Oracle Forms Developer and Oracle Reports Developer

共通ビルトイン・パッケージ リファレンス

リリース 6*i*

2000 **年** 4 **月**

部品番号 : J01126-01

ORACLE®

Oracle Forms Developer and Oracle Reports Developer

共通ビルトイン・パッケージ リファレンス リリース 6i

部品番号: J01126-01

原本名: Oracle Forms Developer and Oracle Reports Developer: Common Built-in Packages, Release 6i

原本部品番号: A73152-01

目次

はじめに

Oracle Forms Developer and Oracle Reports Developer 共通ビルトイン・パッケージ リファレンス リリース*6i*にようこそ。

このリファレンス・ガイドでは、Forms DeveloperおよびReports Developerを効果的に利用できるようにするための情報と、ビルトイン・パッケージに関する詳細な情報が説明されています。

ここでは、このガイドの構成を説明し、Forms DeveloperおよびReports Developerを使用する際に参考になるその他の情報源を紹介します。

前提条件

まず、ご使用のコンピュータおよびそのオペレーティング・システムについて精通している必要があります。たとえば、ファイルの削除およびコピーのコマンドの知識があり、検索パス、サブディレクトリおよびパス名の概念を理解していなければなりません。詳細は、各オペレーティング・システムの製品マニュアルを参照してください。

アプリケーション・ウィンドウの要素などのMicrosoft Windowsの基本要素も理解している必要があります。エクスプローラ、タスクバー、タスクマネージャ、またはレジストリなどのプログラムに精通している必要があります。

表記規則

このマニュアルでは、次のような表記上の規則を使用しています。

規則	意味
固定幅フォント	固定幅フォントのテキストは、表示されたとおりに入力するコマンドを示します。PCに入力するテキストでは、特に断りのない限り大文字と小文字を区別しません。 コマンドでは、大カッコと縦線以外の句読点は表示されているとおり正確に入力する必要があります。
小文字	コマンド文の小文字は変数を表します。適切な値に置き換えてください。
大文字	テキスト内の大文字は、コマンド名、SQL予約語、キーワードを表します。
ゴシック・テキスト	メニュー選択項目やボタンなど、ユーザー・インタフェース項目を示すには、ゴシック・テキストが使用されます。
C>	C>はDOSプロンプトを表します。実際とは異なる場合があります。

関連資料

次のOracleマニュアルを参照することもできます。

タイトル	部品番号
『Oracle Forms Developer and Oracle Reports Developer アプリケーション作成ガイド リリース6i』	J00449-01

Developerビルトイン・パッケージ

Developerビルトイン・パッケージについて

Forms DeveloperおよびReports Developerには複数のクライアント側のビルトイン・パッケージが用意されており、アプリケーションの構築時やアプリケーション・コードのデバッグ時に参照できるPL/SQL構成体がいくつも含まれています。これらのビルトイン・パッケージは、パッケージSTANDARDの拡張機能としてインストールされるわけではありません。したがって、パッケージ内の構成体を参照する場合は常に、構成体名の前にパッケージ名（Text_IO.Put_Lineなど）を付ける必要があります。

用意されているビルトイン・パッケージは次のとおりです。

DDE	**Developerコンポーネント内の動的データ交換（DDE）がサポートされます。**
DEBUG	**PL/SQLプログラム単位のデバッグに必要なプロシージャ、ファンクション、例外が提供されます。**
EXEC_SQL	**Developerアプリケーション用に記述されたPL/SQLコード内で動的SQLを実行するためのプロシージャおよびファンクションが提供されます。**
LIST	**文字列（VARCHAR2）のリストを作成および維持するためのプロシージャ、ファンクション、例外が提供されます。これにより、PL/SQLバージョン1で配列を作成できます。**
OLE2	**OLE2オートメーション・オブジェクトの属性の作成、操作、アクセスに必要なPL/SQLアプリケーション・プログラミング・インターフェース（API）が提供されます。**
ORA_FFI	**PL/SQLから外部（C）ファンクションを呼び出す際に必要なパブリック・インタフェースが提供されます。**
ORA_NLS	**現行の言語環境に関する高水準の情報を抽出できます。**
ORA_PROF	**PL/SQLプログラム単位の調整（特定のコードの実行時間の検査など）に使用するプロシージャ、ファンクションおよび例外が提供されます。**
TEXT_IO	**ファイル情報の読込みおよび書込みに必要な構成体が提供されます。**
TOOL_ENV	**Oracle環境変数との対話が可能になります。**
TOOL_ERR	**DEBUGなど他のビルトイン・パッケージによって作成されたエラー・スタックへのアクセスおよび操作が可能になります。**
TOOL_RES	**リソース・ファイルからすべてのテキスト・データを切り離し、移植性の高いPL/SQLコードを作成するために、リソース・ファイルから文字列リソースを抽出する方法が提供されます。**

次のパッケージは、Forms DeveloperおよびReports Developerの内部のみで使用されます。これらのパッケージには、外部で使用可能なサブプログラムは含まれていません。

ORA_DE	**プライベートPL/SQLサービスのためにDeveloperで使用される構成体が格納されています。**
STPROC	**Developer内部で、データベースに格納されているサブプログラムをコールする際に使用されます。このパッケージのコールは自動的に生成されます。**

DDEパッケージについて

DDEパッケージにより、Developerコンポーネント内での動的データ交換（DDE）サポートが提供されます。

動的データ交換（DDE）とは、Windows95/NTのアプリケーション間で通信し、データを交換できるメカニズムのことです。DDEクライアントのサポートは、Developerのプロシージャ拡張機能として追加されます。DDEサポート用のPL/SQLパッケージにより、アプリケーション開発者がPL/SQLプロシージャやトリガー内でDDE機能にアクセスするのに必要なアプリケーション・プログラミング・インタフェース（API）が提供されます。

DDE機能により、Oracleのアプリケーションから他のDDE対応のWindows95/NTアプリケーション（サーバー）に、次の3つの方法で通信できます。

■ データのインポート

■ データのエクスポート

■ DDEサーバーに対するコマンドの実行

このリリースでは、DDEに次の機能は組み込まれていません。

■ データのリンク（アドバイス・トランザクション）

Oracleのアプリケーションでは、データ項目が変更されたとき、更新通知を自動受信できません。

■ サーバーのサポート

Oracleのアプリケーションでは、DDEクライアントからのコマンドまたはデータ要求には応答できません。したがって、Oracleのアプリケーション側でDDEの会話を開始する必要があります（ただし、データはクライアントとサーバーのどちらにも転送できます）。

サポート機能

他のDDEサーバー・アプリケーションの起動および停止に使います。

接続/切断機能

DDEサーバー・アプリケーションへの接続および切断に使います。

トランザクション機能

DDEサーバー・アプリケーションとのデータ交換に使います。

データ型変換機能

DDEデータ型定数を文字列に変換および復元する際に使います。また、DDE.Getformatnumを使用すると、Windowsで事前定義されていない新しいデータ形式を登録することができます。これらのファンクションによって変換されるのはデータ型定数のみで、データそのものは変換されない点に注意してください（DDEデータはすべて、PL/SQLではCHARデータ型で表示されます）。

注意: Developerのこれまでのリリースでは、Windows固有のDDE機能のコールがWindows以外のプラットフォームで正しくコンパイルおよび実行されるように、スタブ・ライブラリを連結する必要がありました。このリリースでは、連結の必要はなくなっています。ただし、Windows固有のビルトイン機能をWindows以外のプラットフォームで実行しようとすると、次のメッセージが表示されます。

FRM-40735: トリガー<名前>が未処理の例外を引き起こしました。
ORA-06509, 00000 PL/SQL: このパッケージのICDベクトルがありません。

Microsoft Windowsの定義済みデータ形式

定義済みデータ形式の例外については、例外を参照してください。

DDE.Cf_Bitmap	ビットマップ形式のデータです。
DDE.Cf_Dib	ビットマップ・データの先頭にBITMAPINFO構造体が含まれているメモリー・オブジェクトです。
DDE.Cf_Dif	データ交換形式（DIF）のデータです。
DDE.Cf_Dspbitmap	プライベート形式をビットマップで表現したデータです。このデータは、プライベート形式ではなく、ビットマップ形式で表示されます。
DDE.Cf_Dspmetafile-Pict	プライベート・データ形式をメタファイルで表現したデータです。このデータは、プライベート形式ではなく、メタファイル・ピクチャー形式で表示されます。
DDE.CF_Dsptext	プライベート・データ形式をテキストで表現したデータです。このデータは、プライベート形式ではなく、テキスト形式で表示されます。
DDE.Cf_Metafilepict	メタファイル形式のデータです。
DDE.Cf_Oemtext	OEMキャラクタ・セットに含まれているテキスト文字の配列です。各行末に、キャリッジ・リターンと改行（CR-LF）の組合せが付きます。データの終わりには、NULL文字が付きます。
DDE.Cf_Owner-Display	クリップボード所有者が表示する必要があるプライベート形式のデータです。
DDE.Cf_Palette	カラー・パレットのデータです。
DDE.Cf_Pendata	Windowsオペレーティング・システムのペン拡張機能用のデータです。
DDE.Cf_Riff	リソース交換ファイル形式（RIFF）のデータです。

DDE.Cf_Sylk	Microsoftシンボリック・リンク（SYLK）形式のデータです。
DDE.Cf_Text	テキスト文字の配列からなるデータです。各行末に、キャリッジ・リターンと改行（CR-LF）の組合せが付きます。データの終わりには、NULL文字が付きます。
DDE.Cf_Tiff	タグ付きイメージ・ファイル形式（TIFF形式）のデータです。
DDE.Cf_Wave	サウンド・ウェーブを記述するデータです。このデータ形式はCF_RIFFデータ形式のサブセットであり、RIFF WAVEファイルでしか使えません。

DDEの定義済みの例外

DDE.DDE_App_Failure	DDE.App_Beginコールで指定されたアプリケーション・プログラムを、起動できませんでした。
DDE.DDE_App_Not_Found	DDE.App_EndまたはDDE.App_Focusのコールで指定されたアプリケーションIDは、実行中のアプリケーションに対応しません。
DDE.DDE_Fmt_Not_Found	DDE.Getformatstrコールで指定された形式番号が不明です。
DDE.DDE_Fmt_Not_Reg	DDE.Getformatnumコールで指定された形式文字列は、事前定義済みの形式に対応しません。また、ユーザー定義形式として登録されませんでした。
DDE.DDE_Init_Failed	アプリケーション側でDDE通信を初期化できなかったため、DDEレイヤーのコールに失敗しました。
DDE.DDE_Param_Err	DDEパッケージ・ルーチンに、NULL値などの無効なパラメータが渡されました。
DDE.Dmlerr_Busy	サーバー・アプリケーションがビジーのため、トランザクションは失敗しました。
DDE.Dmlerr_Dataacktimeout	同期データ・トランザクションの要求は、タイムアウトになりました。
DDE.Dmlerr_Execack_ Timeout	同期実行トランザクションの要求は、タイムアウトになりました。
DDE.Dmlerr_Invalidparameter	パラメータの検証に失敗しました。次のような原因が考えられます。 ■ アプリケーションで使用されたデータ・ハンドルが、トランザクションで要求されている項目名ハンドルまたはクリップボード・データ形式で初期化されていない。 ■ アプリケーションで、無効な会話識別子が使用された。 ■ アプリケーションの複数のインスタンスで同じオブジェクトが使用された。
DDE.Dmlerr_Memory_Error	メモリー割当てに失敗しました。
DDE.Dmlerr_No_Conv_Established	クライアントの会話の確立に失敗しました。DDE.Initiateコールのサービスまたはトピック名により、エラーを生じることがあります。

DDE.Dmlerr_Notprocessed	トランザクションは失敗しました。DDE.PokeまたはDDE.Requestのトランザクションの項目名にエラーがある可能性があります。
DDE.Dmlerr_Not_Supported	DDEパッケージへのコールが行われましたが、現行のソフトウェア・プラットフォームではDDEがサポートされていません。
DDE.Dmlerr_Pokeacktimeout	同期DDE.Pokeトランザクションの要求は、タイムアウトになりました。
DDE.Dmlerr_Postmsg_Failed	PostMessageファンクションの内部コールに失敗しました。
DDE.Dmlerr_Server_ Died	トランザクションを完了する前にサーバーが終了しました。
DDE.Dmlerr_Sys_Error	DDEレイヤーで内部エラーが発生しました。

DEBUGパッケージについて

DEBUGパッケージには、PL/SQLプログラム単位のデバッグに必要なプロシージャ、ファンクションおよび例外が含まれています。これらのビルトイン・サブプログラムを使用して、デバッグ・トリガーを作成し、トリガーでブレークポイントを設定します。

LISTパッケージについて

LISTパッケージには、文字列（VARCHAR2）のリストを作成およびメンテナンスする際に使用するプロシージャ、ファンクションおよび例外が含まれています。これらのサービスを使用して、PL/SQLバージョン1で配列を作成することができます。

OLE2パッケージについて

OLE2パッケージによりOLE2オートメーション・オブジェクトの属性の作成、処理、アクセスを実行するPL/SQLアプリケーション・プログラム・インターフェース（API）が提供されます。

OLE2オートメーション・オブジェクトには、OLE2オートメーション・クライアントから操作または起動が可能な一連の属性とメソッドがカプセル化されています。OLE2パッケージを利用すれば、PL/SQLからOLE2オートメーション・サーバーへ直接アクセスできます。

オブジェクト・タイプおよびメソッド、構文指定の詳細は、各OLE2オートメーション・サーバーのOLE2プログラマ用マニュアルを参照してください。

ORA_FFIパッケージについて

ORA_FFIパッケージにより、動的ライブラリのC関数を起動する外部ファンクション・インタフェースが提供されます。

浮動小数点数引数は倍精度に変換する必要があることに注意してください。ANSI宣言を使用する必要がある場合は、コード内では倍精度浮動小数点数のみを使用します。

ORA_NLSパッケージについて

ORA_NLSパッケージを使用すると、現行の言語環境に関する高水準の情報を抽出できます。抽出した情報を利用して言語の属性を調べ、ローカルな日付および数値の書式が使用できるようにアプリケーションをカスタマイズすることができます。また、キャラクタ・セットの照合および一般キャラクタ・セットに関する詳細情報も取得できます。

また、現行の言語およびキャラクタ・セットの名前の取出しに必要な機能も用意されているので、特例をテストし利用するためのアプリケーションを作成できます。

ORA_NLS文字定数

現行言語の文字情報を検索するには次の定数を使います。定数はすべてPLS_INTEGER型で、各定数に整数値が割り当てられています。

名前	説明	整数	値
day1	第1日の正式名称	1	日曜日
day2	第2日の正式名称	2	月曜日
day3	第3日の正式名称	3	火曜日
day4	第4日の正式名称	4	水曜日
day5	第5日の正式名称	5	木曜日
day6	第6日の正式名称	6	金曜日
day7	第7日の正式名称	7	土曜日
day1_abbr	第1日の略称	8	日
day2_abbr	第2日の略称	9	月
day3_abbr	第3日の略称	10	火
day4_abbr	第4日の略称	11	水
day5_abbr	第5日の略称	12	木
day6_abbr	第6日の略称	13	金
day7_abbr	第7日の略称	14	土
mon1	第1月の正式名称	15	1月
mon2	第2月の正式名称	16	2月

mon3	第3月の正式名称	17	3月
mon4	第4月の正式名称	18	4月
mon5	第5月の正式名称	19	5月
mon6	第6月の正式名称	20	6月
mon7	第7月の正式名称	21	7月
mon8	第8月の正式名称	22	8月
mon9	第9月の正式名称	23	9月
mon10	第10月の正式名称	24	10月
mon11	第11月の正式名称	25	11月
mon12	第12月の正式名称	26	12月
mon1_abbr	第1月の略称	27	jan
mon2_abbr	第2月の略称	28	feb
mon3_abbr	第3月の略称	29	mar
mon4_abbr	第4月の略称	30	apr
mon5_abbr	第5月の略称	31	may
mon6_abbr	第6月の略称	32	jun
mon7_abbr	第7月の略称	33	jul
mon8_abbr	第8月の略称	34	aug
mon9_abbr	第9月の略称	35	sep
mon10_abbr	第10月の略称	36	oct
mon11_abbr	第11月の略称	37	nov
mon12_abbr	第12月の略称	38	dec
yes_str	問合せに対する肯定応答	39	yes
no_str	問合せに対する否定応答	40	no
am_str	AMのローカル等価	41	午前
pm_str	PMのローカル等価	42	午後
ad_str	ADのローカル等価	43	西暦
bc_str	BCのローカル等価	44	紀元前
decimal	10進文字	45	.
groupsep	グループ・セパレータ	46	,
int_currency	通貨記号	47	JPY
local_currency	各国通貨記号	48	\
local_date_fmt	ローカル日付書式	49	%Y年%m月%d日
local_time_fmt	ローカル時間書式	50	%H時%M分
default_date_fmt	Oracleのデフォルトの日付書式	51	YY-MM-DD
default_time_fmt	Oracleのデフォルトの時間書式	52	HH24:MI:SS
language	言語名	53	JAPANESE
language_abbr	ISO言語名略称	54	JA
character_set	デフォルトのキャラクタ・セット名	55	JA16EUC
territory	デフォルトの地域名	56	JAPAN

current_decimal	現行の小数点文字	57	.
current_groupsep	現行のグループ・セパレータ	58	,
current_currency	現行のローカル通貨記号	59	\
current_date_fmt	現行のOracle日付書式	60	YY-MM-DD
current_language	現行の言語	70	
current_territory	現行の地域	61	JAPAN
current_character_set	現行のキャラクタ・セット	62	JA16SJIS

ORA_NLS数値定数

現行言語の数値情報を検索するには、次の定数を使います。定数はすべてPLS_INTEGER型で、各定数に整数値が割り当てられています。

名前	説明	整数
decimal_places	通貨の小数点位置	63
sign_placement	符号の位置：0=前、1=後	64
initcap_month	月名の頭文字を大文字にする：0=NO、1=YES	65
initcap_day	曜日名の頭文字を大文字にする：0=NO、1=YES	66
week_start	週の最初の日：0=日曜日	67
week_num_calc	週数の計算：1=ISO、 0=非ISO	68
iso_alphabet	現行のISOアルファベット番号	69

ORA_PROFパッケージについて

ORA_PROFパッケージには、PL/SQLプログラム単位の調整時に使用するプロシージャ、ファンクションおよび例外が含まれています。このパッケージのサービスを使用して、作成したコードの実行にかかる時間を追跡することができます。

TEXT_IOパッケージについて

TEXT_IOパッケージには、ファイル情報の読込みおよび書込みに必要な構成体が含まれています。TEXT_IOで使用可能なプロシージャおよびファンクションは、次のカテゴリに分けられます。

ファイル操作	FILE_TYPEタイプ、FOPENおよびIS_OPENファンクション、FCLOSEプロシージャを使用すると、それぞれ、FILE_TYPE変数の定義、ファイルのオープン、オープンしたファイルのチェック、ファイルのクローズができます。
出力（書込み）操作	PUT、PUTF、PUT_LINEおよびNEW_LINEプロシージャを使用すると、オープン・ファイルへの情報の書込みまたはインタプリタへの情報の出力ができます。

