OCI Mapper Debug 2004-1

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

TN 06-12

Abstract

このテクニカルノートには、改訂版の OCI Mapper コンポーネントが付属しており、追加されたデバッグログ機能の使用方法について主に説明しています。コンポーネントのインストール手順についても簡単に扱っています。

このテクニカルノートは、前項編で提供される OCI Mapper コンポーネントプログラムの後編です。記事を簡潔にするため、デバッグ用のメソッドと本編のメソッドは分けて発表することにしました。

OCI Mapper コンポーネントに馴染みがなければ、テクニカルノート 06-06「OCI Mapper 2004-3」を参照してください。

Introduction

OCI Mapper コンポーネントは、4D for Oracle(バージョン 2004 以前の 4D で提供されていたプラグイン)に存在するハイレベルコマンドのエミューレーションフレームワークです。メソッドは、ネイティブ 4D コマンドと 4D for OCI プラグインコマンドを使用して作成され、ソースコードと共にコンポーネントの形式で配布されています。

今回リリースするのは、デバッグバージョンの OCI Mapper コンポーネントです。デバッグバージョンでは、OCI Mapper メソッドコールや 4D for OCI プラグインコールを外部テキストファイルに書き出す機能が追加されており、4D for OCI プログラミングで直面するかもしれない問題を分析し、解決するために活用することができます。他のメソッドからデバッグメソッドをコールすることもできます。

OCI Mapper をすでに使用しているデベロッパの場合、今回のテクニカルノートおよび改訂コポーネントは、OCI Mapper および 4D for OCI コード全般のデバッグに役立つはずです。

4D for OCI を使用しているデベロッパも、自身の 4D for OCI プログラミングのデバッグツールとしてこのコンポーネントを活用することができます。

Note: デバッグ機能は OCI Mapper メソッドを使用しなくても単独で使用することができます。

したがってコンポーネントは OCI Mapper メソッドとは別に、一般的な 4D for OCI コードをデバッグする目的でも使用することができます。

フレームワーク全体でデバッグが利用できるように、OCI Mapper Debug コンポーネントではすべての OCI Mapper メソッドにデバッグコードが追加され、デバッグに関連して合計 26 個のメソッドが追加されました。

デバッグコードを実行すると、ログは標準テキストファイルに書き出されます。

OCI Mapper Debug 2004-1 in-depth

この項目では、OCI Mapper Debug コンポーネントの設計に関する重要なポイントを説明します。まずはじめにコンポーネントの全体的な設計、次いで特定のメソッドについて説明します。

Important General Notes

What Gets Logged?

各 OCI Mapper メソッドについては、メソッド開始時および終了時にログが記録されます。ログエントリーの形式は次のとおりです：

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

例えば、次のようなログが記録されます：

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

いうまでもなく、ログを記録するためのメソッド自体はログの対象外にはなりません。

OCI Mapper メソッドの中で実行される 4D for OCI コールについては、コール前およびコール後にログが記録されます。ログエントリーの形式は次のとおりです：

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

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

p1、p2 は OCI 関数に渡されたパラメータを表わします。

例えば、次のようなログが記録されます：

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

ログを読みやすくするため、数値タイプのパラメータはテキストで書き出されます。

OCI Mapper メソッドのログと 4D for OCI コールのログは、個別にオン/オフを切り替えることができます。

What Parameter Types Are Supported?

現在のバージョンでは、整数および文字列のパラメータだけがサポートされています。デバッグという観点からいえば､もっとも重要なのが両パラメータタイプです｡(倍長整数のハンドル、SQL テキストなど。)

その他のパラメータタイプは「<n/a>」としてログに記録が残ります。ただし、デバッグログ機能は比較的容易に拡張することができます。詳しくは OCIM_DBG_ParamToString メソッドのコメントを参照してください。

Stack Level

デバッグメッセージのログを記録するにあたっては､「スタックフレーム」の概念が採用されています。ただし、コンポーネントコードの中では「スタックレベル」という用語が使用されています。スタックレベルは、ログファイルの中では整数値とスペースによるインデントで表示され、新しいメソッドのログが開始するたびにスタックレベルが上がります。連続したメソッドコールがあれば、それに応じてスタックレベルがさらに上がり、メソッドの実行が終了するとスタックレベルが下がります。

このスタックを 4D が使用するスタックと混同しないでください。この場合のスタックは、単純にログファイルを読みやすくするための仕組みに過ぎません。スタックレベルが適用される結果、ログファイルは次のような書式になります：

```
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
```

