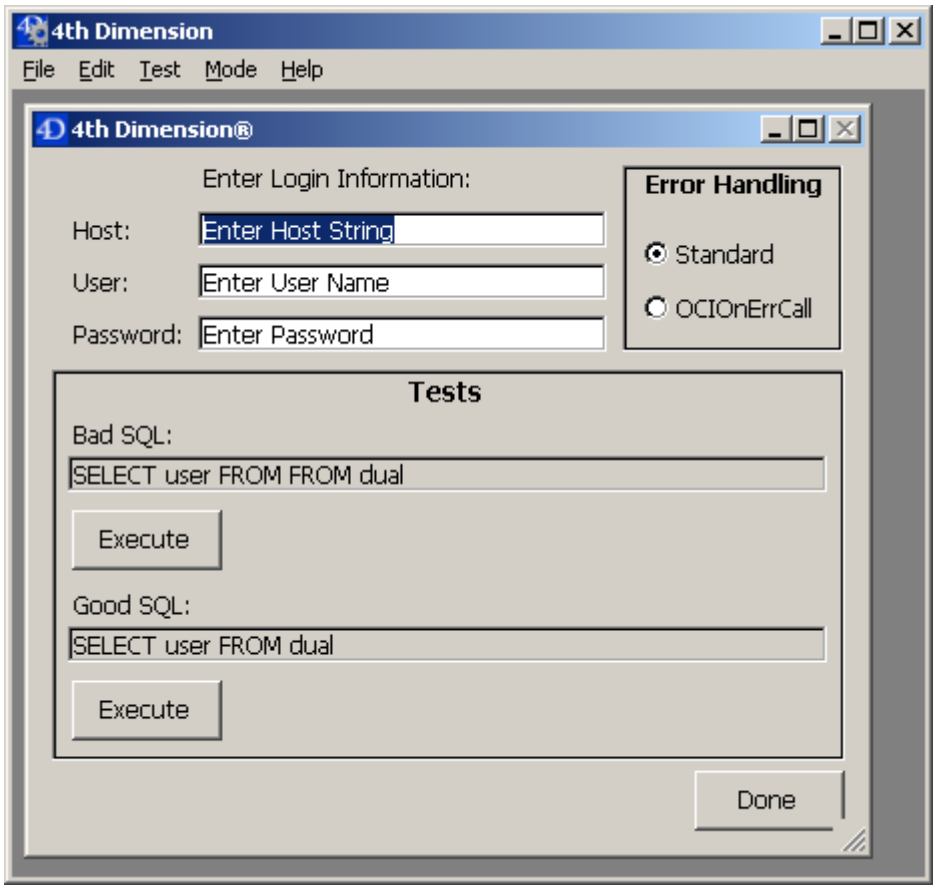
When the data file creation is complete the 4D splash screen is displayed:



Click on the menu labeled "Test" and select "Test Dialog" (or press CTRL+T).



This will open the “Test_Dialog” form:



This form can be used to execute two test cases, one with bad SQL, one with good SQL. Note that the good SQL should execute on any Oracle installation so no setup is needed on the Oracle side.

To run the tests:

- Enter the Oracle host string. This can be a TNS entry or a full host string (e.g. “1.2.3.10:1521/orcl”).
- Enter username (default Oracle username is “scott”)
- Enter password (default Oracle password is “tiger”)

Alter the error handling technique if desired. The options are:

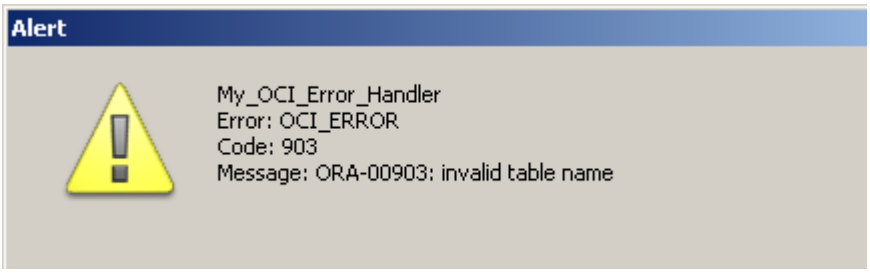
- Standard – uses Standard error handling
- OCIOnErrCall – uses the 4D error handling

Note that there will be no noticeable difference between the two test cases when executed with Standard error handling versus 4D error handling. The custom error handler was designed to be used from both contexts. The differences are in the way the code is written.