出力（読込み）操作	GET_LINEプロシージャを使用すると、オープン・ファイルから1行読み込むことができます。

TEXT_IO構成体の使い方の例

ファイルの内容にエコーをかけるプロシージャの例を次に示します。このプロシージャでは、TEXT_IO構成体のコールが数回行われている点に注意してください。

```
PROCEDURE echo_file_contents IS
  in_file   Text_IO.File_Type;
  linebuf   VARCHAR2(1800);
  filename  VARCHAR2(30);
BEGIN
  filename:=GET_FILE_NAME('c:¥temp¥', File_Filter=>'Text Files
(*.txt)|*.txt|');
  in_file := Text_IO.Fopen(filename, 'r');
  LOOP
    Text_IO.Get_Line(in_file, linebuf);
    :text_item5:=:text_item5||linebuf||chr(10);
    --Text_IO.New_Line;
  END LOOP;
EXCEPTION
  WHEN no_data_found THEN
    Text_IO.Put_Line('Closing the file...');
    Text_IO.Fclose(in_file);
END;
```

TOOL_ENVパッケージについて

TOOL_ENVパッケージを使用すると、サブプログラムで使用する値を検索してOracle環境変数と対話できます。

TOOL_ERRパッケージについて

例外を使用してエラーを知らせるだけでなく、詳細なエラー情報を提供するビルトイン・パッケージ（DEBUGパッケージなど）もあります。この情報は"エラー・スタック"の形式で保管されます。

エラー・スタックには、詳細なエラー・コードおよびそれに関係付けられたエラー・メッセージが含まれています。スタック上のエラーには0（最も古い）から*n*-1（最も新しい）の順に索引が付いています。ここで*n*は現在スタックにあるエラーの総数です。TOOL_ERRパッケージで提供されるサービスを使用して、エラー・スタックへアクセスし、操作することができます。

TOOL_ERR構成体の使い方の例

次のプロシージャでは、Debug.Interpretビルトイン・パッケージが生成したエラーをTOOL_ERRパッケージの構成体を使用して処理する方法を示します。

```
PROCEDURE error_handler IS
/* Call a built-in that interprets a command */
BEGIN
  Debug.Interpret('.ATTACH LIB LIB1');
EXCEPTION
/*
** Check for a specific error code, print the
** message, then discard the error from the stack
** If the error does not match, then raise it.
*/
  WHEN OTHERS THEN
    IF Tool_Err.Code = Tool_Err.Encode('DEPLI',18) THEN
      Text_IO.Put_Line(Tool_Err.Message);
      Tool_Err.Pop;
    ELSE
      RAISE;
    END IF;
END;
```

例外処理コードがTOOL_ERR構成体を使用しなかった場合、エラー警告が出され、メッセージ「PDE-PLI018:ライブラリLIB1が見つかりません」が表示されます。Tool_Err構成体を使用すると、エラーが捕捉され、インタプリタにメッセージが送られます。

TOOL_RESパッケージについて

TOOL_RESパッケージを使用すると、リソース・ファイルから文字列リソースを抽出できます。これは、リソース・ファイルからすべての文字列データを切り離すことで、言語間でのPL/SQLコードの移植を簡単に行うためです。

リソース・ファイルの作成

既存のリソース・ファイルから文字列データを抽出することも、次のユーティリティを使用して文字列データが入っているリソース・ファイルを作成することもできます。

RSPA60	**テキスト・ファイル（.PRN）からリソース・ファイル（.RES）を生成するためのユーティリティです。生成したリソース・ファイルは、Tool_Resパッケージで使用できます。**
RSPR60	**リソース・ファイル（.RES）をテキスト・ファイル（.PRN）に変換するためのユーティリティです。**

これらのユーティリティは、Oracle Terminalに付属しており、この製品のインストール時に自動的に組み込まれます。使用しているプラットフォームで、これらのユーティリティのコマンド・ライン構文を表示するには、引数を指定しないでユーティリティを実行します。

Microsoft Windowsでは、エクスプローラまたはファイル マネージャからこれらの実行プログラムを起動し、コマンド・ライン構文を表示できます。引数付きで実行プログラムを実行するには、「ファイル名を指定して実行」を使います。

リソース・ファイルの構文

リソース・ファイルに必要な文字列を作成する場合には、次の構文を使います。

```
Resource      "resource_name"
Type          "string"
Content
 table
 {
  string string 1 character_count
  "content of string"
 }
```

パラメータの内容は次のとおりです。

resource_name	**Tool_Res.Rfreadで参照できる一意の名前。**
character_count	**文字列の文字数。**
content of string	**実際の文字列。**

例

次のテキスト・ファイルHELLO.PRNは、RESPA60ユーティリティを使用してリソース・ファイルHELLO.RES内に生成されます。

```
Resource "hello_world"
Type "string"
Content
 table
 {
  string string 1 12
  "Hello World!"
 }

Resource "goodbye_world"
Type "string"
Content
 table
 {
  string string 1 14
  "Goodbye World!"
 }
```

このテキスト・ファイルは次のプログラム単位で参照されます。

```
PROCEDURE get_res IS
  resfileh Tool_Res.Rfhandle;
  hellor   VARCHAR2(16);
  goodbyer VARCHAR2(16);
BEGIN
/*Open the resource file we generated */
  resfileh:=Tool_Res.Rfopen('hello.res');

/*Get the resource file strings*/
  hellor:=Tool_Res.Rfread(resfileh, 'hello_world');
  goodbyer:=Tool_Res.Rfread(resfileh, 'goodbye_world');

/*Close the resource file*/
  Tool_Res.Rfclose(resfileh);

/*Print the resource file strings*/
  Text_IO.Put_Line(hellor);
  Text_IO.Put_Line(goodbyer);
END;
```

EXEC_SQLパッケージについて

EXEC_SQLパッケージを使用すると、いくつかの異なる接続を同時に使用して、複数のOracleデータベース・サーバーにアクセスできます。また、Developerに同梱されるOpen Client Adapter(OCA)を使用して、ODBCデータ・ソースに接続することもできます。Oracle以外のデータ・ソースにアクセスするには、OCAと適切なODBCドライバをインストールする必要があります。

EXEC_SQLパッケージには、PL/SQLプロシージャの中で動的SQLを実行するためのプロシージャとファンクションが含まれています。DBMS_SQLパッケージと同様に、SQL文は実行時にソース・プログラムのみに渡されるか、またはソース・プログラムによってビルドされる文字列に格納されます。EXEC_SQLパッケージを使用すると、あらゆるデータ操作言語（DML）やデータ定義言語（DDL）文を発行できます。

EXEC_SQLパッケージは、次のような点でDMBS_SQLパッケージと異なります。

- アドレスによるバインドではなく、値によるバインドを使用します。
- OUTバインド・パラメータの値を取り出すためにEXEC_SQL.Variable_Valueを使用する必要があります。
- 結果セットの値を取り出すために、行をフェッチした後でEXEC_SQL.Column_Valueを使用する必要があります。
- CHAR、RAW、LONG、ROWIDのデータをサポートしていません。

■ CANCEL_CURSORプロシージャまたはファンクションが用意されていません。

■ 配列インタフェースをサポートしていません。

■ PL/SQL変数の値としてNULLがサポートされているため、標識変数は必要ありません。

■ PL/SQL表またはレコード・タイプをサポートしていません。

DBMS_SQLパッケージの詳細は、『Oracle8アプリケーション開発者ガイド』を参照してください。

接続ハンドルとカーソル・ハンドル

Developerアプリケーションでは、1つ以上のデータベースに対して同時に複数の接続を確立できます。ただし、その中には必ずプライマリ・データベース接続が1つあり、これをプライマリDeveloper接続と呼びます。

ハンドルは、DeveloperアプリケーションでOracle接続またはODBC接続を参照するときに使います。プライマリ・データベースまたはその他のデータベースへの接続をオープンすると、タイプEXEC_SQL.ConnTypeの接続ハンドルが作成され、接続の参照に使用されます。各接続ハンドルが、1つのデータベース接続を表します。

接続ハンドルでカーソルをオープンすると、タイプEXEC_SQL.CursTypeのカーソル・ハンドルが作成され、その接続のカーソルの参照に使用されます。各接続ハンドルは、複数のカーソル・ハンドルを持つことができます。

接続とカーソルをオープンすると、データにアクセスできるようになります。複数の接続を同時にオープンした場合は、特定のハンドルを引数としてEXEC_SQLルーチンに明示的に入れることをお薦めします。

プライマリDeveloper接続からのみデータにアクセスする場合は、EXEC_SQLルーチンで接続を指定する必要はありません。EXEC_SQLルーチンにハンドルが指定されていないと、EXEC_SQL.Default_Connectionが自動的に呼び出されて、プライマリDeveloper接続を取得します。

問合せまたはOracle以外のストアド・プロシージャからの結果セットの取出し

EXEC_SQLパッケージは、別のOracleデータ・ソースやODBCデータ・ソースから1つのフォームやレポートに結果セットを取り出す必要があるときに、特に役立ちます。

結果セットを返す文を処理するには:

1 列ごとにEXEC_SQL.Define_Columnを使用して、値を受け取るための変数を指定します。

2 EXEC_SQL.Executeを呼び出して文を実行します。

3 EXEC_SQL.Fetch_Rowsを使用して、結果セットの行を取り出します。

4 EXEC_SQL.Column_Valueを使用して、EXEC_SQL.Fetch_Rowsによって取り出された各列の値を取得します。

5 EXEC_SQL.Fetch_Rowsが0を返すまで、ステップ3と4を繰り返します。

EXEC_SQLの定義済みの例外

EXEC_SQL.Invalid_Connection	無効な接続ハンドルが渡されました。
EXEC_SQL.Package_Error	一般エラーです。EXEC_SQL.Last_Error_CodeとEXEC_SQL.Last_Error_Mesgを使用してエラーを取り出してください。
EXEC_SQL.Invalid_Column_Number	EXEC_SQL.Describe_Columnプロシージャが、結果セットに存在しない列番号を検出しました。
EXEC_SQL.Value_Error	EXEC_SQL.Column_Valueが、EXEC_SQL.Define_Columnによって取り出された元の値と異なる値を検出しました。

EXEC_SQLパッケージの使い方

接続に対する任意のSQLの実行

2つのデータベース間でのデータのコピー

Oracle以外のデータベースのストアド・プロシージャの実行およびその結果セットのフェッチ

接続に対する任意のSQLの実行

次のプロシージャは、SQL文と、'ユーザー[/パスワード][@データ・ソース]'形式のオプションの接続文字列を渡します。接続文字列が渡された場合、プロシージャはデータ・ソースに対してSQL文を実行します。それ以外の場合は、プライマリDeveloper接続に対してSQL文をインプリメントします。

```
PROCEDURE exec (sql_string IN VARCHAR2,
  connection_string IN VARCHAR2 DEFAULT NULL)
IS
  connection_id EXEC_SQL.ConnType;
  cursor_number EXEC_SQL.CursType;
  ret          PLS_INTEGER;
BEGIN

IF connection_string IS NULL THEN
  connection_id := EXEC_SQL.DEFAULT_CONNECTION;
ELSE
  connection_id :=

EXEC_SQL.OPEN_CONNECTION(connection_string);
END IF;
cursor_number :=
   EXEC_SQL.OPEN_CURSOR(connection_id);


EXEC_SQL.PARSE(connection_id, cursor_number,
sql_string);

ret := EXEC_SQL.EXECUTE(connection_id,
cursor_number);




EXEC_SQL.CLOSE_CURSOR(connection_id,
cursor_number);
EXEC_SQL.CLOSE_CONNECTION(connection_id);







EXCEPTION
  WHEN EXEC_SQL.PACKAGE_ERROR THEN
    TEXT_IO.PUT_LINE('ERROR (' ||
      TO_CHAR(EXEC_SQL.LAST_ERROR_CODE
       (connection_id)) || '): ' ||
      EXEC_SQL.LAST_ERROR_MESG(connection_id));
    IF EXEC_SQL.IS_CONNECTED(connection_id) THEN
      IF EXEC_SQL.IS_OPEN(connection_id,
        cursor_number) THEN
        EXEC_SQL.CLOSE_CURSOR(connection_id,
        cursor_number);
      END IF;
      EXEC_SQL.CLOSE_CONNECTION(connection_id);
    END IF;
END;
```

-- 新しい接続をオープンします。接続文字列が空の場合は、プライマリDeveloper接続を使うものと判断します。

-- SQL文の実行に必要な接続のカーソルをオープンします。

-- その接続のSQL文を解析します。

-- SQL文を実行します。接続先がOracleの場合は解析時にすべてのDDLが処理されますが、Oracle以外のデータ・ソースの場合はこの処理は保証されません。

-- カーソルをクローズします。
-- 接続が完了しました。connection_idは、EXEC_SQL.OPEN_CONNECTIONまたはEXEC_SQL.DEFAULT_CONNECTIONの呼出しから得られます。どちらの場合も、EXEC_SQL.CLOSE_CONNECTIONを呼び出す必要があります。connection_idがEXEC_SQL.OPEN_CONNECTIONによって取得された場合は、EXEC_SQL.CLOSE_CONNECTIONによって接続が終了します。connection_idがEXEC_SQL.DEFAULT_CONNECTIONによって取得された場合は、EXEC_SQL.CLOSE_CONNECTIONでは接続は終了しませんが、EXEC_SQLパッケージ固有のリソースは解放されます。

-- これはEXEC_SQLパッケージが呼び出す一般エラーで、呼出しの1つに予期しないエラーがあることを示します。エラー番号とエラー・メッセージが標準出力に表示されます。

2つのデータベース間でのデータのコピー

次のプロシージャでは動的SQLを特に使用する必要はありませんが、EXEC_SQLパッケージの概念を示しています。

このプロシージャは、ソース表（ソース接続）からコピー先表（コピー先接続）に行をコピーします。ここでは、ソース表とコピー先表に次の列があると想定します。

```
ID of type NUMBER
NAME of type VARCHAR2(30)
BIRTHDATE of type DATE
```

```
PROCEDURE copy (source_table IN VARCHAR2,
  destination_table IN VARCHAR2,
  source_connection IN VARCHAR2 DEFAULT NULL,
  destination_connection IN VARCHAR2 DEFAULT NULL)
IS
  id               NUMBER;
  name             VARCHAR2(30);
  birthdate         DATE;
  source_connid     EXEC_SQL.ConnType;
  destination_connid EXEC_SQL.ConnType;
  source_cursor     EXEC_SQL.CursType;
  destination_cursor EXEC_SQL.CursType;
  ignore            PLS_INTEGER
BEGIN
  IF source_connection IS NULL THEN
    source_connid := EXEC_SQL.DEFAULT_CONNECTION;
  ELSE
    source_connid :=
      EXEC_SQL.OPEN_CONNECTION(source_connection);
  END IF;
  IF destination_connection IS NULL THEN
    destination_connid := EXEC_SQL.CURR_CONNECTION;
  ELSE
    destination_connid :=
      EXEC_SQL.OPEN_CONNECTION(destination_connection);
  END IF;
```

-- 接続をオープンします。ユーザーが2番目の接続を指定しない場合は、プライマリDeveloper接続が使われます。

```
 source_cursor := EXEC_SQL.OPEN_CURSOR(source_connid);
  EXEC_SQL.PARSE(source_connid, source_cursor,
    'SELECT id, name, birthdate FROM ' || source_table);
  EXEC_SQL.DEFINE_COLUMN(source_connid, source_cursor, 1,
    id);
  EXEC_SQL.DEFINE_COLUMN(source_connid, source_cursor, 2,
    name, 30);
  EXEC_SQL.DEFINE_COLUMN(source_connid, source_cursor, 3,
    birthdate);
  ignore := EXEC_SQL.EXECUTE(source_connid, source_cursor);
  destination_cursor :=
    EXEC_SQL.OPEN_CURSOR(destination_connid);
EXEC_SQL.PARSE(destination_connid, destination_cursor,
    'INSERT INTO ' || destination_table || '
    (id, name, birthdate) VALUES (:id, :name, :birthdate)');
  LOOP
    IF EXEC_SQL.FETCH_ROWS(source_connid, source_cursor) > 0
    THEN
      EXEC_SQL.COLUMN_VALUE(source_connid, source_cursor,
        1, id);
      EXEC_SQL.COLUMN_VALUE(source_connid, source_cursor,
        2, name);     EXEC_SQL.COLUMN_VALUE(source_connid,
source_cursor,
        3, birthdate);
      EXEC_SQL.BIND_VARIABLE(destination_connid,
        destination_cursor, ':id', id);
      EXEC_SQL.BIND_VARIABLE(destination_connid,
        destination_cursor, ':name', name);
      EXEC_SQL.BIND_VARIABLE(destination_connid,
        destination_cursor, ':birthdate', birthdate);
      ignore := EXEC_SQL.EXECUTE(destination_connid,
        destination_cursor);
    ELSE
      EXIT;
    END IF;
  END LOOP;
  EXEC_SQL.PARSE(destination_connid, destination_cursor,
    'commit');
  ignore := EXEC_SQL.EXECUTE(destination_connid,
    destination_cursor);
```

-- ソース表から選択するカーソルを用意します。

-- コピー先表に挿入するカーソルを用意します。

-- ソース表から行をフェッチして、コピー先表に挿入します。
-- 行の列値を取得します。これらはローカル変数として格納されます。

-- コピー先表に挿入するカーソルに値をバインドします。

-- コピーする行がなくなりました。

-- コピー先カーソルをコミットします。

```
  EXEC_SQL.CLOSE_CURSOR(destination_connid,
    destination_cursor);
  EXEC_SQL.CLOSE_CURSOR(source_connid, source_cursor);
  EXEC_SQL.CLOSE_CONNECTION(destination_connid);
  EXEC_SQL.CLOSE_CONNECTION(source_connid);
EXCEPTION
  WHEN EXEC_SQL.PACKAGE_ERROR THEN
    IF EXEC_SQL.LAST_ERROR_CODE(source_connid) != 0 THEN
      TEXT_IO.PUT_LINE('ERROR (source: ' ||
        TO_CHAR(EXEC_SQL.LAST_ERROR_CODE(source_connid))
        || '): ' ||
        EXEC_SQL.LAST_ERROR_MESG(source_connid));
    END IF;
    IF EXEC_SQL.LAST_ERROR_CODE(destination_connid) != 0 THEN
      TEXT_IO.PUT_LINE('ERROR (destination: ' ||

TO_CHAR(EXEC_SQL.LAST_ERROR_CODE(destination_connid))
        || '): ' ||
        EXEC_SQL.LAST_ERROR_MESG(destination_connid));
    END IF;
    IF EXEC_SQL.IS_CONNECTED(destination_connid) THEN
      IF EXEC_SQL.IS_OPEN(destination_connid,
        destination_cursor) THEN
        EXEC_SQL.CLOSE_CURSOR(destination_connid,
          destination_cursor);
      END IF;
      EXEC_SQL.CLOSE_CONNECTION(destination_connid);
    END IF;
    IF EXEC_SQL.IS_CONNECTED(source_connid) THEN
      IF EXEC_SQL.IS_OPEN(source_connid, source_cursor) THEN
        EXEC_SQL.CLOSE_CURSOR(source_connid, source_cursor);
      END IF;

      EXEC_SQL.CLOSE_CONNECTION(source_connid);
    END IF;
END;
```