スタックレベルに変化が生じるのは、4D メソッドが実行されるときだけです。OCI コールの最中に別のコールを実行することはあり得ないので、スタックは変化しません。

Debug Log Monitor Process

コンポーネントには、ハードディスク容量を圧迫しかねないログファイルのサイズを別プロセスで監視するためのメソッドも含まれています。このプロセスを使用しない場合、デバッグログファイルはひたすら大きくなることになります。デフォルトの設定では、監視プロセスを使用するようになっています。

Debug Log Files

コンポーネントが生成するデバッグログファイルは、標準デキストファイルです。

ファイルサイズは、デフォルトで 500KB が上限となっており、ファイルサイズが上限に達したログファイルは、別ファイルとしてアーカイブされます。(デバッグログが実行中の場合。)ファイルサイズの上限は、変更することができます。

デフォルトの設定では、最大で 6 ファイルまでがアーカイブされ、それよりも古いものは残されません。保存するファイルの数は、変更することができます。

複数のプロセスがログファイルにアクセスできるように、セマフォが使用されています。

Important Method Notes

この項目では、OCI Mapper Debug 2004-1 コンポーネントのデバッグ関連メソッドについて説明します。

メソッド : OCIM_DBG_INIT

デバッグログを初期化し、ログモニターを開始するためのメソッドです。OCI Mapper メソッドを使用しているアプリケーションの場合、OCI_TOOL_INITVAROCI が自動的にコールするので、このメソッドをコールする必要はありません。OCI Mapper メソッドを使用していないアプリケーションの場合、最初のこのメソッドを実行する必要があります。

メソッド : OCIM_DBG_LOGFILE_NEW

デバッグログファイルの名称を変更する場合、このメソッドを編集してください。OCI Mapper Debug コンポーネントには、ユーザインタフェースでログファイル名を指定するメカニズムは実装されていません。

(インタープロセス変数に直接アクセスすることは勧められていません。)

メソッド : OCIM_DBG_ParamToString

整数および文字列以外のパラメータタイプをサポートするなど、デバッグログを拡張する場合、このメソッドを編集してください。入力値はパラメータに対するポインタ、出力値はそのパラメータを文字列で表わしたものである必要があります。

OCI Mapper Debug 2004-1 in-depth

OCI Mapper コンポーネントに追加された 26 個のメソッドは、いずれもデバッグ関連メソッドであることを示すために「OCIM_DBG_」というプリフィックスが付いています。

この項目では、4D デベロッパにとって有用なコマンドだけが挙げられています。ここで挙げられていないコマンドについては、それぞれのメソッドに記述されたコメント文を参照してください。デバッグ関連メソッド以外のメソッドについては、テクニカルノート 06-06 または、それぞれのメソッドに記述されたコメント文を参照してください。

すべてのコンポーネントのコマンドは、入出力パラメータおよびコマンドの目的がヘッダコメント文で説明されています。

OCIM_DBG_GetLogFileName

カレントログファイルのファイル名を返します。

OCIM_DBG_GetLogFilePath

カレントログファイルのパス名を返します。

OCIM_DBG_GetLogLimit

アーカイブするログファイルの最大数を返します。

OCIM_DBG_GetLogSizeLimit

ログファイルの最大ファイルサイズを返します。

OCIM_DBG_LOGFILE_DELETECURRENT

カレントログファイルが存在すれば削除します。ファイルアクセスはセマフォで保護されているので、このコマンドはいつでも実行することができます。

OCIM_DBG_LOGFILE_NEW

新しいログファイルを作成します。(実際にログすべき動作があるまでは、ファイルは作成されません。)コマンド実行後、ログは作成されたログファイルに書き出されます。デバッグログファイル名の形式は次のようなものです：