In the "Tests" section of the form two tests can be performed:

- Bad SQL – executes a malformed select statement
- Good SQL – executes a valid SQL statement

For the "Bad SQL" test an alert message will be displayed with an error like:

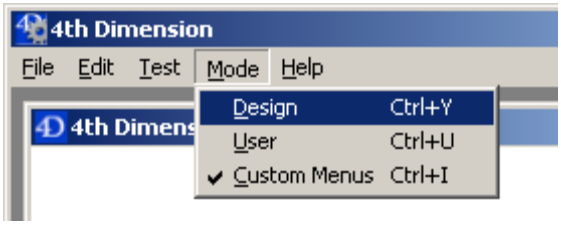


The above error occured because the SQL statement ("SELECT user FROM FROM dual") has an extra "FROM" clause in it. There was no table "FROM" in the database. Obviously, if a table named "FROM" exists in the target Oracle database a different error will be generated.

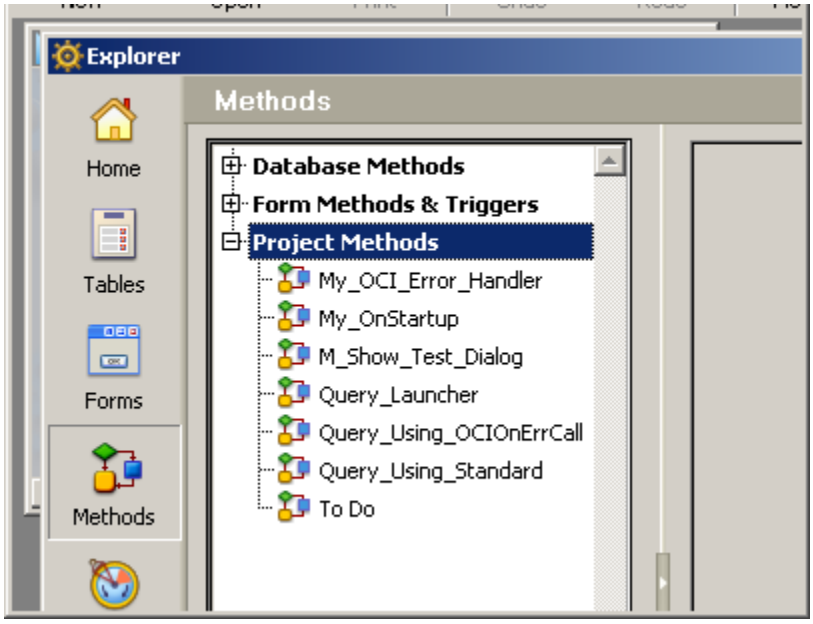
For the "Good SQL" test an alert message will be displayed with the value from the successful execution of the statement like this:

To exit out of the "Test_Dialog" form click the "Done" button or press Esc.

Next switch to Design Mode by clicking on the "Mode" menu and selecting Design (or press CTRL+Y).



Open the Explorer and view the project methods:



The following methods should be of interest:

- My_OCI_Error_Handler - the generic custom error handler used by both tests available from the "Test_Dialog" form.
- Query_Using_Standard – demonstrates the Standard error handling technique.
- Query_Using_OCIOnErrCall – demonstrates the 4D error handling technique.

The other methods simply support the functionality of the "Test_Dialog" form and are not relevant to OCI error handling.

Notice in Query_Using_Standard an error check is performed for every single OCI function call.

Notice in Query_Using_OCIOnErrCall there are no calls to "My_OCI_Error_Handler" since 4D for OCI will call the installed **error-handling method** automatically if an error occurs.

Note that, in the context of 4D error handling, "error" means anything other than OCI_SUCCESS. In other words, while OCI_SUCCESS_WITH_INFO might not be considered an error, 4D for OCI will still call the **error-handling method** so that you can decide what to do in that case.

Conclusion

This technical note described the general concepts of OCI error handling required to understand error handling in 4D for OCI. Two 4D for OCI error handling techniques were presented and compared. This information should allow the 4D for OCI developer to write their own custom error handlers and choose whichever error handling technique they prefer.