-- すべてをクローズします。

-- これはEXEC_SQLパッケージが呼び出す一般エラーです。ソース接続またはコピー先接続のエラーに関する情報（エラー番号とメッセージ）を取得します。

-- すべての接続とカーソルをクローズします。

Oracle以外のデータベースのストアド・プロシージャの実行およびその結果セットのフェッチ

次のプロシージャは、結果セットを返すMicrosoft SQL Serverのストアド・プロシージャを実行し、その結果セットをフェッチします。ストアド・プロシージャは次のとおりです。

```
create proc example_proc @id integer as
  select ename from emp where empno = @id
```

このプロシージャは、プライマリDeveloper接続（それがSQL Serverの接続であると想定して）でストアド・プロシージャを実行し、返されるすべての行を出力します。また、ここではプロシージャ例3の実行前に、プライマリDeveloper接続がすでにオープンされていると想定します。さもないと、エラーが発生します。

```
CREATE PROCEDURE example3 (v_id IN NUMBER) IS
  v_ename VARCHAR2(20);
  v_cur EXEC_SQL.CursType;
  v_rows INTEGER;
BEGIN
  v_cur := EXEC_SQL.OPEN_CURSOR;


  EXEC_SQL.PARSE(v_cur, '{ call example_proc ( :v_id ) }');
  EXEC_SQL.BIND_VARIABLE(v_cur, ':v_id', v_id);
  EXEC_SQL.DEFINE_COLUMN(v_curs, 1, v_ename, 20);
  v_rows := EXEC_SQL.EXECUTE(v_curs);
  WHILE EXEC_SQL.FETCH_ROWS(v_curs) > 0 LOOP
    EXEC_SQL.COLUMN_VALUE(v_curs, 1, v_ename);
    TEXT_IO.PUT_LINE('Ename = ' || v_ename);
  END LOOP;
  EXEC_SQL.CLOSE_CURSOR(v_cur);
  EXCEPTION
    WHEN EXEC_SQL.PACKAGE_ERROR THEN
      TEXT_IO.PUT_LINE('ERROR (' ||
        TO_CHAR(EXEC_SQL.LAST_ERROR_CODE)
        || '): ' ||
        EXEC_SQL.LAST_ERROR_MESG);
      EXEC_SQL.CLOSE_CURSOR(v_cur);
    WHEN EXEC_SQL.INVALID_CONNECTION THEN
      TEXT_IO.PUT_LINE('ERROR: Not currently connected
        to a database');
END example3;
```

-- 接続ハンドルが渡されない場合、EXEC_SQLはプライマリDeveloper接続を使います。

-- ODBCデータ・ソースに対するストアド・プロシージャを呼び出すには、ODBC構文を使いますが、パラメータはOracleパラメータとして指定する必要があります。

-- デフォルト接続がないと、例外INVALID_CONNECTIONが呼び出されます。

パッケージ・サブプログラムのアルファベット順リスト

DDE.App_Begin

DDE.App_End

DDE.App_Focus

DDE.DMLERR_Not_Supported

DDE.Execute

DDE.Getformatnum

DDE.Getformatstr

DDE.Initiate

DDE.IsSupported

DDE.Poke

DDE.Request

DDE.Terminate

Debug.Break

Debug.Getx

Debug.Interpret

Debug.Setx

Debug.Suspend

EXEC_SQL.Open_Connection

EXEC_SQL.Curr_Connection

EXEC_SQL.Default_Connection

EXEC_SQL.Open_Cursor

EXEC_SQL.Parse

EXEC_SQL.Describe_Column

EXEC_SQL.Bind_Variable

EXEC_SQL.Define_Column

EXEC_SQL.Execute

EXEC_SQL.Execute_And_Fetch

EXEC_SQL.Fetch_Rows

EXEC_SQL.More_Result_Sets

EXEC_SQL.Column_Value

EXEC_SQL.Variable_Value

EXEC_SQL.Is_Open

EXEC_SQL.Close_Cursor

EXEC_SQL.Is_Connected

EXEC_SQL.Is_OCA_Connection

EXEC_SQL.Close_Connection

EXEC_SQL.Last_Error_Position

EXEC_SQL.Last_Row_Count

EXEC_SQL.Last_SQL_Function_Code

EXEC_SQL.Last_Error_Code

EXEC_SQL.Last_Error_Mesg

List.Appenditem

List.Destroy

List.Deleteitem

List.Fail

List.Getitem

List.Insertitem

List.Listofchar

List.Make

List.Nitems

List.Prependitem

OLE2.Add_Arg

OLE2.Create_Arglist

OLE2.Destroy_Arglist

OLE2.Get_Char_Property

OLE2.Get_Num_Property

OLE2.Get_Obj_Property

OLE2.Invoke

OLE2.Invoke_Num

OLE2.Invoke_Char

OLE2.Invoke_Obj

OLE2.IsSupported

OLE2.List_Type

OLE2.Obj_Type

OLE2.OLE_Not_Supported

OLE2.Release_Obj

OLE2.Set_Property

Ora_FFI.Find_Function

Ora_FFI.Find_Library

Ora_FFI.Funchandletype

Ora_FFI.Generate_Foreign

Ora_FFI.Is_Null_Ptr

Ora_FFI.Libhandletype

Ora_FFI.Load_Library

Ora_FFI.Pointertype

Ora_FFI.Register_Function

Ora_FFI.Register_Parameter

Ora_FFI.Register_Return

Ora_NLS.American

Ora_NLS.American_Date

Ora_NLS.Bad_Attribute

Ora_NLS.Get_Lang_Scalar

Ora_NLS.Get_Lang_Str

Ora_NLS.Linguistic_Collate

Ora_NLS.Linguistic_Specials

Ora_NLS.Modified_Date_Fmt

Ora_NLS.No_Item

Ora_NLS.Not_Found

Ora_NLS.Right_to_Left

Ora_NLS.Simple_CS

Ora_NLS.Single_Byte

Ora_Prof.Bad_Timer

Ora_Prof.Create_Timer

Ora_Prof.Destroy_Timer

Ora_Prof.Elapsed_Time

Ora_Prof.Reset_Timer

Ora_Prof.Start_Timer

Ora_Prof.Stop_Timer

Text_IO.FClose

Text_IO.File_Type

Text_IO.Fopen

Text_IO.Is_Open

Text_IO.Get_Line

Text_IO.New_Line

Text_IO.Put

Text_IO.PutF

Text_IO.Put_Line

Tool_Env.Getvar

Tool_Err.Clear

Tool_Err.Code

Tool_Err.Encode

Tool_Err.Message

Tool_Err.Nerrors

Tool_Err.Pop

Tool_Err.Tool_Error

Tool_Err.Toperror

Tool_Res.Bad_File_Handle

Tool_Res.Buffer_Overflow

Tool_Res.File_Not_Found

Tool_Res.No_Resource

Tool_Res.Rfclose

Tool_Res.Rfhandle

Tool_Res.Rfopen

Tool_Res.Rfread

DDEパッケージ

DDEパッケージ

DDE.App_Begin

説明

アプリケーション・プログラムが起動され、アプリケーション識別子が返されます。

構文

```
FUNCTION DDE.App_Begin
   (AppName VARCHAR2,
    AppMode PLS_INTEGER)
RETURN PLS_INTEGER;
```

パラメータ

AppName	アプリケーション名。

AppMode	**アプリケーション起動モード。** **App_Mode_Normal　アプリケーション・ウィンドウが標準サイズで起動されます。** **App_Mode_Minimized　アプリケーション・ウィンドウが最小サイズで起動されます。** **App_Mode_Maximized　アプリケーション・ウィンドウが最大サイズで起動されます。**

戻り値

アプリケーション識別子

使用上の注意

アプリケーション名にはパスを含むことができます。アプリケーション名にパスが含まれていない場合には、次の順番で各ディレクトリが検索されます。

- 現行ディレクトリ
- Windowsディレクトリ
- Windows SYSTEMディレクトリ
- 現行タスク用の実行可能ファイルが入っているディレクトリ

*AppName*では、アプリケーション・プログラム名の後ろに引数を付けることができます。アプリケーション・プログラム名と引数の間はスペースで区切ります。

アプリケーションは、*AppMode*で指定したサイズで起動します。標準サイズ、最小サイズまたは最大サイズのいずれも指定できます。

同じアプリケーション・ウィンドウについて、すべてのDDE.App_EndおよびDDE.App_Focusコールに、DDE.App_Beginによって戻されたアプリケーション識別子を使用する必要があります。

DDE.App_Beginの例

```
 /*
** Start MS Excel with spreadsheet emp.xls loaded
*/
DECLARE
  AppID  PLS_INTEGER;
BEGIN
    AppID := DDE.App_Begin('c:¥excel¥excel.exe emp.xls',
     DDE.App_Mode_Minimized);
END;
```

DDE.App_End

説明

Dde_App_Beginで起動したアプリケーション・プログラムを終了します。

構文

```
PROCEDURE DDE.App_End
   (AppID PLS_INTEGER);
```

パラメータ

AppID	DDE.App_Beginによって戻されるアプリケーション識別子。

使用上の注意

アプリケーションは標準のWindows方式でも終了できます。たとえば、「コントロール」メニューをダブルクリックすれば終了できます。

DDE.App_Endを使用してアプリケーション・プログラムを終了するには、DDE.App_Beginをコールしてアプリケーションを起動しておく必要があります。

DDE.App_Endの例

```
 /*
** Start Excel, perform some operations on the
** spreadsheet, then close the application.
*/
DECLARE
  AppID  PLS_INTEGER;
BEGIN
    AppID := DDE.App_Begin('c:¥excel¥excel.exe emp.xls',
    DDE.App_Mode_Normal);
  ...
   DDE.App_End(AppID);
END;
```

DDE.App_Focus

説明

DDE.App_Beginで起動したアプリケーション・プログラムをアクティブ化します。

構文

```
PROCEDURE DDE.App_Focus
   (AppID PLS_INTEGER);
```

パラメータ

AppID	DDE.App_Beginによって戻されるアプリケーション識別子。

使用上の注意

アプリケーションは標準のWindows方式でもアクティブ化できます。たとえば、アプリケーション・ウィンドウ内のどこかをクリックすればアクティブ化できます。

DDE.App_Focusを使用してアプリケーション・プログラムをアクティブにするには、DDE.App_Beginをコールしてアプリケーションを起動しておく必要があります。

DDE.App_Focusの例

```
 /*
** Start Excel, then activate the application window
*/
DECLARE
   AppID  PLS_INTEGER;
BEGIN
  AppID := DDE.App_Begin('c:¥excel¥excel.exe',
    DDE.App_Mode_Maximized);
  DDE.App_Focus(AppID);
END;
```

DDE.Execute

説明

受信側サーバー・アプリケーションが受付け可能なコマンドを実行します。

構文

```
PROCEDURE DDE.Execute
   (ConvID  PLS_INTEGER,
    CmdStr  VARCHAR2,
    Timeout PLS_INTEGER);
```

パラメータ

ConvID	DDE.Initiateによって戻されるDDE対話識別子。
CmdStr	サーバーで実行されるコマンド文字列。
Timeout	タイムアウトするまでの時間（ミリ秒単位）。

使用上の注意

CmdStr 受信側サーバー・アプリケーションが受付け可能なコマンド文字列が実行されます。

*Timeout*では、DDEサーバー・アプリケーションからの応答をどれだけの時間待つかをミリ秒単位で指定します。無効な数値（負数など）を指定した場合には、デフォルト値の1000が使用されます。

DDE.Executeの例

```
 /*
** Initiate Excel, then perform a recalculation
*/
DECLARE
  ConvID  PLS_INTEGER;
BEGIN
   ConvID := DDE.Initiate('EXCEL', 'abc.xls');
   DDE.Execute(ConvID, '[calculate.now()]', 1000);
END;
```

DDE.Getformatnum

説明

指定したデータ形式名が変換または登録され、データ形式文字列が数値表現で返されます。

構文

```
FUNCTION DDE.Getformatnum
   (DataFormatName VARCHAR2)
RETURN PLS_INTEGER;
```

パラメータ

DataFormat-Name	**データ形式名の文字列。**

使用上の注意

DDE.Getformatnumではデータ形式が文字列から数値に変換されます。この数値は、DDE.PokeトランザクションとDDE.RequestトランザクションでDataFormat変数を表すのに使えます。

指定した名前が登録されていない場合は、DDE.Getformatnumが新規に登録し、一意の形式番号を戻します。これは唯一、DDE.PokeまたはDDE.Requestトランザクション内で、事前定義されていない形式を使用するための方法です。

DDE.Getformatnumの例

```
 /*
** Get predefined format number for "CF_TEXT" (should
** return CF_TEXT=1) then register a user-defined
** data format called "MY_FORMAT"
*/
DECLARE
```

```
  FormatNum    PLS_INTEGER;
  MyFormatNum  PLS_INTEGER;
BEGIN
  FormatNum := DDE.Getformatnum('CF_TEXT');
  MyFormatNum := DDE.Getformatnum('MY_FORMAT');
END;
```

DDE.Getformatstr

説明

データ形式番号がデータ形式名文字列に変換されます。

構文

```
FUNCTION DDE.Getformatstr
   (DataFormatNum PLS_INTEGER)
RETURN VARCHAR2;
```

パラメータ

DataFormat-Num	**データ形式番号。**

戻り値

指定したデータ形式番号を文字列で表したもの。

使用上の注意

DDE.Getformatstrでは、データ形式番号が有効な場合にデータ形式名が返されます。有効な形式番号とは、事前定義済みの形式およびDDE.Getformatnumで登録されたユーザー定義の形式です。

DDE.Getformatstrの例

```
 /*
** Get a data format name (should return the string
** 'CF_TEXT')
*/
DECLARE
  FormatStr  VARCHAR2(80);
BEGIN
  FormatStr := DDE.Getformatstr(CF_TEXT);
END;
```

DDE.Initiate

説明

サーバー・アプリケーションとDDEコマンドを開始します。

構文

```
FUNCTION DDE.Initiate
   (Service VARCHAR2,
    Topic   VARCHAR2)
RETURN PLS_INTEGER;
```

パラメータ

Service	サーバー・アプリケーションのDDEサービス・コード。
Topic	対話のトピック名。

戻り値

DDE対話識別子。

使用上の注意

*Service*および*Topic*の値は、そのDDEサーバー・アプリケーションがサポートする値によって異なります。*Service*は通常はアプリケーション・プログラムの名前です。ファイル・ベースの文書を操作するアプリケーションの場合、*Topic*は通常は文書のファイル名です。システム・トピックは通常は各サービスによってサポートされます。

同じDDE会話について、すべてのDDE.Execute、DDE.Poke、DDE.RequestおよびDDE.Terminateコールには、DDE.Initiateによって戻された対話識別子を使用する必要があります。

対話識別子が交換されないと、アプリケーションでは、複数のサービスおよびトピックについて、1度に複数のやりとりが発生する場合があります。

やりとりを終了するには、DDE.Terminateを使います。

DDE.Initiateの例

```
 /*
** Open a DDE Conversation with MS Excel on
** topic abc.xls
*/
DECLARE
  ConvID  PLS_INTEGER;
BEGIN
  ConvID := DDE.Initiate('EXCEL', 'abc.xls');
END;
```

DDE.IsSupported

説明

DDEパッケージが現行プラットフォームでサポートされているかどうか確認するのに使います。

構文

```
DDE.ISSUPPORTED
```

戻り値

DDEがプラットフォームでサポートされる場合はTRUE、サポートされない場合はFALSE。

DDE.IsSupportedの例

```
/*
** Before calling a DDE object in platform independent code,
** use this predicate to determine if DDE is supported on the
** current platform.
*/
IF (DDE.ISSUPPORTED)THEN
       . . . PL/SQL code using the DDE package
ELSE
       . . . message that DDE is not supported
END IF;
```

DDE.DMLERR_Not_Supported

説明

DDEパッケージをコールしたときに、現行ソフトウェア・プラットフォームではそのDDEがサポートされていない場合に、この例外がコールされます。

構文

```
DDE.DMLERR_NOT_SUPPORTED EXCEPTION;
```

DDE.Poke

説明

データがサーバー・アプリケーションに送信されます。

構文

```
PROCEDURE DDE.Poke
   (ConvID     PLS_INTEGER,
    Item       VARCHAR2,
    Data       VARCHAR2,
    DataFormat PLS_INTEGER,
    Timeout    PLS_INTEGER);
```

パラメータ

ConvID	DDE.Initiateによって戻されるDDE対話識別子。
Item	データの送信先となるデータ項目名。
Data	送信するデータ・バッファ。
DataFormat	送信データの形式。
Timeout	タイムアウトするまでの時間（ミリ秒単位）。

使用上の注意

*Item*の値は、現行の対話トピックについてサーバー・アプリケーションがサポートする値によって異なります。

*DataFormat*では事前定義済みのデータ形式の定数を使えます。

サーバー・アプリケーションが認識できる形式であれば、DDE.Getformatnumで登録されたユーザー定義の形式も使用できます。指定したデータ形式がサーバー・アプリケーションで処理されるかどうかの確認は、ユーザーが各自の責任で行う必要があります。

*Timeout*では、DDEサーバー・アプリケーションからの応答をどれだけの時間待つかをミリ秒単位で指定します。無効な数値（負数など）を指定した場合には、デフォルト値の1000が使用されます。

DDE.Pokeの例

```
 /*
** Open a DDE Conversation with MS Excel on topic
** abc.xls and end data "foo" to cell at row 2,
** column 2
*/
DECLARE
  ConvID  PLS_INTEGER;
BEGIN
  ConvID = DDE.Initiate('EXCEL', 'abc.xls');
  DDE.Poke(ConvID, 'R2C2', 'foo', DDE.CF_TEXT, 1000);
END;
```

DDE.Request

説明

サーバー・アプリケーションのデータが要求されます。

構文

```
PROCEDURE DDE.Request
   (ConvID     PLS_INTEGER,
    Item       VARCHAR2,
    Buffer     VARCHAR2,
    DataFormat PLS_INTEGER,
    Timeout    PLS_INTEGER);
```

パラメータ

ConvID	DDE.Initiateによって戻されるDDE対話識別子。
Item	要求するデータ項目の名前。
Buffer	結果データ・バッファ。
DataFormat	要求するバッファの形式。
Timeout	タイムアウトするまでの時間（ミリ秒単位）。

使用上の注意

*Item*の値は、現行の対話トピックについてサーバー・アプリケーションがサポートする値によって異なります。

要求したデータを扱うのに必要な戻りデータ・バッファのサイズ確認は、ユーザーが各自の責任で行う必要があります。要求したデータよりバッファ・サイズが小さいと、データは切り捨てられます。

*DataFormat*では事前定義済みのデータ形式の定数を使えます。

サーバー・アプリケーションが認識できる形式であれば、DDE.Getformatnumで登録されたユーザー定義の形式も使用できます。指定したデータ形式をサーバー・アプリケーションが扱うかどうかの確認は、ユーザーが各自の責任で行う必要があります。

*Timeout*ではDDEサーバー・アプリケーションからの応答をいつまで待つかをミリ秒単位で指定します。無効な数値（負数など）を指定した場合には、デフォルト値の1000が使用されます。

DDE.Requestの例

```
 /*
** Open a DDE Conversation with MS Excel for Windows on
** topic abc.xls then request data from 6 cells
** between row 2, column 2 and row 3, column 4
*/
```

```
DECLARE
  ConvID   PLS_INTEGER;
  Buffer   VARCHAR2(80);
BEGIN
   ConvID := DDE.Initiate('EXCEL', 'abc.xls');
   DDE.Request (ConvID, 'R2C2:R3C4', Buffer, DDE.Cf_Text,
     1000);
END;
```