OCI Mapper Log MM-DD-YYYY hh_mm_ss.txt

MM は現在の月、DD は現在の日、YYYY は現在の年、hh は現在の時間、mm は現在の分、ss は現在の秒を表わします。

ファイルアクセスはセマフォで保護されているので、このコマンドはいつでも実行することができます。

OCIM_DBG_LogMapperOff

OCI Mapper メソッドコールのログ書き出しを停止します。

OCIM_DBG_LogMapperOn

OCI メソッドコールのログ書き出しを再開します。

OCIM_DBG_LogOCIOff

4D for OCI 関数コールのログ書き出しを停止します。

OCIM_DBG_LogOCIOn

4D for OCI 関数コールのログ書き出しを再開します。

OCIM_DBG_Log_MessageNoPad

スタックレベルを増減しないメッセージをログに書き出します。メッセージ形式は次のようになります：

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

OCIM_DBG_Log_MethEnd

4D メソッドの終了を意味するメッセージをログに書き出します。カレントプロセスのスタックレベルがひとつ下がります。

OCIM_DBG_Log_MethStart

4D メソッドの開始を意味するメッセージをログに書き出します。カレントプロセスのスタックレベルがひとつ上がります。

OCIM_DBG_Log_OCIMethAfter

4D for OCI 関数コールの完了を意味するメッセージをログに書き出します。メッセージ形式は次のようになります：

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

p1、p2 は OCI 関数に渡されたパラメータを表わします。

OCIM_DBG_Log_OCIMethBefore

4D for OCI 関数コール直前を意味するメッセージをログに書き出します。メッセージ形式は次のようになります：

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

p1、p2 は OCI 関数に渡されたパラメータを表わします。

OCIM_DBG_MonitorLogStart

デバッグログファイルを監視し、必要に応じてアーカイブするプロセスを起動します。

OCIM_DBG_MonitorLogStop

デバッグログ監視プロセスが停止するようにインタープロセス変数を書き換えます。

OCIM_DBG_SetLogFilePath

ログファイルの保存先ファイルパスを設定します。デフォルトパスは OS の一時フォルダです。ファイルアクセスはセマフォで保護されているので、このコマンドはいつでも実行することができます。

OCIM_DBG_SetLogLimit

アーカイブするログファイルの最大数を設定します。デフォルトの数は 6 です。

OCIM_DBG_SetLogSizeLimit

ログファイルの最大ファイルサイズを設定します。デフォルトのサイズは 500KB です。

以下のデバッグ関連メソッドについては省略しましたが、それぞれのコメント文に説明が記されています：

OCIM_DBG_FormatLogFilePath

OCIM_DBG_INIT

OCIM_DBG_LOGFILE_ARCHIVE

OCIM_DBG_Log_Message

OCIM_DBG_MonitorLog

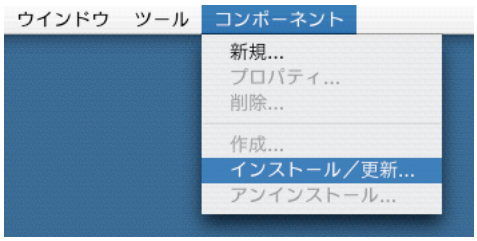
OCIM_DBG_ParamToString

Installation Procedures for OCI Mapper 2004-3

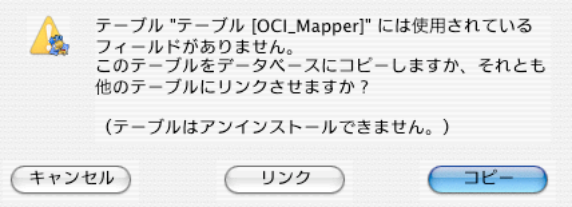
OCI Mapper Debug 2004-1 を使用するためには 4D 2004 および 4D for OCI 2004 が必要です。

データベースに新しく OCI Mapper Debug コンポーネントをインストールする

1. データベースを 4D Insider で開きます。
2. コンポーネントメニューからインストール／更新を選択します。



3. "OCI Mapper Debug 2004-1.4CP"ファイルの場所をブラウズし、開くボタンをクリックします。
4. 4D Insider が OCI Mapper コンポーネントをインストールします。このとき、"OCI_Mapper"テーブルもコピーするようにしてください。