DDE.Terminate

説明

アプリケーションとの指定した対話が終了します。

構文

```
PROCEDURE DDE.Terminate
    (ConvID PLS_INTEGER);
```

パラメータ

ConvID	対話識別子。

使用上の注意

DDE.Terminate呼出しの後、終了した対話識別子を使用していたDDE.Execute、DDE.Poke、DDE.RequestおよびDDE.Terminateを呼び出すとエラーが発生します。

DDE.Terminateを使用してサーバー・アプリケーションとの対話を終了するには、DDE.Initiateを使用して対話を起動しておく必要があります。

DDE.Terminateの例

```
 /*
** Open a DDE Conversation with MS Excel on topic
** abc.xls perform some operations, then terminate
** the conversation
*/
DECLARE
  ConvID      PLS_INTEGER;
BEGIN
  ConvID := DDE.Initiate('EXCEL', 'abc.xls');
  ...
  DDE.Terminate(ConvID);
END;
```

Debugパッケージ

DEBUGパッケージ

Debug.Break

Debug.Getx

Debug.Interpret

Debug.Setx

Debug.Suspend

Debug.Break

説明

デバッグ・トリガー内でブレークポイントを指定するのに使います。

構文

```
Debug.Break  EXCEPTION;
```

使用上の注意

Debug.Breakは、条件ブレークポイントを作成するのに役立ちます。例外が呼び出されると、デバッグ・トリガー位置にブレークポイントを入力した場合と同じく、制御がインタプリタに渡されます。

Debug.Breakの例

```
 /*
** Create a breakpoint only when the value
** of 'my_sal' exceeds 5000
*/
IF Debug.Getn('my_sal') > 5000 THEN
  RAISE Debug.Break;
END IF;
```

Debug.Getx

説明

指定したローカル変数の値が検索されます。

構文

```
FUNCTION Debug.Getc
   (varname VARCHAR2)
RETURN VARCHAR2;

FUNCTION Debug.Getd
   (varname VARCHAR2)
RETURN DATE;

FUNCTION Debug.Geti
   (varname VARCHAR2)
RETURN PLS_INTEGER;

FUNCTION Debug.Getn
   (varname VARCHAR2)
RETURN NUMBER;
```

パラメータ

varname	VARCHAR2またはCHAR(Debug.Getcにより、CHAR型はVARCHAR2型に変換されます)、DATE、PLS_INTEGERまたはNUMBER変数。

使用上の注意

デバッグ・トリガーからローカル値を決定する場合に役立ちます。

Debug.Getxの例

```
 /*
** Retrieve the value of the variable 'my_ename'
** and use it to test a condition
*/
IF Debug.Getc('my_ename') = 'JONES' THEN
  RAISE Debug.Break;
END IF;
```

サブプログラムbarをコールするプログラム単位fooがあるとします。このサブプログラム(bar)は、他の多くのプログラム単位からもコールされます。プロシージャbarが、それをコールする多くのプロシージャから引数'message'を受け付ける状況を考えてみます。プロシージャfooは、'hello world'という一意の引数をbarに渡します。この場合、fooがその引数を渡すときにのみブレークポイントを呼び出すトリガーを次のようにプロシージャbarに定義できます。

```
PL/SQL> .TRIGGER PROC bar LINE 3 IS
     >BEGIN
     >  IF Debug.Getn('message') = 'hello world' THEN
     >    RAISE Debug.Break;
     >  END IF;
     >END;
```

Debug.Interpret

説明

*input*で指定したPL/SQL文またはProcedure Builder Interpreterコマンドが、インタプリタに入力されたときと同じように実行します。

構文

```
PROCEDURE Debug.Interpret
   (input VARCHAR2);
```

パラメータ

input	**プロシージャ・ビルダーのコマンド文字列。**

使用上の注意

デバッグ・トリガーからProcedure Builderファンクションを自動的に起動するのに役立ちます。

Debug.Interpretの例

```
 /*
** Execute the command SHOW STACK when
** a condition is met
*/
IF Debug.Getc('my_ename') = 'JONES' THEN
  Debug.Interpret('.SHOW LOCALS');
END IF;
```

サブプログラムbarをコールするプログラム単位fooがあるとします。このサブプログラム(bar)は、他の多くのプログラム単位からもコールされます。ここではbarにブレークポイントを作成しますが、サブプログラムがfooからコールされたときにのみブレークポイントが使用可能になり、それ以外のプログラム単位からコールされたときは使用可能にならないようにします。

このためには、次のステップを実行する必要があります。

1 プロシージャbarで、実行を停止する位置にブレークポイントを作成します。

2 作成したブレークポイントを使用不可にします。

ステップ1と2は、どちらも「ブレークポイント」ダイアログ・ボックスの中で実行できます。プロシージャfooに、プロシージャbarに作成した最初のブレークポイントを使用可能にするブレークポイント・トリガー付きのブレークポイントを作成します。次に例を示します。

```
PL/SQL>.BREAK PROC foo LINE 6 TRIGGER
     >BEGIN
     >  Debug.Interpret('.enable break 1');
```

```
>  Debug.Interpret('.go');
>END;
```

次の例では、ブレークポイントに到達するたびにトリガーを起動するブレークポイントを作成します。

```
PL/SQL> .break proc my_proc line 10 trigger
     +> DEBUG.INTERPRET('.SHOW LOCALS');
```

Debug.Setx

説明

ローカル変数に新しい値を設定します。

構文

```
PROCEDURE Debug.Setc
   (varname  VARCHAR2,
    newvalue VARCHAR2);
PROCEDURE Debug.Setd
   (varname  VARCHAR2,
    newvalue DATE);
PROCEDURE Debug.Seti
   (varname  VARCHAR2,
    newvalue PLS_INTEGER);
PROCEDURE Debug.Setn
   (varname  VARCHAR2,
    newvalue NUMBER);
```

パラメータ

varname	VARCHAR2またはCHAR（Debug.Setcにより、CHAR型はVARCHAR2型に変換されます）、DATE、PLS_INTEGERまたはNUMBER変数。
newvalue	*varname*の適切な値。

使用上の注意

デバッグ・トリガーでローカル値を変更するのに役立ちます。

Debug.Setxの例

```
 /*
** Set the value of the local variable 'my_emp' from a
** Debug Trigger
*/
Debug.Setc('my_emp', 'SMITH');
```

```
/*
** Set the value of the local variable 'my_date' from a
** Debug Trigger
*/
Debug.Setd('my_date', '02-OCT-94');
```

Debug.Suspend

説明

現行のプログラム単位の実行が中断され、インタプリタに制御が渡されます。

構文

```
PROCEDURE Debug.Suspend;
```

Debug.Suspendの例

```
 /*
** This example uses Debug.Suspend
*/
PROCEDURE proc1 IS
BEGIN
  FOR i IN 1..10 LOOP
    Debug.Suspend;
    Text_IO.Put_Line('Hello');
  END LOOP;
 END;
```

EXEC_SQLパッケージ

EXEC_SQLパッケージ

次のファンクションとプロシージャは、通常のセッションでコールされる順に表示されています。

EXEC_SQL.Open_Connection

EXEC_SQL.Curr_Connection

EXEC_SQL.Default_Connection

EXEC_SQL.Open_Cursor

EXEC_SQL.Parse

EXEC_SQL.Describe_Column

EXEC_SQL.Bind_Variable

EXEC_SQL.Define_Column

EXEC_SQL.Execute

EXEC_SQL.Execute_And_Fetch

EXEC_SQL.Fetch_Rows

EXEC_SQL.More_Result_Sets

EXEC_SQL.Column_Value

EXEC_SQL.Variable_Value

EXEC_SQL.Is_Open

EXEC_SQL.Close_Cursor

EXEC_SQL.Is_Connected

EXEC_SQL.Is_OCA_Connection

EXEC_SQL.Close_Connection

次のファンクションでは、SQL文の実行後、接続時の最後に参照されたカーソルの情報が取り出されます。

EXEC_SQL.Last_Error_Position

EXEC_SQL.Last_Row_Count

EXEC_SQL.Last_SQL_Function_Code

EXEC_SQL.Last_Error_Code

EXEC_SQL.Last_Error_Mesg

EXEC_SQL.Open_Connection

```
FUNCTION EXEC_SQL.Open_Connection
    (Username     IN VARCHAR2,
     Password     IN VARCHAR2,
     Data source  IN VARCHAR2)
RETURN EXEC_SQL.ConnType;
```

パラメータ

Connstr	'ユーザー名[パスワード][@データベース文字列]'形式の文字列。
Username	データベースの接続に使用するユーザー名の文字列。
Password	ユーザー名に対するパスワードの文字列。
Data source	SQLNetの別名を指定する文字列または'ODBC:'で始まるOCA接続。

戻り値

新規データベース接続のハンドル。

EXEC_SQL.Open_Connectionの例

```
PROCEDURE getData IS
  --
  -- a connection handle must have a datatype of EXEC_SQL.conntype
  --
  connection_id EXEC_SQL.CONNTYPE;

  ...

BEGIN
  --
  -- a connection string is typically of the form
'username/password@database_alias'
  --
  connection_id := EXEC_SQL.OPEN_CONNECTION('connection_string');

  ...

END;
```

EXEC_SQL.Curr_Connection

説明

Developerによって確立された同じデータベース接続を使用する接続ハンドルが返されます。EXEC_SQL.Curr_Connectionは、EXEC_SQL.Default_Connectionに置き換えられます。

構文

```
FUNCTION EXEC_SQL.Curr_Connection
RETURN EXEC_SQL.ConnType;
```

戻り値

プライマリDeveloper接続のハンドル。

使用上の注意

EXEC_SQL.Curr_Connectionの代わりにEXEC_SQL.Default_Connectionを使用してください。ただし、下位互換性のためにEXEC_SQL.Curr_Connectionもサポートされています。

EXEC_SQL.Default_Connection

説明

Developerによって確立された同じデータベース接続を使用する接続ハンドルが返されます。EXEC_SQL.Curr_Connectionは、EXEC_SQL.Default_Connectionに置き換えられます。

構文

```
FUNCTION EXEC_SQL.Default_Connection
RETURN EXEC_SQL.ConnType;
```

戻り値

プライマリDeveloper接続のハンドル。

使用上の注意

デフォルト接続は、プライマリDeveloper接続です。デフォルト接続は、EXEC_SQL.Default_Connectionが初めてコールされたときに検索され、EXEC_SQLパッケージ内のキャッシュに格納されます。 そのキャッシュのハンドルがユーザーに返されます。次回以降のEXEC_SQL.Default_Connectionコールの際には、このハンドルが単純にキャッシュから取り出されます。

このように接続がキャッシュされるので、デフォルト接続のみでデータにアクセスしている場合は、他のEXEC_SQLメソッドコールの際に、明示的に接続ハンドルを指定する必要がありません。EXEC_SQLでは、自動的にキャッシュが検索され、接続ハンドルが取得されます。

キャッシュをクリアするには、EXEC_SQL.Default_Connectionコールで取得した接続ハンドル上で、EXEC_SQL.Close_Connectionをコールします。デフォルト接続では、EXEC_SQL.Close_Connectionによって接続が終了することはありません。 この場合、EXEC_SQLによって使用されているリソースが解放されるのみです。

EXEC_SQL.Default_Connectionおよび EXEC_SQL.Curr_Connectionの例

```
/*
** This example illustrates the use of
** EXEC_SQL.Default_Connection and
** EXEC_SQL.Curr_Connection.
*/
PROCEDURE esdefaultcon2 IS
  connection_id EXEC_SQL.CONNTYPE;
  bIsConnected BOOLEAN;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);
  nIgn PLS_INTEGER;
  nRows PLS_INTEGER := 0;
  nTimes PLS_INTEGER := 0;
  mynum NUMBER;

BEGIN
  --
  -- obtain the default connection and check that it is valid
  --
  connection_id := EXEC_SQL.DEFAULT_CONNECTION;
  bIsConnected := EXEC_SQL.IS_CONNECTED;
  IF bIsConnected = FALSE THEN
    TEXT_IO.PUT_LINE('No primary connection.Please connect before
retrying.');
    RETURN;
  END IF;
  --
  -- subsequent calls to EXEC_SQL.Open_Cursor, EXEC_SQL.Parse,
       EXEC_SQL.Define_Column,
  -- EXEC_SQL.Execute, EXEC_SQL.Fetch_Rows, EXEC_SQL.Column_Value,
  -- EXEC_SQL.Close_Cursor, EXEC_SQL.Close_Connection all use this
connection
  -- implicitly from the cache
  --
  cursorID := EXEC_SQL.OPEN_CURSOR;
  sqlstr := 'select empno from emp';
  EXEC_SQL.PARSE(cursorID, sqlstr, exec_sql.V7);
  EXEC_SQL.DEFINE_COLUMN(cursorID, 1, mynum);
  nIgn := EXEC_SQL.EXECUTE(cursorID);
```

```
  LOOP
    IF (EXEC_SQL.FETCH_ROWS(cursorID) > 0) THEN
      EXEC_SQL.COLUMN_VALUE(cursorID, 1, mynum);

      ...

    ELSE
      exit;
    END IF;
  END LOOP;
  EXEC_SQL.CLOSE_CURSOR(cursorID);
  EXEC_SQL.CLOSE_CONNECTION;
END;
```

EXEC_SQL.Open_Cursor

説明

指定した接続上で新規のカーソルが作成され、カーソル・ハンドルが返されます。カーソルが不要になった場合は、EXEC_SQL.Close_Cursorを使用して、カーソルを明示的にクローズする必要があります。

構文

```
FUNCTION EXEC_SQL.Open_Cursor
    [Connid     IN CONNTYPE]
RETURN EXEC_SQL.CursType;
```

パラメータ

Connid	**使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。**

戻り値

新規カーソルのハンドル。

使用上の注意

カーソルは、同じSQL文を（再解析なしで）繰り返し実行したり、新規のSQL文を（解析付きで）実行する場合に使用できます。新規の文に対してカーソルを再使用すると、新規の文が解析されるときに、そのカーソルの内容が自動的にリセットされます。つまり、カーソルは再使用する前に、クローズしたり再オープンしたりする必要がありません。

EXEC_SQL.Open_Cursorの例

```
PROCEDURE getData IS
  --
  -- a cursorID must be of type EXEC_SQL.cursType
  --
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;

  ...

BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION('connect_str');

  ...

  --
  -- this cursor is now associated with a particular connection
  --
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);

  ...

END;
```

EXEC_SQL.Parse

説明

このプロシージャでは、指定したカーソルの文を解析します。

構文

```
PROCEDURE EXEC_SQL.Parse
   ([Connid     IN CONNTYPE,]
     Curs_Id    IN CURSTYPE,
     Statement  IN VARCHAR2
     [Language  IN PLS_INTEGER]);
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	文に割り当てるカーソル・ハンドルです。
Statement	解析するSQL文です。最後にセミコロンを含めることはできません。

Language_flag	OracleがSQL文を処理する方法を決定するフラグです。有効なフラグは次のとおりです。 **V6** Oracle V6の動作を指定する。 **V7** Oracle V7の動作を指定する。 **NATIVE** デフォルト。

使用上の注意

SQL文はすべて解析プロシージャを使って解析する必要があります。解析することにより、文の構文がチェックされ、構文とカーソルがコード内で関連付けられます。OCI解析とは異なり、EXEC_SQL解析は即座に実行されます。EXEC_SQL解析を後回しにすることはできません。

データ操作言語(DML)やデータ定義言語(DDL)の文を解析することができます。Oracleデータ・ソースでは、DDL文は解析中に実行されます。非Oracleデータ・ソースでは、DDL文は、解析中または実行中に実行される可能性があります。これは、すべてのDDL文が常にEXEC_SQLで解析、実行される必要があることを意味します。

EXEC_SQL.Parseの例

```
PROCEDURE getData IS
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);

  ...

BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION(connect_str);
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  --
  -- the statement to be parsed is stored as a VARCHAR2 variable
  --
  sqlstr := 'select ename from emp';
  --
  -- perform parsing
  --
  EXEC_SQL.PARSE(connection_id, cursorID, sqlstr, exec_sql.V7);

  ...

END;
```

EXEC_SQL.Describe_Column

説明

解析したSQL文の結果セット列に関する情報を取得します。結果セットにない列番号を記述しようとすると、EXEC_SQL.Invalid_Column_Number例外が発生します。ヒント

構文

```
PROCEDURE EXEC_SQL.Describe_Column
   ([Connid     IN CONNTYPE,
     Curs_Id    IN CURSTYPE,
     Position   IN PLS_INTEGER,
     Name      OUT VARCHAR2,
     Collen     OUT PLS_INTEGER,
     Type      OUT PLS_INTEGER]);
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	説明する列に関連付けられたカーソル・ハンドルです。
Position	説明する列の結果セットの位置です。位置は、左から右の順に1から番号が付けられます。
Name	出力時に列の名前を含みます。
Collen	出力時にバイト単位の列の最大長を含みます。
Type	出力時に列タイプを含みます。有効な値は次のとおりです。 EXEC_SQL.VARCHAR2_TYPE EXEC_SQL.NUMBER_TYPE EXEC_SQL.FLOAT_TYPE EXEC_SQL.LONG_TYPE EXEC_SQL.ROWID_TYPE EXEC_SQL.DATE_TYPE EXEC_SQL.RAW_TYPE EXEC_SQL.LONG_RAW_TYPE EXEC_SQL.CHAR_TYPE (ANSI固定CHAR) EXEC_SQL.MLSLABLE_TYPE (Trusted Oracleのみ)

EXEC_SQL.Describe_Columnの例

```
PROCEDURE esdesccol(tablename VARCHAR2) IS
  connection_id EXEC_SQL.CONNTYPE;
  cursor_number EXEC_SQL.CURSTYPE;
  sql_str VARCHAR2(256);
  nIgnore PLS_INTEGER;
  nColumns PLS_INTEGER := 0;       --count of number of columns returned
  colName VARCHAR2(30);
```

```
  colLen PLS_INTEGER;
  colType PLS_INTEGER;

BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION('connection_string');
  --
  -- when you do a "select *..." from a table which is known only at runtime,
  -- you cannot know what the columns are a priori.EXEC_SQL.Describe_Column
becomes
  -- very usefule then
  --
  sql_str := 'select * from ' || tablename;
  cursor_number := EXEC_SQL.OPEN_CURSOR(connection_id);
  EXEC_SQL.PARSE(connection_id, cursor_number, sql_str, exec_sql.V7);
  nIgnore := EXEC_SQL.EXECUTE(connection_id, cursor_number);

  LOOP
    nColumns := nColumns + 1;  --used as column index into result set
    --
    -- describe_column is in general used within a PL/SQL block with an
exception
    -- block included to catch the EXEC_SQL.invalid_column_number exception.
    -- when no more columns are found, we can store the returned column
names
    -- and column lengths in a PL/SQL table of records and do further queries
    -- to obtain rows from the table.In this example, colName, colLen and
colType
    -- are used to store the returned column characteristics.
    --
  BEGIN
    EXEC_SQL.DESCRIBE_COLUMN(connection_id, cursor_number,
                  nColumns, colName, colLen, colType);
        TEXT_IO.PUT_LINE(' col= ' || nColumns || ' name ' || colName  ||
                     ' len= ' || colLen   || ' type ' || colType );
  EXCEPTION
    WHEN EXEC_SQL.INVALID_COLUMN_NUMBER THEN
      EXIT;
    END;
  END LOOP;

  nColumns := nColumns - 1;
  IF (nColumns <= 0) THEN
    TEXT_IO.PUT_LINE('No columns returned in query');
  END IF;

  ...

  EXEC_SQL.CLOSE_CURSOR(connection_id, cursor_number);
  EXEC_SQL.CLOSE_CONNECTION(connection_id);
END;
```

EXEC_SQL.Bind_Variable

説明

所定の値をSQL文の名前付き変数にバインドします。

構文

```
PROCEDURE EXEC_SQL.Bind_Variable
   ([Connid     IN CONNTYPE],
     Curs_Id    IN CURSTYPE,
     Name       IN VARCHAR2,
     Value      IN <datatype>);
```

<datatype>には、次の値のうちのいずれかが使用できます。

```
     NUMBER
     DATE
     VARCHAR2
PROCEDURE EXEC_SQL.Bind_Variable
   ([Connid          IN CONNTYPE],
     Curs_Id         IN CURSTYPE,
     Name            IN VARCHAR2,
     Value           IN VARCHAR2,
     Out_Value_Size  IN PLS_INTEGER);
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	変数のバインドを行うカーソル・ハンドルです。
Name	SQL文内の変数の名前です。
Value	INおよびIN/OUT変数の場合、値は名前付き変数にバインドするデータです。OUT変数の場合、実際にはデータは無視されますが、後でVariable_Valueによって取り出されるPL/SQL変数のタイプを示すため、Bind_Variableを使用する必要があります。
Out_Value_Size	VARCHAR2 OUTまたはIN/OUT変数で予想されるOUTの値のバイト単位の最大サイズ。サイズが指定されていない場合は、値パラメータの現行の長さが使用されます。

使用上の注意

SQL文のプレースホルダを使って、実行時に入力データが提供される場所をマークします。文がPL/SQLブロックまたは出力パラメータ付きのストアド・プロシージャの場合、出力値のプレースホルダも使用する必要があります。

各入力プレースホルダには、値を提供するためのEXEC_SQL.Bind_Variableを使用する必要があります。各出力プレースホルダにもEXEC_SQL.Bind_Variableを使用する必要があります。これは、次回以降のEXEC_SQL.Bind_Variableコールで、値の引出しに使用される変数タイプを指定するためです。

SQL文の入力プレースホルダまたはバインド変数は、コロンで始まる名前で識別されます。たとえば、「:X」という文字列は、次のSQL文のバインド変数になっています。

```
SELECT ename FROM emp WHERE SAL > :X;
```

対応するEXEC_SQL.Bind_Variableプロシージャは次のとおりです。

```
BIND_VARIABLE(connection_handle, cursor_handle, ':X', 3500);
```

EXEC_SQL.Bind_Variableの例

```
PROCEDURE getData(input_empno NUMBER) IS
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);

  ...

BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION(connect_str);
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  --
  -- the statement to be parsed contains a bind variable
  --
  sqlstr := 'select ename from emp where empno = :bn';
  --
  -- perform parsing
  --
  EXEC_SQL.PARSE(connection_id, cursorID, sqlstr, exec_sql.V7);
  --
  -- the bind_variable procedure assigns the value of the input argument
to the named
  -- bind variable.Note the use of the semi-colon and the quotes to designate
the
  -- bind variable.The bind_variable procedure is called after the parse
procedure.
  --
  EXEC_SQL.BIND_VARIABLE(connection_id, cursorID, ':bn', input_empno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 1, mynum);

  ...

END;
```

EXEC_SQL.Define_Column

説明

このプロシージャは、SELECT文、または結果セットを返す非Oracleのストアド・プロシージャのコールにのみ使用されます。指定したカーソルからフェッチされる列を定義します。列は結果セットの相対位置によって定義されます。 最初の相対位置は、整数1で識別されます。 定義される列タイプは、列パラメータのPL/SQLタイプによって決まります。

構文

```
PROCEDURE EXEC_SQL.Define_Column
   ([Connid     IN CONNTYPE],
     Curs_Id    IN CURSTYPE,
     Position   IN PLS_INTEGER,
     Column     IN <datatype>);
```

<datatype>には、次の値のうちのいずれかが使用できます。

```
     NUMBER
     DATE
     VARCHAR2
```

```
PROCEDURE EXEC_SQL.Define_Column
   ([Connid       IN CONNTYPE],
     Curs_Id      IN CURSTYPE,
     Position     IN PLS_INTEGER,
     Column       IN VARCHAR2,
     Column_Size  IN PLS_INTEGER);
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	定義する列のあるカーソル・ハンドルです。
Position	行または結果セットの列の相対位置です。文の最初の列の相対位置は1です。
Column	定義される列の値です。値のタイプによって定義される列のタイプが決まります。変数に格納されている実際の値は無視されます。
Column_Size	列値の予想されるバイト単位の最大サイズ（列タイプがVARCHAR2の場合のみ）。

使用上の注意

問合せを行う場合は、EXEC_SQL.Column_Valueでデータを取り出す前に、列を定義する必要があります。

EXEC_SQL.Define_Columnの例

```
PROCEDURE getData IS
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);
  loc_ename VARCHAR2(30);   -- these are variables local to the procedure;
  loc_eno NUMBER;          --used to store the return values from our desired
  loc_hiredate DATE;        -- query

 ...

BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION(connect_str);
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  sqlstr := 'select ename, empno, hiredate from emp ';
  EXEC_SQL.PARSE(connection_id, cursorID, sqlstr, exec_sql.V7);
  EXEC_SQL.BIND_VARIABLE(connection_id, cursorID, ':bn', input_empno);
  --
  -- we make one call to DEFINE_COLUMN per item in the select list.We must
use local
  -- variables to store the returned values.For a result value that is a
VARCHAR,
  -- it  is important to specify the maximumn length.For a result value
that is a
  -- number or a date, there is no need to specify the maximum length.We
obtain the
  -- relative positions of the columns being returned from the select
statement,
  -- sql_str.
  --
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 1, loc_ename, 30);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 2, loc_eno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 3, loc_hiredate);

 ...

END;
```

EXEC_SQL.Execute

説明

指定したカーソルでSQL文を実行します。

構文

```
FUNCTION EXEC_SQL.Execute
    ([Connid     IN CONNTYPE],
```

```
     Curs_Id    IN CURSTYPE)
RETURN PLS_INTEGER;
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	実行するSQL文のカーソル・ハンドルです。

戻り値

処理した行数。

使用上の注意

戻り値は、INSERT、UPDATE、DELETET文でのみ有効です。DDLなどの他の文では、戻り値が定義されていないために無視されます。

EXEC_SQL.Executeの例

```
PROCEDURE getData IS
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);
  loc_ename VARCHAR2(30);
  loc_eno NUMBER;
  loc_hiredate DATE;
  nIgn PLS_INTEGER;

 ...

BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION(connect_str);
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  sqlstr := 'select ename, empno, hiredate from emp ';
  EXEC_SQL.PARSE(connection_id, cursorID, sqlstr, exec_sql.V7);
  EXEC_SQL.BIND_VARIABLE(connection_id, cursorID, ':bn', input_empno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 1, loc_ename, 30);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 2, loc_eno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 3, loc_hiredate);
  --
  -- after parsing, and calling BIND_VARIABLE and DEFINE_COLUMN, if
necessary, you
  -- are ready to execute the statement.Note that all information about
the
  -- statement and its result set is encapsulated in the cursor referenced
as cursorID.
  --
  nIgn := EXEC_SQL.EXECUTE(connection_id, cursorID);
```

```
 ...

END;
```

EXEC_SQL.Execute_And_Fetch

説明

このファンクションは、EXEC_SQL.ExecuteとEXEC_SQL.Fetch_Rowsを順番にコールします。指定したカーソルでSQL文が実行され、問合せを満たす最初の行が取り出されます。EXEC_SQL.Execute_And_Fetchをリモート・データベースに対してコールすると、ラウンドトリップ回数が減る可能性があります。

構文

```
FUNCTION EXEC_SQL.Execute_And_Fetch
    ([Connid     IN CONNTYPE],
      Curs_Id    IN CURSTYPE,
      Exact      IN BOOLEAN DEFAULT FALSE)
RETURN PLS_INTEGER;
```

パラメータ

Connid	**使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。**
Curs_Id	**実行するSQL文のカーソル・ハンドルです。**
Exact	**デフォルトではFALSEです。EXEC_SQL.Package_Errorの例外を発生させる場合は、値をTRUEに設定します。例外が発生しても、行は取り出されます。**

戻り値

フェッチされた行数（0または1のいずれか）。

EXEC_SQL.Execute_And_Fetchの例

```
PROCEDURE getData(input_empno NUMBER) IS
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);
  loc_ename VARCHAR2(30);
  loc_eno NUMBER;
  loc_hiredate DATE;
  nIgn PLS_INTEGER;

 ...
```

```
BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION(connect_str);
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  --
  -- assuming that empno is a primary key of the table emp, the where clause
guarantees
  -- that 0 or 1 row is returned
  --
  sqlstr := 'select ename, empno, hiredate from emp '
  sqlstr := sqlstr || ' where empno = ' || input_empno;
  EXEC_SQL.PARSE(connection_id, cursorID, sqlstr, exec_sql.V7);
  EXEC_SQL.BIND_VARIABLE(connection_id, cursorID, ':bn', input_empno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 1, loc_ename, 30);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 2, loc_eno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 3, loc_hiredate);
  --
  -- do execute_and_fetch after parsing the statement, and calling
bind_variable
  -- and define_column if necessary
  --
  nIgn := EXEC_SQL.EXECUTE_AND_FETCH (connection_id, cursorID);
  IF (nIgn = 0 ) THEN
    TEXT_IO.PUT_LINE (' No employee has empno = ' || input_empno);
  ELSE IF (nIgn = 1) THEN
    TEXT_IO.PUT_LINE (' Found one employee with empno ' || input_empno);
  --
  -- obtain the values in this row
  --
    EXEC_SQL.column_value(connection_id, cursorID, 1, loc_ename);
    EXEC_SQL.column_value(connection_id, cursorID, 2, loc_eno);
    EXEC_SQL.column_value(connection_id, cursorID, 3, loc_hiredate);


 ...

  END IF;


 ...

END;
```

EXEC_SQL.Fetch_Rows

説明

指定したカーソルで問合せを満たす行を取り出します。

構文

```
FUNCTION EXEC_SQL.Fetch_Rows
    ([Connid     IN CONNTYPE],
      Curs_Id    IN CURSTYPE)
RETURN PLS_INTEGER;
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	フェッチの対象となるSQL文のカーソル・ハンドルです。

戻り値

実際にフェッチされた行数。

使用上の注意

各EXEC_SQL.Fetch_Rowsコールによって、1つの行がバッファに取り出されます。0が返されるまで、繰り返しEXEC_SQL.Fetch_Rowsを使用してください。Oracleデータベースでは、結果セットにデータがないことを意味します。非Oracleデータ・ソースでは、指定したカーソルにデータがないことを意味しません。詳細は、EXEC_SQL.More_Results_Setsを参照してください。

各EXEC_SQL.Fetch_Rowsコールを行った後に、EXEC_SQL.Column_Valueを使って、フェッチされた行の各列を読み込んでください。

EXEC_SQL.Fetch_Rowsの例

```
PROCEDURE getData IS
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);
  loc_ename VARCHAR2(30);
  loc_eno NUMBER;
  loc_hiredate DATE;
  nIgn PLS_INTEGER;
  nRows PLS_INTEGER := 0;  -- used for counting the actual number of rows
returned

 ...

BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION(connect_str);
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  sqlstr := 'select ename, empno, hiredate from emp ';
  EXEC_SQL.PARSE(connection_id, cursorID, sqlstr, exec_sql.V7);
  EXEC_SQL.BIND_VARIABLE(connection_id, cursorID, ':bn', input_empno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 1, loc_ename, 30);
```

```
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 2, loc_eno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 3, loc_hiredate);
  nIgn := EXEC_SQL.EXECUTE(connection_id, cursorID);
  --
  -- call FETCH_ROWS to obtain a row.When a row is returned, obtain the
values,
  -- and increment the count.
  --
  WHILE (EXEC_SQL.FETCH_ROWS(connection_id, cursorID) > 0 ) LOOP
    nRows := nRows + 1;
    EXEC_SQL.COLUMN_VALUE(connection_id, cursorID, 1, loc_ename;
    EXEC_SQL.COLUMN_VALUE(connection_id, cursorID, 2, loc_eno);
    EXEC_SQL.COLUMN_VALUE(connection_id, cursorID, 3, loc_hiredate);

 ...

  END LOOP;
  --
  -- The loop terminates when FETCH_ROWS returns 0. This could have happen
because
  -- the query was incorrect or because there were no more rows.To distinguish
  -- between these cases, we keep track of the number of rows returned.
  --
  IF (nRows <= 0) THEN
    TEXT_IO.PUT_LINE ('Warning:query returned no rows');
  END IF;

 ...

END;
```

EXEC_SQL.More_Result_Sets

説明

このファンクションは、非オラクル接続にのみ適用されます。指定したカーソルに、他に取り出すべき結果セットがないかどうかを確認します。

構文

```
FUNCTION EXEC_SQL.More_Result_Sets
   ([Connid    IN CONNTYPE],
     Curs_Id   IN CURSTYPE)
RETURN BOOLEAN;
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	フェッチの対象となるSQL文のカーソル・ハンドルです。

戻り値

TRUEまたはFALSE。

使用上の注意

Oracleデータベースに対して使用すると、このファンクションは常にFALSEを返します。非Oracleストアド・プロシージャに、取り出すべき別の結果セットがあると、ファンクションは結果セットを初期化し、TRUEを返します。EXEC_SQL.Describe_Columnを使って新しい結果セットの情報を取得し、必要に応じてEXEC_SQL.Fetch_Rowsを使ってデータを取り出します。

EXEC_SQL.More_Result_Setsの例

```
PROCEDURE esmoreresultsets(sqlstr VARCHAR2) IS
  conidODBC EXEC_SQL.CONNTYPE;
  nRes PLS_INTEGER;
  nRows PLS_INTEGER := 0 ;
  curID EXEC_SQL.CURSTYPE;
BEGIN
  --
  -- an ODBC connection string; usually has the form
'username/password@ODBD:dbname'
  --
  conidODBC := EXEC_SQL.OPEN_CONNECTION('connection_str_ODBC');
  curID := EXEC_SQL.OPEN_CURSOR(conidODBC);
  EXEC_SQL.PARSE(conidODBC, curID, sqlstr, exec_sql.v7);
  nRes := EXEC_SQL.EXECUTE(conidODBC, curID);
  --
  -- obtain results from first query in sqlstr
  WHILE (EXEC_SQL.FETCH_ROWS(conidODBC, curID) > 0) LOOP
    nRows := nRows + 1;

 ...

  END LOOP;
  --
  -- for some non-Oracle databases, sqlstr may contain a batch of queries;
  -- MORE_RESULT_SETS checks for additional result sets
  --
  IF (EXEC_SQL.MORE_RESULT_SETS(conidODBC, curID)) THEN
    TEXT_IO.PUT_LINE(' more result sets ');
```

```
  ELSE
    TEXT_IO.PUT_LINE(' no more result sets ');
  END IF;

 ...

  EXEC_SQL.CLOSE_CONNECTION(conidODBC);
END;
```

EXEC_SQL.Column_Value

説明

このプロシージャは、所定のカーソル位置にカーソルの値を返します。これは、EXEC_SQL.Fetch_Rowsコールによってフェッチされたデータにアクセスするために使用します。

構文

```
PROCEDURE EXEC_SQL.Column_Value
    ([Connid       IN CONNTYPE],
      Curs_Id      IN CURSTYPE,
      Position     IN PLS_INTEGER,
      Value       OUT <datatype>,
     [Column_Error  OUT NUMBER],
     [Actual_Length OUT PLS_INTEGER]);
```

<datatype>には、次の値のうちのいずれかが使用できます。

```
NUMBER
DATE
VARCHAR2
```

パラメータ

名前	モード	説明
Connid	IN	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	IN	列値を取得する行のカーソル・ハンドルです。
Position	IN	指定したカーソルの列の相対位置です。左から始まり、最初の列の位置は1になります。
Value	OUT	指定した列と行の値を返します。
Column_Error	OUT	指定した列値についてエラー・コードを返します（Oracleデータ・ソースのみ）。
Actual_Length	OUT	切捨ての前に実際の列値の長さを返します。

使用上の注意

EXEC_SQL.Column_Valueを使って値を取り出す前に、EXEC_SQL.Define_Columnを使って列データ文字を定義する必要があります。EXEC_SQL.Define_Columnで指定した値とは異なるPL/SQLタイプの値を指定すると、EXEC_SQL.Value_Errorの例外が発生します。

EXEC_SQL.Column_Valueの例

```
PROCEDURE getData IS
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);
  loc_ename VARCHAR2(30);
  loc_eno NUMBER;
  loc_hiredate DATE;
  nIgn PLS_INTEGER;
  nRows PLS_INTEGER := 0;

 ...

BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION(connect_str);
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  sqlstr := 'select ename, empno, hiredate from emp ';
  EXEC_SQL.PARSE(connection_id, cursorID, sqlstr, exec_sql.V7);
  EXEC_SQL.BIND_VARIABLE(connection_id, cursorID, ':bn', input_empno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 1, loc_ename, 30);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 2, loc_eno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 3, loc_hiredate);
  nIgn := EXEC_SQL.EXECUTE(connection_id, cursorID);
  --
  -- You must have used DEFINE_COLUMN to define the column data
  -- characteristics before using COLUMN_VALUE to retrieve the value.
  -- Assign the row's first value to the local variable loc_ename.
  -- Assign the row's second value to the local variable loc_eno.
  -- Assign the row's third value to the local variable loc_hiredate.
  --
  WHILE (EXEC_SQL.FETCH_ROWS(connection_id, cursorID) > 0 ) LOOP
    nRows := nRows + 1;
    EXEC_SQL.COLUMN_VALUE(connection_id, cursorID, 1, loc_ename);
    EXEC_SQL.COLUMN_VALUE(connection_id, cursorID, 2, loc_eno);
    EXEC_SQL.COLUMN_VALUE(connection_id, cursorID, 3, loc_hiredate);

   ...
```

```
  END LOOP;
  IF (nRows <= 0) THEN
    TEXT_IO.PUT_LINE ('Warning: query returned no rows');
  END IF;

 ...

END;
```

EXEC_SQL.Variable_Value

説明

このプロシージャは、指定したカーソルで名前付きバインド変数の出力値を取り出します。無名PL/SQLブロックのバインド変数の値も返されます。

構文

```
PROCEDURE EXEC_SQL.Variable_Value
    ([Connid     IN CONNTYPE],
      Curs_Id    IN CURSTYPE,
      Name       IN VARCHAR2,
      Value      OUT <datatype>);
```

<datatype>には、次の値のうちのいずれかが使用できます。

```
NUMBER
DATE
VARCHAR2
```

パラメータ

名前	モード	説明
Connid	IN	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	IN	バインド変数を取り出すカーソル・ハンドルです。
Name	IN	バインド変数の名前です。
Value	OUT	指定したカーソルのバインド変数の値を返します。

使用上の注意

EXEC_SQL.Bind_Variableを使ってバインド変数に指定したデータ型以外のものを取り出そうとすると、EXEC_SQL.Value_Error例外が発生します。

EXEC_SQL.Variable_Valueの例

It is assumed that the following procedure, tstbindnum, exists on the server which is specified by the connection string used in OPEN_CONNECTION.

Create or replace procedure tstbindnum (input IN NUMBER, output OUT NUMBER) as

```
BEGIN
  output := input * 2;
END;
```

All this procedure does is to take an input number, double its value, and return it in the out variable.