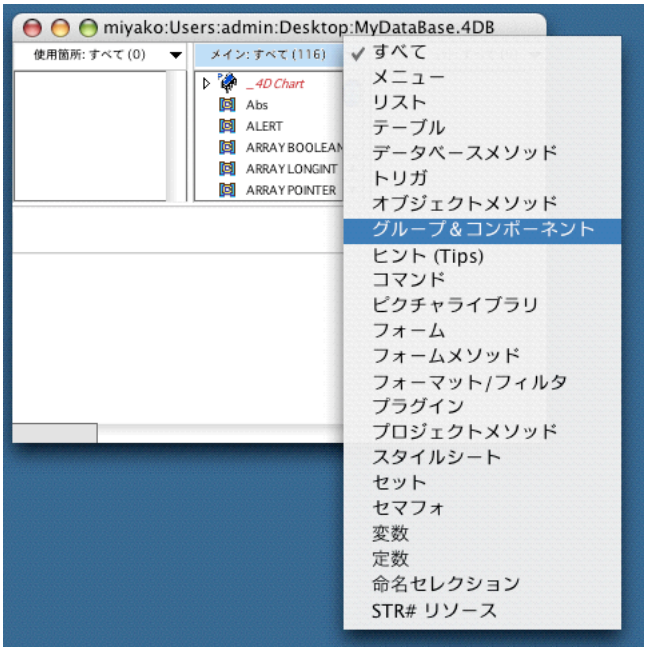


5. 4D Insider を終了します。
6. データベースを 4D で開きます。
7. On Startup データベースメソッドに OCI_TOOL_INITVAROCI または OCIM_DBG_INIT を挿入します。
8. 4D for OCI プラグインが正しくインストールされていることを確認します。

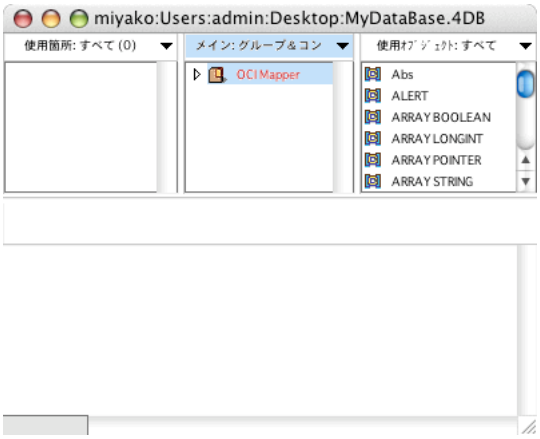
OCI Mapper がインストールされているデータベースにコンポーネントをインストールする

Note: OCI Mapper Debug コンポーネントをインストールする前に、普通の OCI Mapper をアンインストールする必要があります。

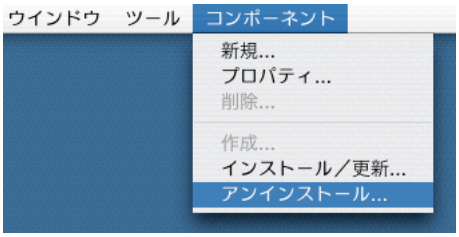
1. データベースを 4D Insider で開きます。
2. 先に普通の OCI Mapper をアンインストールする必要があるので、4D Insider のメインポップアップメニューをクリックし、グループ＆コンポーネントを選択します：



3. OCI Mapper コンポーネントを選択します。

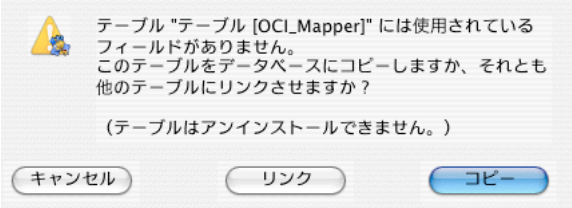


4. コンポーネントメニューからアンインストールを選択します：



5. OK ボタンをクリックして OCI Mapper をアンインストールします。
6. コンポーネントメニューからインストール／更新を選択します。

7. "OCI Mapper Debug 2004-1.4CP"ファイルの場所をブラウズし、開くボタンをクリックします。
8. 4D Insider が OCI Mapper コンポーネントをインストールします。このとき、"OCI_Mapper"テーブルもコピーするようにしてください：



9. 4D Insider を終了します。

Using the OCI Mapper Debug Component

はじめてデータベースに OCI Mapper コンポーネントをインストールしたのであれば、On Startup データベースメソッドに OCI_TOOL_INITVAROCI が追加されていることを確認してください。このメソッドをコールした後でなければ、他の OCI Mapper メソッドをコールすることができません。コンポーネントのログデバッグ機能だけを利用するのであれば、OCI_TOOL_INITVAROCI の代わりに OCIM_DBG_INIT をコールしても構いません。

普通の OCI Mapper コンポーネントが元々インストールされていたのであれば、On Startup データベースメソッドには OCI_TOOL_INITVAROCI がすでに存在するはずです。データベースを開始すると、自動的にデバッグログ監視プロセスが起動し、デフォルトの設定でログ出力が実行されます。