```
PROCEDURE esvarvalnum (input IN NUMBER) IS
  connection_id EXEC_SQL.CONNTYPE;
  bIsConnected BOOLEAN;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);
  nRes PLS_INTEGER;
  mynum NUMBER;
BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION('connection_string');
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  sqlstr := 'begin  tstbindnum(:bn1, :bnret);  end;';  -- an anonymous
block
  EXEC_SQL.PARSE(connection_id, cursorID, sqlstr, exec_sql.V7);
  --
  -- define input value
  --
  EXEC_SQL.BIND_VARIABLE(connection_id, cursorID, ':bn1', input);
  --
  -- set up output value
  --
  EXEC_SQL.BIND_VARIABLE(connection_id, cursorID, ':bnret', mynum);
  nRes := EXEC_SQL.EXECUTE(connection_id, cursorID);
  --
  -- after the statement is executed, we call VARIABLE_VALUE to obtain the
value
  -- of the bind variable :bnret
  --
  EXEC_SQL.VARIABLE_VALUE(connection_id, cursorID, ':bnret', mynum);
  EXEC_SQL.CLOSE_CURSOR(connection_id, cursorID);
  EXEC_SQL.CLOSE_CONNECTION(connection_id);
END;
```

EXEC_SQL.Is_Open

説明

指定したカーソルが指定した接続上でオープンしている場合にTRUEを返します。

構文

```
FUNCTION EXEC_SQL.Is_Open
   ([Connid     IN CONNTYPE],
     Curs_Id    IN CURSTYPE)
RETURN BOOLEAN;
```

パラメータ

Connid	**使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。**
Curs_Id	**オープンしているかどうかを確認するカーソル・ハンドルです。**

戻り値

TRUEまたはFALSE。

EXEC_SQL.Is_Open、EXEC_SQL.Close_Cursor、EXEC_SQL.Is_Connectedおよび EXEC_SQL.Close_Connectionの例

```
/*
** This example illustrates the use of EXEC_SQL.Is_Open,
** EXEC_SQL.Close_Cursor, EXEC_SQL.Is_Connected and
** EXEC_SQL.Close_Connection.
*/
PROCEDURE esclosecursor.pld IS
  connection_id EXEC_SQL.CONNTYPE;
  bIsConnected BOOLEAN;
  cr1 EXEC_SQL.CURSTYPE;
  sqlstr1 VARCHAR2(200);
  sqlstr2 VARCHAR2(200);
  nRes PLS_INTEGER;
  bOpen BOOLEAN;
  nRows PLS_INTEGER;
  loc_ename VARCHAR2(30);
  loc_eno NUMBER;
  loc_hiredate DATE;
BEGIN
  BEGIN
    connection_id := EXEC_SQL.OPEN_CONNECTION('connection_str');
  EXCEPTION
    WHEN EXEC_SQL.PACKAGE_ERROR THEN
    TEXT_IO.PUT_LINE(' connection open failed ');
```

```
  END;
  --
  -- confirm that connection is valid
  --
  bIsConnected := EXEC_SQL.IS_CONNECTED(connection_id);
  IF bIsConnected = FALSE THEN
    TEXT_IO.PUT_LINE('No present connection to any data source.Please
connect before retrying.');
    RETURN;
  END IF;
  --
  -- open a cursor and do an update
  --
  cr1 := EXEC_SQL.OPEN_CURSOR(connection_id);
  sqlstr1 := 'update emp set empno = 3600 where empno = 7839';
  EXEC_SQL.PARSE(connection_id, cr1, sqlstr1, exec_sql.V7);

  nRes := EXEC_SQL.EXECUTE(connection_id, cr1);
  --
  -- reuse the same cursor, if open, to do another query.
  --
  sqlstr2 := 'select ename, empno, hiredate from emp ';
  --
  -- use IS_OPEN to check the state of the cursor
  --
  IS (EXEC_SQL.IS_OPEN(connection_id, cr1) != TRUE) THEN
    TEXT_IO.PUT_LINE('Cursor no longer available ');
    RETURN;
  END IF;
  --
  -- associate the cursor with another statement, and proceed to do the
query.
  --
  EXEC_SQL.PARSE(connection_id, cr1, sqlstr2, exec_sql.V7);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cr1, 1, loc_ename, 30);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cr1, 2, loc_eno);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cr1, 3, loc_hiredate);
  nIgn := EXEC_SQL.EXECUTE(connection_id, cr1);
  WHILE (EXEC_SQL.FETCH_ROWS(connection_id, cursorID) > 0 ) LOOP
    nRows := nRows + 1;
    EXEC_SQL.COLUMN_VALUE(connection_id, cr1, 1, loc_ename);
    EXEC_SQL.COLUMN_VALUE(connection_id, cr1, 2, loc_eno);
    EXEC_SQL.COLUMN_VALUE(connection_id, cr1, 3, loc_hiredate);

 ...

  END LOOP;
  --
  -- close the cursor and connection to free up resources
  --
```

```
  EXEC_SQL.CLOSE_CURSOR(connection_id, cr1);
  EXEC_SQL.CLOSE_CONNECTION(connection_id);
END;
```

EXEC_SQL.Close_Cursor

説明

指定したカーソルをクローズし、割り当てられていたメモリーを解放します。

構文

```
PROCEDURE EXEC_SQL.Close_Cursor
   ([Connid     IN CONNTYPE],
     Curs_Id    IN OUT CURSTYPE);
```

パラメータ

パラメータ	モード	説明
Connid	IN	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
Curs_Id	IN	クローズするカーソル・ハンドルです。
	OUT	NULLに設定します。

使用上の注意

カーソルが不要になった場合は、そのカーソルをクローズします。不要なカーソルをクローズしないと、新規カーソルをオープンすることができません。

EXEC_SQL.Is_Connected

説明

指定した接続ハンドルがデータ・ソースに接続されていると、TRUEが返されます。

構文

```
FUNCTION EXEC_SQL.Is_Connected
   [Connid     IN CONNTYPE]
RETURN BOOLEAN;
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。

戻り値

TRUEまたはFALSE。

EXEC_SQL.Is_OCA_Connection

説明

指定した接続ハンドルがOCA接続用のものである場合に、TRUEが返されます。

構文

```
FUNCTION EXEC_SQL.Is_OCA_Connection
    ([Connid     IN CONNTYPE]
RETURN BOOLEAN;
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。

戻り値

TRUEまたはFALSE。

EXEC_SQL.Is_OCA_Connectionの例

```
PROCEDURE esmoreresultsets(sqlstr VARCHAR2) IS
  conidODBC EXEC_SQL.CONNTYPE;
  nRes PLS_INTEGER;
  nRows PLS_INTEGER := 0 ;
  curID EXEC_SQL.CURSTYPE;
BEGIN
  --
  -- an ODBC connection string
  --
  conidODBC := EXEC_SQL.OPEN_CONNECTION('connection_str_ODBC');
  curID := EXEC_SQL.OPEN_CURSOR(conidODBC);
  EXEC_SQL.PARSE(conidODBC, curID, sqlstr, exec_sql.v7);
  nRes := EXEC_SQL.EXECUTE(conidODBC, curID);
  --
  -- obtain results from first query in sqlstr
  --
  WHILE (EXEC_SQL.FETCH_ROWS(conidODBC, curID) > 0) LOOP
    nRows := nRows + 1;

 ...
```

```
  END LOOP;
  --
  -- check whether this is an OCA connection.Does not continue for an Oracle
  --  connection.
  --
  IF (EXEC_SQL.IS_OCA_CONNECTION != TRUE) THEN
    TEXT_IO.PUT_LINE('Not an OCA connection ');
    RETURN;
  END IF;
  --
  -- check for more result sets
  --
  IF (EXEC_SQL.MORE_RESULT_SETS(conidODBC, curID)) THEN
    TEXT_IO.PUT_LINE(' more result sets ');
  ELSE
    TEXT_IO.PUT_LINE(' no more result sets ');
  END IF;

 ...

  EXEC_SQL.CLOSE_CONNECTION(conidODBC);
END;
```

EXEC_SQL.Close_Connection

説明

このプロシージャは、接続ハンドルによって使用されていたリソースをすべて解放し、無効にします。

構文

```
PROCEDURE EXEC_SQL.Close_Connection
   ([Connid     IN OUT CONNTYPE]);
```

パラメータ

名前	モード	説明
Connid	IN	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。
	OUT	ハンドルをNULLに設定します。ハンドルに割り当てられていたメモリーもすべて解放されます。

使用上の注意

EXEC_SQL.Open_Connectionを使って接続をオープンした場合に、EXEC_SQL.Close_Connectionを使用すると、データベース接続もクローズされます。EXEC_SQL.Default_Connectionを使ってオープンした場合は、EXEC_SQL.Close_Connectionによってデータベース接続がクローズされることはありません。

不要になった接続はクローズする必要があります。接続をクローズしないと、データベース接続はオープン状態のままになり、オープンしているカーソルなど、接続に割り当てられたメモリーが使用された状態のままになります。これにより、接続デッドロックになる場合があります。

EXEC_SQL.Last_Error_Position

説明

エラーが発生したSQL文のバイト・オフセットを返します。文の最初の文字は、0の位置に配置されます。

構文

```
FUNCTION EXEC_SQL.Last_Error_Position
    [Connid     IN CONNTYPE]
RETURN PLS_INTEGER;
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。

戻り値

整数。

使用上の注意

このファンクションは、EXEC_SQL.PARSEの後、他のEXEC_SQLプロシージャまたはファンクションを使用する前に使用してください。OCAデータ・ソースについては、エラーが発生したバイト・オフセットを確認することはできません。

EXEC_SQL.Last_Error_Positionの例

```
PROCEDURE eslasterrorpos(sqlstr VARCHAR2) is
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;
  nErrPos PLS_INTEGER := 0;
  errmesg VARCHAR2(256);
```

```
BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION('');
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  BEGIN
  --
  -- parsing statement from caller
  --
    EXEC_SQL.parse(connection_id, cursorID, sqlstr, exec_sql.V7);
  --
  -- check for error in statement; find out position where statement syntax
is in error
  --
  EXCEPTION
    WHEN EXEC_SQL.PACKAGE_ERROR THEN
      nErrPos := EXEC_SQL.LAST_ERROR_POSITION(connection_id);
      TEXT_IO.PUT_LINE(' position in text where error occured '|| nErrPos);
      errmesg := EXEC_SQL.LAST_ERROR_MESG(connection_id);
      TEXT_IO.PUT_LINE(' error message ' || errmesg);
      RETURN;
  END;
  --
  -- here to execute statement
  --

  ...

  ... nRes := EXEC_SQL.EXECUTE(connection_id, cursorID);

  ...

  EXEC_SQL.CLOSE_CURSOR(connection_id, cursorID);
  EXEC_SQL.CLOSE_CONNECTION(connection_id);
END;
```

EXEC_SQL.Last_Row_Count

説明

フェッチされた行の累積数が返されます。

構文

```
FUNCTION EXEC_SQL.Last_Row_Count
   [Connid    IN CONNTYPE]
RETURN PLS_INTEGER;
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。

戻り値

整数。

使用上の注意

このファンクションは、EXEC_SQL.Fetch_RowsまたはEXEC_SQL.Execute_And_Fetchをコールした後に使用してください。EXEC_SQL.Executeコールの後に使用すると、0が返されます。

EXEC_SQL.Last_Row_Countの例

```
PROCEDURE eslastrowcount is
  connection_id EXEC_SQL.CONNTYPE;
  cursorID EXEC_SQL.CURSTYPE;
  sqlstr VARCHAR2(1000);
  nIgn PLS_INTEGER;
  nRows PLS_INTEGER := 0 ;
  mynum NUMBER;
BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION('connection_str');
  cursorID := EXEC_SQL.OPEN_CURSOR(connection_id);
  --
  -- in this query, we order the results explicitly
  --
  sqlstr := 'select empno from emp order by empno';
  EXEC_SQL.PARSE(connection_id, cursorID, sqlstr, exec_sql.V7);
  EXEC_SQL.DEFINE_COLUMN(connection_id, cursorID, 1, mynum);
  nIgn := EXEC_SQL.EXECUTE(connection_id, cursorID);
  LOOP
    nIgn := EXEC_SQL.FETCH_ROWS(connection_id, cursorID);
    --
    -- do whatever processing is desired
    --
    IF (nIgn > 0) THEN
      EXEC_SQL.COLUMN_VALUE(connection_id, cursorID, 1, mynum);

      ...

    END IF;
    nRows := EXEC_SQL.LAST_ROW_COUNT(connection_id);
    --
    -- In this example, we are only interested in the first 10 rows, and
exit after
    -- fetching them
    --
```

```
    IF (nRows > 10) THEN
      EXIT;
    END IF;
  END LOOP;
  EXEC_SQL.CLOSE_CURSOR(connection_id, cursorID);
  EXEC_SQL.CLOSE_CONNECTION(connection_id);
END;
```

EXEC_SQL.Last_SQL_Function_Code

説明

SQL文のタイプを示す、最後のSQLファンクション・コードが返されます。有効なファンクション・コードの一覧については、『Oracle8コール・インタフェース プログラマーズ・ガイド』を参照してください。

構文

```
FUNCTION EXEC_SQL.Last_SQL_Function_Code
    [Connid     IN CONNTYPE]
RETURN PLS_INTEGER;
```

パラメータ

Connid	**使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。**

戻り値

整数。

使用上の注意

このファンクションは、SQL文を解析した直後に使用してください。

EXEC_SQL.Last_SQL_Function_Codeの例

```
/*
** In this procedure, a statement is passed in and executed. If the statement
is a
** select statement, then further processing is initiated to determine
the column
** characteristics.
*/


PROCEDURE eslastfunccode(sqlstr VARCHAR2) IS
  connection_id EXEC_SQL.CONNTYPE;
  cursor_number EXEC_SQL.CursType;
```

```
  --
  -- The values for the function codes is dependent on the RDBMS version.
  --
  SELECTFUNCCODE PLS_INTEGER := 3;
  sql_str VARCHAR2(256);
  nColumns PLS_INTEGER := 0;
  nFunc PLS_INTEGER := 0;
  colName VARCHAR2(30);
  colLen PLS_INTEGER;
  colType PLS_INTEGER;
BEGIN
  connection_id := EXEC_SQL.OPEN_CONNECTION('connection_str');
  cursor_number := EXEC_SQL.OPEN_CURSOR(connection_id);
  EXEC_SQL.PARSE(connection_id, cursor_number, sql_str, exec_sql.V7);
  nIgnore := EXEC_SQL.EXECUTE(connection_id, cursor_number);
  --
  -- check what kind of function it is
  --
  nFunc := EXEC_SQL.LAST_SQL_FUNCTION_CODE(connection_id);
  IF (nFunc != SELECTFUNCCODE) THEN
    RETURN;
  END IF;
  --
  -- proceed to obtain the column characteristics
  --
  LOOP
    nColumns := nColumns + 1;
    BEGIN
      EXEC_SQL.DESCRIBE_COLUMN(connection_id, cursor_number,
                    nColumns, colName, colLen, colType);
          TEXT_IO.PUT_LINE(' col= ' || nColumns || ' name ' || colName  ||
                       ' len= ' || colLen   || ' type ' || colType );
    EXCEPTION
    WHEN EXEC_SQL.INVALID_COLUMN_NUMBER THEN
      EXIT;
    END;
  END LOOP;
  EXEC_SQL.CLOSE_CURSOR(connection_id, cursor_number);
  EXEC_SQL.LCOSE_CONNECTION(connection_id);
END;
```

EXEC_SQL.Last_Error_Code

説明

接続上で発生した最後のOracleエラー・コードを返します。

構文

```
FUNCTION EXEC_SQL.Last_Error_Code
    [Connid     IN CONNTYPE]
RETURN PLS_INTEGER;
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。

戻り値

整数。

使用上の注意

このファンクションは、EXEC_SQL.Package_Error例外が発生した直後に使用してください。

EXEC_SQL.Last_Error_Codeおよび EXEC_SQL.Last_Error_Mesgの例

```
/*
** In the following procedure, we execute a statement that is passed in.
If there
** are any exceptions shown, we check to see its nature using LAST_ERROR_CODE
** and LAST_ERROR_MESG.
*/


procedure eslastfunccode(sqlstr varchar2) is
  connection_id exec_sql.connType;
  cursor_number exec_sql.CursType;
  sql_str VARCHAR2(256);
  nIgnore pls_integer;
BEGIN
  connection_id := exec_sql.open_connection('connection_str');
  cursor_number := exec_sql.open_cursor(connection_id);
  exec_sql.parse(connection_id, cursor_number, sql_str, exec_sql.V7);
  nIgnore := exec_sql.execute(connection_id, cursor_number);
  exec_sql.close_cursor(connection_id, cursor_number);
  exec_sql.close_connection(connection_id);
--
-- check the error in the exception block
--
EXCEPTION
  WHEN exec_sql.package_error THEN
    text_io.put_line('error :'||
       to_char(exec_sql.last_error_code(connection_id))  || '  ' ||
       exec_sql.last_error_mesg(connection_id) );
  --
  -- ensure that even though an error has occurred, the cursor and connection
```

```
  -- are closed.
  --
  IF exec_sql.is_connected(connection_id) THEN
    IF exec_sql.is_open(connection_id, cursor_number) THEN
      exec_sql.close_cursor(connection_id, cursor_number);
    END IF;
    exec_sql.close_connection(connection_id);
  END IF;
END;
```

EXEC_SQL.Last_Error_Mesg

説明

接続上で発生した最後のエラー・コードのテキスト・メッセージを返します。

構文

```
FUNCTION EXEC_SQL.Last_Error_Mesg
   [Connid    IN CONNTYPE]
RETURN VARCHAR2;
```

パラメータ

Connid	使用する接続のハンドルです。接続を指定しないと、EXEC_SQL.Default_Connectionによって、キャッシュのプライマリDeveloper接続ハンドルが取り出されます。

戻り値

文字列。

使用上の注意

このファンクションは、EXEC_SQL.Package_Error例外が発生した直後に使用してください。

ヒント

結果セットの列数を取得するには、EXEC_SQL.Invalid_Column_Number例外が発生するまで、列を1から通してループします。

プライマリ・データベース接続の変更

EXEC_SQL.Default_Connectionをコールした後にプライマリDeveloper接続を変更すると、次回のEXEC_SQL.Default_Connectionコールの際にも、キャッシュのハンドルが返されます。 新しいプライマリ接続のハンドルは自動的に返されません。

確実に正しいハンドルを使用するために、プライマリ接続を変更する前に、常にEXEC_SQL.Close_Connection（引数なし）を使用してください。これにより、EXEC_SQLの以前の接続に割り当てられていたメモリー・リソースを解放することもできます。

LISTパッケージ

LISTパッケージ

List.Appenditem

説明

リストに項目が追加されます。

構文

```
PROCEDURE List.Appenditem
   (List Listofchar,
    item VARCHAR2);
```

パラメータ

List	**リスト。**
item	**リスト項目。**

List.Appenditemの例

```
 /*
** Add an item to the end of 'my_List' then
** print out the number of items in the List
*/
PROCEDURE append (my_List List.Listofchars) IS
BEGIN
```

```
  List.Appenditem(my_List, 'This is the last item.');
  Text_IO.Put_Line(List.Getitem(my_List,
    List.Nitems(my_List)));
END;
```

List.Destroy

説明

リスト全体が破棄されます。

構文

```
PROCEDURE List.Destroy
   (List Listofchar);
```

パラメータ

List	**リスト。**

List.Destroyの例

```
 /*
**  Destroy the List
*/
List.Destroy(my_package.my_List);
```

List.Deleteitem

説明

リスト内の指定した位置にある項目が削除されます。

構文

```
PROCEDURE List.Deleteitem
   (List Listofchar),
    pos  PLS_INTEGER);
```

パラメータ

List	**リスト。**
pos	**位置（ベースは0です）。**

List.Deleteitemの例

```
 /*
** Delete the third item from 'my_List'
```

```
*/
List.Deleteitem(my_package.my_List, 2));
```

List.Fail

説明

リスト操作が失敗すると呼び出されます。

構文

```
List.Fail   EXCEPTION;
```

List.Failの例

```
 /*
** Provide an exception handler for the
** List.Fail exception
*/
EXCEPTION
  WHEN List.Fail THEN
    Text_IO.Put_Line('list Operation Failed');
```

List.Getitem

説明

リストから項目が取り出されます。

構文

```
FUNCTION List.Getitem
   (List Listofchar,
    pos  PLS_INTEGER)
RETURN VARCHAR2;
```

パラメータ

List	**リスト。**
pos	**位置（ベースは0です）。**

戻り値

指定したリストの項目。

List.Getitemの例

```
 /*
```

```
** Retrieve and print out the second item
** in my_List
*/
Text_IO.Put_Line(List.Getitem(my_List, 1));
```

List.Insertitem

説明

リストの指定した位置に項目が挿入されます。

構文

```
PROCEDURE List.Insertitem
   (List Listofchar),
    pos  PLS_INTEGER,
    item VARCHAR2);
```

パラメータ

List	**リスト。**
pos	**位置（ベースは0です）。**
item	**リスト項目。**

List.Insertitem**の例**

```
/*
** Add a string to the List In third place
** then retrieve the third item and print it
*/
PROCEDURE insert_item(my_List List.Listofchar) IS
BEGIN
  List.Insertitem(my_List, 2, 'This is the third
    item.');
  Text_IO.Put_Line(List.Getitem(my_List, 2));
END;
```

List.Listofchar

説明

リストに対するハンドルが指定されます。

構文

```
TYPE List.Listofchar;
```

List.Listofcharの例

```
 /*
** Declare a variable of the type
** Listofchar, then create the List
*/
PROCEDURE my_proc IS
  my_List List.Listofchar;
BEGIN
  my_List := List.Make;
END;
```

List.Make

説明

新しい空のリストが作成されます。リストは使用する前に作成しておく必要があります。

構文

```
FUNCTION List.Make
RETURN Listofchar;
```

使用上の注意

このファンクションを使用して作成したリストは、List.Destroyプロシージャを使用して必ず破棄する必要があります。

List.Makeの例

```
 /*
** Create a List Of the type Listofchar
*/
PROCEDURE my_proc IS
  my_List List.Listofchar;
BEGIN
  my_List := List.Make;
END;
```

List.Nitems

説明

リストの項目数が戻されます。

構文

```
FUNCTION List.Nitems
   (List Listofchar)
RETURN PLS_INTEGER;
```

パラメータ

List	**リスト。**

List.Nitems**の例**

```
 /*
** For each item in my_List, retrieve the
** value of the item and print it out
*/
PROCEDURE print_List Is
BEGIN
  FOR i IN 0..List.Nitems(my_pkg.my_List)-1 LOOP
    Text_IO.Put_Line(List.Getitem(my_pkg.my_List, i));
  END LOOP;
END;
```

List.Prependitem

説明

リストの先頭にリスト項目が追加されます。

構文

```
PROCEDURE List.Prependitem
   (List Listofchar),
    item VARCHAR2);
```

パラメータ

List	**リスト。**
item	**リスト項目。**

List.Prependitem**の例**

```
 /*
** Insert a string to the beginning of my_List
** then retrieve it and print it out
*/
PROCEDURE prepend(my_List List.Listofchars) IS
BEGIN
  List.Prependitem(my_List, 'This is the first item.');
  Text_IO.Put_Line(List.Getitem(my_List, 0));
```

```
        END;
```

OLE2パッケージ

OLE2パッケージ

OLE2.Add_Arg

説明

OLE2.Create_Arglistを使用して作成した引数リストに引数が追加されます。

構文

```
PROCEDURE OLE2.Add_Arg
   (List  List_Type,
    value NUMBER);

PROCEDURE OLE2.Add_Arg
   (List  List_Type,
    value VARCHAR2);
```

パラメータ

List	**OLE2.Create_Arglist関数に割り当てられた引数リストの名前。**
value	**引数の値。**

使用上の注意

引数は、NUMBER型またはVARCHAR2型のどちらでも構いません。

OLE2.Add_Arg**の例**

```
 /*
** Add an argument to my_Arglist
*/
OLE2.Add_Arg(my_Arglist, 'Sales Revenue');
```

OLE2.Add_Arg_Obj

説明

OLE2.Create_Arglistを使用して作成した引数リストにオブジェクト引数が追加されます。

構文

```
PROCEDURE OLE2.Add_Arg_Obj
   (List IN List_Type,
    value IN Obj_Type);
```

パラメータ

List	**OLE2.Create_Arglist関数のコールから戻されたリスト・ハンドル。**
value	**OLE2オートメーション・サーバーに渡されるObj_Typeの値。**

OLE2.Add_Arg_Obj**の例**

```
 /*
** When the OLE interface must accept an unknown object
** as an argument instead of a pure scalar type, use the
** Add_Arg_Obj procedure.
*/
```

```
object = OLE2.CREATE_OBJ(obj_name);
listh := OLE2.CREATE_ARGLIST;

OLE2.ADD_ARG_OBJ(listh, object);
```

OLE2.Create_Arglist

説明

OLE2オートメーション・サーバーに渡す引数リストが作成されます。

構文

```
FUNCTION OLE2.Create_Arglist
RETURN List_Type;
```

戻り値

引数リストに対するハンドル。

OLE2.Create_Arglistの例

```
 /*
** Declare a variable of the type OLE2.List_Type
** then create the argument List Of that name
*/
  my_Arglist  OLE2.List_Type;
BEGIN
  my_Arglist := OLE2.Create_Arglist;
  OLE2.Add_Arg(my_Arglist, 'Sales Revenue');
END;
```

OLE2.Create_Obj

説明

OLE2オートメーション・オブジェクトが作成されます。

構文

```
FUNCTION OLE2.Create_Obj
   (object  VARCHAR2)
RETURN obj_type;
```

パラメータ

object	**OLE2オートメーション・オブジェクト。**

戻り値

OLE2オートメーション・オブジェクトに対するハンドル。

OLE2.Create_Objの例

```
 /*
** Create an OLE2 Object
*/
obj := OLE2.Create_Obj('Excel.Application.5');
```

OLE2.Destroy_Arglist

説明

OLE2.Create_Arglist関数を使用して、以前に作成した引数リストが破棄されます。

構文

```
PROCEDURE OLE2.Destroy_Arglist
   (List List_Type);
```

パラメータ

List	**OLE2.Create_Arglist関数に割り当てられた引数リストの名前。**

OLE2.Destroy_Arglistの例

```
 /*
** Destroy an argument list.
*/
OLE2.Destroy_Arglist(My_Arglist);
```

OLE2.Get_Char_Property

説明

OLE2オートメーション・オブジェクトのプロパティが取得されます。

構文

```
FUNCTION OLE2.Get_Char_Property
   (object   obj_type,
    property VARCHAR2,
    Arglist  List_Type := 0)
RETURN VARCHAR2;
```

パラメータ

object	OLE2オートメーション・オブジェクト。
property	OLE2オートメーション・オブジェクト内のプロパティの名前。
Arglist	OLE2.Create_Arglist関数に割り当てられた引数リストの名前。

戻り値

文字値。

OLE2.Get_Char_Propertyの例

```
 /*
** Get the property for the object.
*/
str := OLE2.Get_Char_Property(obj, 'text');
```

OLE2.Get_Num_Property

説明

OLE2オートメーション・オブジェクトの数値が取得されます。

構文

```
FUNCTION OLE2.Get_Num_Property
   (object   obj_type,
    property VARCHAR2,
    Arglist  List_Type := 0)
RETURN NUMBER;
```

パラメータ

object	OLE2オートメーション・オブジェクト。
property	OLE2オートメーション・オブジェクト内のプロパティの名前。
Arglist	OLE2.Create_Arglist関数に割り当てられた引数リストの名前。

戻り値

数値。

OLE2.Get_Num_Propertyの例

```
 /*
** Get the number value for the center of the map.
*/
x := OLE2.Get_Num_Property(Map,'GetMapCenterX');
y := OLE2.Get_Num_Property(Map,'GetMapCenterY');
```

OLE2.Get_Obj_Property

説明

OLE2オートメーション・オブジェクトのオブジェクト・タイプの値が取得されます。

構文

```
FUNCTION OLE2.Get_Obj_Property
   (object   obj_type,
    property VARCHAR2,
    Arglist  List_Type := 0)
RETURN OBJ_TYPE;
```

パラメータ

object	**OLE2オートメーション・オブジェクト。**
property	**OLE2オートメーション・オブジェクトの名前。**
Arglist	**OLE2.Create_Arglist関数に割り当てられた引数リストの名前。**

戻り値

OLE2オートメーション・オブジェクトのプロパティ。

OLE2.Get_Obj_Propertyの例

```
 /*
**Get the object type value for the spreadsheet object.
*/
the_obj:=OLE2.Get_Obj_Property(spreadsheet_obj,
       'Application');
```

OLE2.Invoke

説明

OLE2メソッドが起動されます。

構文

```
PROCEDURE OLE2.Invoke
   (object obj_type,
    method VARCHAR2,
    List   List_Type := 0);
```

パラメータ

object	**OLE2オートメーション・オブジェクト。**

method	OLE2オブジェクトのメソッド（プロシージャ）。
List	OLE2.Create_Arglist関数に割り当てられた引数リストの名前。

OLE2.Invokeの例

```
 /*
**Invoke ZoomIn.
*/
OLE2.Invoke(Map, 'ZoomIn', my_Arglist);
```

OLE2.Invoke_Num

説明

指定したメソッドを使用して、OLE2オートメーション・オブジェクトの数値が取得されます。

構文

```
FUNCTION OLE2.Invoke_Num
   (object   obj_type,
    method   VARCHAR2,
    Arglist  List_Type := 0)
RETURN NUMBER;
```

パラメータ

object	OLE2オートメーション・オブジェクト。
method	数値を戻すOLE2オートメーション・メソッド（関数）の名前。
Arglist	OLE2.Create_Arglist関数に割り当てられた引数リストの名前。

戻り値

数値。

OLE2.Invoke_Numの例

```
 /*
**  Get the number value using the specified method.
*/
the_num:=OLE2.Invoke_Num (spreadsheet_obj, 'npv',
       my_Arglist);
```

OLE2.Invoke_Char

説明

指定したメソッドを使用して、OLE2オートメーション・オブジェクトの文字値が取得されます。

構文

```
FUNCTION OLE2.Invoke_Char
   (object   obj_type,
    method   VARCHAR2,
    Arglist  List_Type := 0)
RETURN VARCHAR2;
```

パラメータ

object	**OLE2オートメーション・オブジェクト。**
method	**文字値を戻すOLE2オートメーション・メソッド（関数）の名前。**
Arglist	**OLE2.Create_Arglist関数に割り当てられた引数リストの名前。**

戻り値

文字値。

OLE2.Invoke_Charの例

```
 /*
** Get the character value for spell_obj.
*/
correct:=OLE2.Invoke_Char(spell_obj, 'spell', my_Arglist);
```

OLE2.Invoke_Obj

説明

OLE2オートメーション・オブジェクトのオブジェクト・タイプの値が取得されます。

構文

```
FUNCTION OLE2.Invoke_Obj
   (object   obj_type,
    method   VARCHAR2,
    Arglist  List_Type := 0)
RETURN OBJ_TYPE;
```

パラメータ

object	**OLE2オートメーション・オブジェクト。**

method	起動するOLE2オートメーション・メソッドの名前。
Arglist	OLE2.Create_Arglist関数に割り当てられた引数リストの名前。

戻り値

OLE2オートメーション・オブジェクト。

OLE2.Invoke_Objの例

```
 /*
** Get the object type value for wp_obj.
*/
para_obj:=OLE2.Invoke_Obj(wp_obj, 'get_para', my_Arglist);
```

OLE2.IsSupported

説明

OLE2パッケージが現行プラットフォームでサポートされることが確認されます。

構文

```
OLE2.ISSUPPORTED
```

戻り値

プラットフォームでOLE2がサポートされていればTRUE、そうでなければFALSEが戻されます。

OLE2.IsSupportedの例

```
/*
** Before calling an OLE2 object in platform independent code,
** use this predicate to determine if OLE2 is supported on the
** current platform.
*/
IF (OLE2.ISSUPPORTED)THEN
       . . . PL/SQL code using the OLE2 package
ELSE
       . . . message that OLE2 is not supported
END IF;
```

OLE2.Last_Exception

説明

PL/SQL例外によって通知された最後のOLE2例外が戻されます。

構文

```
FUNCTION last_exception return NUMBER;
```

または

```
FUNCTION last_exception(message OUT VARCHAR2) return NUMBER;
```

パラメータ

message	テキスト文字列（VARCHAR2）で、OLE2エラー・メッセージのテキストが含まれます。 この変数に情報が含まれていれば、コール元に対してエラー・コード値に加えて戻された情報です。

戻り値

最後のOLE2例外から戻された完全なOLE2エラー・コード。

使用上の注意

- この関数に対しては、どちらの構文も使用できます。1番目の構文では、エラー・コードのみが戻されます。2番目の構文を使用すると、エラー・コードに加えてテキストの説明も戻されます。
- この関数は、完全なOLE2 (Windows)型のエラー・コードをNUMBERとして戻します。エラー・コード部分のみを抽出するには、最高位ビット（Severity）を正確に削除し、残りの数値をINTEGERまたはBINARY_INTEGER形式に変換します。エラー・コードを整数として抽出するプロシージャの例は、OLE2.Last_Excecptionの例を参照してください。

OLE2.Last_Exceptionの例

```
PACKAGE olepack IS
  PROCEDURE init(...);
  PROCEDURE something(...);
  PROCEDURE shutdown(...);
  FUNCTION get_err(message OUT VARCHAR2) RETURN BINARY_INTEGER;
END olepack;

PACKAGE BODY olepack IS
...
  FUNCTION get_err(message OUT VARCHAR2) RETURN BINARY_INTEGER IS
  --
  -- OLE errors are formatted as 32 bit unsigned integers and
  -- returned as Oracle NUMBERS.We want to extract only the
  -- error code, which is contained in the lowest 16 bits.
  -- We must first strip off the top [severity] bit, if it
  -- exists.Then, we must translate the error to an
  -- INTEGER or BINARY INTEGER and extract the error code.
  --
```

```
  -- Some helpful constants:
  --  0x80000000 = 2147483648
  -- 0x100000000 = 4294967296
  --  0x0000FFFF =     65535
  --
     hibit    NUMBER := 2147483648;
     four_gig  NUMBER := 4294967296;
     code_mask NUMBER := 65535;
     excep    NUMBER;
     trunc_bi  BINARY_INTEGER;
     ole_code  BINARY_INTEGER;
  BEGIN
     excep := OLE2.LAST_EXCEPTION(message);
     IF (excep >= hibit) AND (excep <= four_gig) THEN
        trunc_bi := excep - hibit;
     END IF;
     -- Mask out just the Code section
     ole_code := BITAND(trunc_bi, code_mask);
     RETURN ole_code;
  END get_err;
END olepack;

PROCEDURE ole_test IS
  err_code BINARY_INTEGER;
  err_text VARCHAR2(255);
BEGIN
  olepack.init(...);
  olepack.something(...);
  olepack.shutdown(...);
EXCEPTION
  WHEN OLE2.OLE_ERROR THEN
    err_code := olepack.get_err(err_text);
    TEXT_IO.PUT_LINE('OLE Error #' || err_code || ':' || err_text);
    olepack.shutdown(...);
END ole_test;
```

OLE2.List_Type

説明

引数リストに対するハンドルが指定されます。

構文

```
List   OLE2.LIST_TYPE;
```

OLE2.List_Type の例

```
...
 alist   OLE2.LIST_TYPE;
...
   alist := OLE2.CREATE_ARGLIST;
   OLE2.ADD_ARG(alist, <argument1>);
   OLE2.ADD_ARG(alist, <argument2>);
   ...

   wkbook := OLE2.INVOKE_OBJ(my_obj, 'method1', alist);
   ...
```

OLE2.Obj_Type

説明

OLE2オートメーション・オブジェクトに対するハンドルが指定されます。

構文

obj OLE2.OBJECT_TYPE;

使用上の注意

- OLE2.Obj_Typeは、OLE2パッケージのunknownオブジェクトに相当するものです。
- 詳細は、Formsドキュメンテーション内のFORMS_OLE.GET_INTERFACE_POINTERを参照してください。

OLE2.Obj_Typeの例

```
/*
** Create an OLE2 Object
*/
 obj OLE2.OBJECT_TYPE;
 ...
  obj := OLE2.Create_Obj('Excel.Application.5');
 ...
```

OLE2.OLE_Error

説明

この例外は、OLE2パッケージでのエラー時に発生します。

構文

```
OLE2.OLE_ERROR EXCEPTION;
```

OLE2.OLE_Errorの例

```
PROCEDURE ole_test IS
  err_code BINARY_INTEGER;
  err_text VARCHAR2(255);
BEGIN
  olepack.init(...);
  olepack.something(...);
  olepack.shutdown(...);
EXCEPTION
  WHEN OLE2.OLE_ERROR THEN
    err_code := olepack.get_err(err_text);
    TEXT_IO.PUT_LINE('OLE Error #' || err_code || ':' || err_text);
    olepack.shutdown(...);
END ole_test;
```

OLE2.OLE_Not_Supported

説明

OLE2パッケージをコールしたときに、OLE2が現行ソフトウェア・プラットフォームでサポートされていない場合、この例外がコールされます。

構文

```
OLE2.OLE_NOT_SUPPORTED EXCEPTION;
```

OLE2.OLE_Not_Supportedの例

```
PROCEDURE ole_test IS
  err_code BINARY_INTEGER;
  err_text VARCHAR2(255);
BEGIN
  olepack.init(...);
  olepack.something(...);
  olepack.shutdown(...);
EXCEPTION
```

```
  WHEN OLE2.OLE_NOT_SUPPORTED THEN
    TEXT_IO.PUT_LINE('OLE2 is not supported on this computer');
    olepack.shutdown(...);
END ole_test;
```