デバッグログの出力を停止するには、OCIM_DBG_MonitorLogStop を使用します。そうしないのであれば、ログファイルがひたすら大きくなり続けることに留意してください。ログを再開するには OCIM_DBG_MonitorLogStart を使用します。

OCI Mapper メソッドのログ出力を停止するには、OCIM_DBG_LogMapperOff を使用します。ログを再開するには OCIM_DBG_LogMapperOn を使用します。

OCI Mapper による 4D for OCI コールのログ出力を停止するには、OCIM_DBG_LogOCIOff を使用します。ログを再開するには OCIM_DBG_LogOCIOn を使用します。

デバッグログファイルの使用にあたっては、幾つか設定できるオプションがあります：

- ログファイルを保存する場所。
- 保管するログファイルアーカイブの最大数。
- ログファイルの最大ファイルサイズ。

ログファイルを保存する場所は、OCIM_DBG_SetLogFilePath で変更することができます。デフォルトの設定は OS の一時フォルダです。

保管するログファイルアーカイブの最大数はOCIM_DBG_SetLogLimitで変更することができます。アーカイブが実行されるためには、監視プロセスが作動していなければなりません。デフォルトの設定は6です。

ログファイルの最大サイズは、OCIM_DBG_SetLogSizeLimitで設定することができます。アーカイブが実行されるためには、監視プロセスが作動していなければなりません。デフォルトの設定は500KBです。

Logging Your Own 4D Methods

自作のメソッドをログに残すには、次のようにコードを編集します：

メソッドの冒頭に次のような一行を追加します：

OCIM_DBG_Log_MethStart (**Current method name**)

メソッドの末尾に次のような一行を追加します：

OCIM_DBG_Log_MethEnd (**Current method name**)

仮にプロジェクトメソッド「MyMethod」において上記のような処理を施した場合、次のようなログが生成されます：

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

Logging Your Own 4D for OCI Calls

自作のメソッドで4D for OCIコールをログに残すには、次のようにコードを編集します：

4D for OCI関数をコールする4Dメソッドが「OCIStmtExecute」である場合：

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

4D for OCI関数をコールする前に次のようなコードを追加します：

```
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)
```

4D for OCIの定数OCI_DEFAULTが文字列として渡されている点に注目してください。数値でわたしても構いませんが、ログファイルを解析する際のことを考えれば、文字列のほうが便利です。いずれにしても、OCIM_DBG_Log_OCIMethBeforeのパラメータはすべてポインタで渡すようにしてください。

4D for OCI 関数をコールした後に次のようなコードを追加します：

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

OCIM_DBG_Log_OCIMethAfter の第一パラメータがコマンドの返り値である点に注目してください。OCIOnErrCall 意外の OCI Mapper コマンドはすべて返り値を返します。

上記のような処理を施した場合、次のようなログが生成されます：

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

OCI Mapper Debug コンポーネントを使用してみて不便に感じる点、不足に感じる点があるかもしれません。ソースコードは公開されており、そのような場合、自由に変更して構いません。実際、いろいろなデータベースに合わせて各デベロッパが柔軟にコードを応用できるように、ソースコードに詳細なドキュメント解説をつけるために相当な労力が払われました。

Useful Resources

役に立つ 4D 情報

4D for Oracle ドキュメント：

ftp://ftp.4d-japan.com/ACI_PRODUCT_REFERENCE_LIBRARY/68/ORACLE67.pdf

4D for OCI ドキュメント：

ftp://ftp.4d-japan.com/ACI_PRODUCT_REFERENCE_LIBRARY/68/4DOCI8.pdf

役に立つ Oracle 情報

Oracle Call Interface プログラマーズガイド、10g リリース（10.2）：

http://downloadwest.oracle.com/docs/cd/B19306_01/appdev.102/b14250/toc.htm

Oracle Call Interface プログラマーズガイド、リリース 8.1.6：

http://download-west.oracle.com/docs/cd/A87862_01/NT817CLI/index.htm

Conclusion

OCI Mapper は生産性と効率を上げるために 4D for OCI 用のフレームワークを作成するという手法の良い例です。反面、純粋な 4D for OCI プログラミングからは逸脱するような路線をとっているのも事実であり、結果としてそれだけデバッグ作業が厄介になる可能性があります。そのような訳で OCI Mapper Debug コンポーネントはとても有用なツールです。実際、OCI Mapper Debug コンポーネントは、4D for OCI プログラミングを支援する安全ベルトを自分で用意するという優れた実例です。