OLE2.Release_Obj

説明

PL/SQLクライアントにとって不要になったOLE2オートメーション・オブジェクトが通知されます。

構文

```
PROCEDURE OLE2.Release_Obj
   (object   obj_type);
```

パラメータ

object	**OLE2オートメーション・オブジェクト。**

使用上の注意

オペレーティング・システムはそのオブジェクトに関連付けられたリソースの割当てを解除できます。ユーザーは、OLE2パッケージを使用して作成または起動した各OLE2オートメーション・オブジェクトを解放する必要があります。

OLE2.Release_Objの例

```
 /*
**Release the OLE2 object objap.
*/
declare
  objap   OLE2.Obj_Type;
begin
  objap:=OLE2.Create_Obj('Excel.application.5');
  OLE2.Release_Obj(objap);
end;
```

OLE2.Set_Property

説明

OLE2オートメーション・オブジェクトのプロパティ値が設定されます。

構文

```
PROCEDURE OLE2.Set_Property
   (object   obj_type,
    property VARCHAR2,
    value    NUMBER,
    Arglist  List_Type := 0);

PROCEDURE OLE2.Set_Property
   (object   obj_type,
    property VARCHAR2,
    value    VARCHAR2,
    Arglist  List_Type := 0);
```

パラメータ

object	**OLE2オートメーション・オブジェクト。**
property	**OLE2オートメーション・オブジェクト内のプロパティの名前。**
value	**プロパティの値。**
Arglist	**OLE2.Create_Arglist関数に割り当てられた引数リストの名前。**

OLE2.Set_Propertyの例

```
/*
**Set properties for the OLE2 object `Excel.Application'.
*/
application:=OLE2.CREATE_OBJ('Excel.Application');
OLE2.Set_Property(application,'Visible', 'True');

workbooks:=OLE2.INVOKE_OBJ(application, 'Workbooks');
workbook:=OLE2.INVOKE_OBJ(workbooks,'Add');
worksheets:=OLE2.INVOKE_OBJ(workbook, 'Worksheets');
worksheet:=OLE2.INVOKE_OBJ(worksheets,'Add');
args:=OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(args, 4);
OLE2.ADD_ARG(args, 2);
cell:=OLE2.Invoke_Obj(worksheet, 'Cells', args);
OLE2.DESTROY_ARGLIST(args);
OLE2.Set_Property(cell, 'Value', 'Hello Excel!');
```

ORA_FFIパッケージ

ORA_FFIパッケージ

Ora_Ffi.Ffi_Error

説明

Ora_Ffiパッケージを使用しているときにエラーが発生するとコールされます。

構文

```
EXCEPTION  Ora_Ffi_Error;
```

Ora_Ffi.Ffi_Errorの例

```
/* This example uses Ora_Ffi_Error */
```

```
PRODEDURE register_libs IS
   testlib_lhandle ora_ffi.libhandletype;
BEGIN
   /* Attempt to load a dll library
      from a non-existant directory*/
   testlib_lhandle := ora_ffi.load_library
            ('C:¥baddir¥', 'libtest.dll');
EXCEPTION
   WHEN Ora_Ffi.Ffi_Error THEN
      /* print error message */
      text_io.put_line(tool_err.message);
      /* discard the error */
      tool_err.pop;
END;
```

Ora_Ffi.Find_Function

説明

指定したファンクションに対するファンクション・ハンドルを探して、そのハンドルを戻します。ファンクション名またはライブラリ名のどちらかを指定することにより、ファンクション・ハンドルを検索できます。このファンクションは、Ora_Ffi.Register_Functionで登録済みのものでなければなりません。

構文

```
FUNCTION Ora_Ffi.Find_Function
   (libHandle  libHandleType,
    funcname   VARCHAR2)
RETURN funcHandleType;
FUNCTION Ora_Ffi.Find_Function
   (libname   VARCHAR2,
    funcname  VARCHAR2)
RETURN funcHandleType;
```

パラメータ

libHandle	Ora_Ffi.Load_Libraryまたは、Ora_Ffi.Find_Libraryによって戻されるライブラリ・ハンドル。
funcname	見つけるファンクションの名前。
libname	ファンクションが入っているライブラリの名前。

戻り値

指定したファンクションに対するハンドル。

Ora_Ffi.Find_Functionの例

```
/* Find foreign function handle for
   a given foreign library handle and
   foreign function name */

BEGIN
   ...
   funchandle := ora_ffi.find_function
           (libhandle, 'my_func');
   ...
END;


/* Find foreign function handle for
   a given foreign function and
   foreign library names */
BEGIN
   ...
   funchandle := ora_ffi.find_function
           (libhandle, 'my_func');
   ...
END;
```

Ora_Ffi.Find_Library

説明

指定した外部ライブラリ名に対するハンドルを探して、そのハンドルを戻します。このライブラリは、Ora_Ffi.Load_Libraryで登録済みのものである必要があります。

構文

```
FUNCTION Ora_Ffi.Find_Library (libname VARCHAR2)
RETURN libHandleType;
```

パラメータ

libname	ライブラリ名。

戻り値

指定した外部ライブラリに対するハンドル。

Ora_Ffi.Find_Libraryの例

```
/* Find foreign library handle for
```

```
   a given library name */

BEGIN
   ...
   libhandle := ora_ffi.find_library
          ('mylib.dll');
   ...
END;
```

Ora_Ffi.Funchandletype

説明

外部ファンクションに対するハンドルが指定されます。Ora_Ffi.Find_Functionを使用して、指定されたハンドルを取得できます。

構文

```
TYPE Ora_Ffi.Funchandletype;
```

Ora_Ffi.Funchandletypeの例

```
/* This example uses Ora_Ffi_Funchandletype */

PROCEDURE define_c_funcs IS
   getresult_fhandle  ora_ffi.funcHandleType;
   foo_fhandle       ora_ffi.funcHandleType;
BEGIN
   /* Register the info for function getresult */
   getresult_fhandle := ora_ffi.register_function
            (testlib_lhandle, 'getresult');
   ...
   /* Register the info for function foo */
   foo_fhandle := ora_ffi.register_function
            (testlib_lhandle,'foo');
   ...
END;
```

Ora_Ffi.Generate_Foreign

説明

指定したライブラリで定義されているすべてのファンクションのPL/SQLコード・パッケージが生成されます。まず、ライブラリをロードし、起動するファンクションをすべて登録し、それらのパラメータおよび戻り値を登録する必要があります。

構文

```
PROCEDURE Ora_Ffi.Generate_Foreign
   (handle libHandleType);
PROCEDURE Ora_Ffi.Generate_Foreign
   (handle  libHandleType,
    pkgname VARCHAR2);
```

パラメータ

handle	Ora_Ffi.Load_Libraryまたは、Ora_Ffi.Find_Libraryによって戻されるライブラリ・ハンドル。
pkgname	生成されるパッケージの名前。 パッケージ名を指定しない場合、ライブラリ名の先頭にFFI_を付けた名前が使用されます。たとえば、ライブラリ名がLIBTESTの場合、パッケージ名はFFI_LIBTESTになります。

使用上の注意

- Ora.Ffi.Generate_Foreignファンクションによって生成されるパッケージは、現在の名前空間の中に作成され、Procedure Builderのオブジェクト・ナビゲータでは「プログラム単位」ノードの下に表示されます。パッケージが生成されたら、PL/SQLライブラリの「プログラム単位」ノードまたはデータベースの「ストアド・プログラム単位」ノードにそれをコピーし、「ファイル」→「テキスト・エクスポート」を使用して、定義したその他の新しいパッケージまたはプロシージャと同じように、テキスト・ファイルにエクスポートできます。

- Ora_Ffi.Generate_Foreignファンクションによって生成されるPL/SQLパッケージは、登録されているファンクションのそれぞれに対して、必要なPRAGMAコンパイラ・ディレクティブを自動的にインクルードします。

  ```
  PRAGMA interface (C, func_name, 11265);
  ```

  *func_name*は、すでにロードされているDLLライブラリからの、登録済み外部ファンクションの名前です。生成されるPL/SQLパッケージの名前を指定できますが、そのパッケージの内部では、各エントリ・ポイントはそれらがマッピングする外部ファンクションの名前に一致します。

Ora_Ffi.Generate_Foreignの例

```
/* Define components of package test */

PACKAGE test IS
...
END;

/*Define package body procedures */
PACKAGE BODY test IS
   PRODEDURE register_libs IS
   BEGIN
      /* Load the test library */
```

```
      testlib_lhandle := Ora_Ffi_.load_library
        ('c:¥orawin95¥oralibs¥','testlib.dll');
    END;

    PROCEDURE define_c_funcs IS
      getresult_fhandle Ora_Ffi.Funchandletype;
      foo_handle       Ora_Ffi.Funchandletype;
    BEGIN
      /* Register the info for function getresult */
      getresult_fhandle := ora_ffi.register_function
           (testlib_lhandle,'getresult');
      ...
      /* Register the info for function foo */
      foo_fhandle := ora_ffi.register_function
           (testlib_lhandle,'foo');
      ...
      /* Generate PL/SQL package containing all
         functions defined in test library */
      ora_ffi.generate_foreign
           (testlib_lhandle, 'test_ffi_pkg');
      ...
    END;
END;
```

Ora_Ffi.Is_Null_Ptr

説明

ライブラリ、ファンクションまたはポインタに対するハンドルがNULLかどうか判断されます。

構文

```
FUNCTION Ora_Ffi.Is_Null_Ptr (handle libHandleType)
RETURN BOOLEAN;
FUNCTION Ora_Ffi.Is_Null_Ptr (handle  funcHandleType)
RETURN BOOLEAN;
FUNCTION Ora_Ffi.Is_Null_Ptr (handle  pointerType)
RETURN BOOLEAN;
```

パラメータ

handle	評価するライブラリ、ファクションまたはポインタ。

戻り値

TRUE	ハンドルがNULLの場合。
FALSE	ハンドルがNULLでない場合。

Ora_Ffi.Is_Null_Ptrの例

```
/* This example uses Ora_Ffi.Is_Null_Ptr */

PROCEDURE register_libs IS
...
BEGIN
   /* Load foreign function library */
   libhandle := Ora_Ffi.load_library
      ('C:¥oralibs¥', 'libfoo.dll');
   /* Test whether library is null */
   IF (ora_ffi.is_null_ptr(libhandle)) THEN
   ...
   END IF;
END;
```

Ora_Ffi.Libhandletype

説明

外部ファンクションに対するハンドルが指定されます。指定されたハンドルを取得するには、Ora_Ffi.Find_Functionを使います。

構文

```
TYPE Ora_Ffi.Libhandletype;
```

Ora_Ffi.Libhandletypeの例

```
/* This example uses Ora_Ffi.Libhandletype */

PACKAGE test is
   /* Specify that testlib_lhandle
      is a library handle varible type */
   testlib_lhandle  ora_ffi.libHandleType;
   ...
END;

PACKAGE BODY test IS
   PROCEDURE register_libs IS
   BEGIN
      testlib_lhandle := Ora_Ffi.Load_library
          ('C:¥libdir¥','test.dll');
      ...
   END;
   ...
END;
```

Ora_Ffi.Load_Library

説明

ファンクションを登録するために指定した動的ライブラリがロードされます。

構文

```
FUNCTION Ora_Ffi.Load_Library
   (dirname VARCHAR2,
    libname VARCHAR2)
RETURN libHandleType;
```

パラメータ

dirname	ライブラリが存在するディレクトリ。
libname	ライブラリのファイル名。

戻り値

外部ライブラリに対するハンドル。ライブラリが見つからないか、またはロードできない場合は、NULLのハンドルを戻します。

Ora_Ffi.Load_Libraryの例

```
/* This example uses Ora_Ffi.Load_Library */

PACKAGE test is
   /* Declare testlib_lhandle as an Ora_Ffi
      library handle varible type.
   testlib_lhandle  ora_ffi.libHandleType;
   ...
END;

PACKAGE BODY test IS
   PROCEDURE register_libs IS
   BEGIN
      /* Load the dynamic link library 'test.dll'
         from the directory C:¥libdir¥ and return
         the handle testlib_lhandle.*/
      testlib_lhandle := Ora_Ffi.Load_library
           ('C:¥libdir¥','test.dll');
   ...
   END;
   ...
 END;
```

Ora_Ffi.Pointertype

説明

汎用Cポインタ（つまり、型が指定されないポインタ）の値とみなすことができます。

構文

```
TYPE Ora_Ffi.Pointertype;
```

Ora_Ffi.Pointertypeの例

```
/* This example uses Ora_Ffi.Pointertype */

PACKAGE imglib IS
   /* Declare Function get_image which
      returns a generic C pointer.*/
   FUNCTION get_image(ikey IN OUT VARCHAR2)
     RETURN Ora_Ffi.Pointertype ;
   /* Declare Procedure show_image with parameter
      idata which is a generic C pointer.*/
   PROCEDURE show_image(idata Ora_Ffi.Pointertype,
                        iscale NUMBER);
END;
...

PROCEDURE display_image(keywrd IN OUT VARCHAR2) IS
   /* Declare img_ptr as a generic C pointer type */
   img_ptr Ora_Ffi.Pointertype;
BEGIN
   img_ptr := imglib.get_image(keywrd);
   imglib.show_image(img_ptr,2);
END;
```

Ora_Ffi.Register_Function

説明

指定した外部ファンクションが登録されます。

構文

```
FUNCTION Ora_Ffi.Register_Function
  (libHandle  libHandleType,
   funcname   VARCHAR2,
   callstd    NUMBER   := C_STD)
RETURN funcHandleType;
```

パラメータ

libHandle	Ora_Ffi.Load_Libraryまたは、Ora_Ffi.Find_Libraryによって戻されるライブラリ・ハンドル。
funcname	登録されるファンクションの名前。
callstd	外部ファンクションによって使用されるコール。（詳細は、使用するコンパイラのマニュアルを参照してください。この引数の値には、次のパッケージ化定数のどちらかを指定できます。 C_STD 外部ファンクションがCコール標準を使用することを意味します。 PASCAL_STD 外部ファンクションがPascalコール標準を使用することを意味します。

戻り値

外部ファンクションに対するハンドル。

Ora_Ffi.Register_Functionの例

```
/* Define Procedure define_c_funcs which calls two
   Ora_Ffi functions, getresult and foo.*/

PROCEDURE define_c_funcs is
   getresult_fhandle  ora_ffi.funcHandleType;
   foo_fhandle       ora_ffi.funcHandleType;
BEGIN
   /* Register the info for function getresult */
   getresult_fhandle := ora_ffi.register_function
          (testlib_lhandle,'getresult');
   ...

   /* Register the info for function foo */
   foo_fhandle := ora_ffi.register_function
          (testlib_lhandle,'foo');
   ...
   /* Generate PL/SQL package containing all
      functions defined in test library */
   ora_ffi.generate_foreign
          (testlib_lhandle, 'test_ffi_pkg');
   ...
END;
```

Ora_Ffi.Register_Parameter

説明

指定した外部ファンクションの現行引数の型が登録されます。

構文

```
PROCEDURE Ora_Ffi.Register_Parameter
   (funcHandle funcHandleType,
    cargtype   PLS_INTEGER);
PROCEDURE Ora_Ffi.Register_Parameter
   (funcHandle funcHandleType,
    cargtype   PLS_INTEGER,
    plsargtype PLS_INTEGER);
```

パラメータ

funcHandle	Ora_Ffi.Register_FunctionまたはOra_Ffi.Find_Functionによって戻されるファンクション・ハンドル。
cargtype	コールされるC外部ファンクションに対する、現行の引数のCデータ型。この引数の値には、次のパッケージ化定数のどちらかを指定できます。 C_CHAR CHARを意味します。 C_CHAR_PTR CHAR *を意味します。 C_DOUBLE DOUBLEを意味します。 C_DOUBLE_PTR DOUBLE *を意味します。 C_FLOAT FLOATを意味します。 C_FLOAT_PTR FLOAT *を意味します。 C_INT INTを意味します。 C_INT_PTR INT *を意味します。 C_LONG LONGを意味します。 C_LONG_PTR LONG *を意味します。 C_SHORT SHORTを意味します。 C_SHORT_PTR SHORT *を意味します。 C_VOID_PTR VOID *を意味します。
plsargtype	対応するPL/SQLの引数型（オプション）。

Ora_Ffi.Register_Parameterの例

```
/* Define Procedure define_c_funcs which calls two
   Ora_Ffi functions, getresult and foo.*/

PROCEDURE define_c_funcs is
   getresult_fhandle   ora_ffi.funcHandleType;
   foo_fhandle        ora_ffi.funcHandleType;

BEGIN
   /* Register the info for function getresult */
   getresult_fhandle := ora_ffi.register_function
          (testlib_lhandle,'getresult');
  ...
   /* Register the info for function foo */
   foo_fhandle := ora_ffi.register_function
          (testlib_lhandle,'foo');
   /* Register the return type for function foo */
   ora_ffi.register_return
```

```
        (foo_fhandle, ora_ffi.C_SHORT);
   /* Register the parameter info for function foo */
   ora_ffi.register_parameter
        (foo_fhandle, ora_ffi.C_FLOAT);
   ora_ffi.register_parameter
        (foo_fhandle, ora_ffi.C_INT);
   ora_ffi.register_parameter
        (foo_fhandle, ora_ffi.C_CHAR_PTR);

   /* Generate PL/SQL package containing all functions
        defined in test library */
   ora_ffi.generate_foreign
        (testlib_lhandle, 'test_ffi_pkg');
   ...
END;
```

Ora_Ffi.Register_Return

説明

指定した外部ファンクションの戻り値の型が登録されます。

構文

```
PROCEDURE Ora_Ffi.Register_Return
   (funcHandle  funcHandleType,
    creturntype PLS_INTEGER);
PROCEDURE Ora_Ffi.Register_Return
   (funcHandle  funcHandleType,
    creturntype PLS_INTEGER,
    plsreturntype PLS_INTEGER);
```

パラメータ

funcHandle	**Ora_Ffi.Register_FunctionまたはOra_Ffi.Find_Functionによって戻されるファンクション・ハンドル。**

creturntype	外部ファンクションによって戻されるCデータ型。この引数の値には、次のパッケージか定数のどちらかを指定できます。 C_CHAR　CHARを意味します。 C_CHAR_PTR　CHAR *を意味します。 C_DOUBLE　DOUBLEを意味します。 C_DOUBLE_PTR　DOUBLE *を意味します。 C_FLOAT　FLOATを意味します。 C_FLOAT_PTR　FLOAT *を意味します。 C_INT　INTを意味します。 C_INT_PTR　INT *を意味します。 C_LONG　LONGを意味します。 C_LONG_PTR　LONG *を意味します。 C_SHORT　SHORTを意味します。 C_SHORT_PTR　SHORT *を意味します。 C_VOID_PTR　VOID *を意味します。
plsreturntype	対応するPL/SQLの戻り型（オプション）。

Ora_Ffi.Register_Returnの例

```
/* Define Procedure define_c_funcs which calls two
   Ora_Ffi functions, getresult and foo.*/

PROCEDURE define_c_funcs is
   getresult_fhandle   ora_ffi.funcHandleType;
   foo_fhandle        ora_ffi.funcHandleType;

BEGIN
   /* Register the info for function getresult */
   getresult_fhandle := ora_ffi.register_function
          (testlib_lhandle,'getresult');
   /* Register the return type for function getresult */
   ora_ffi.register_return
          (getresult_fhandle, ora_ffi.C_CHAR_PTR);

   /* Register the info for function foo */
   foo_fhandle := ora_ffi.register_function
          (testlib_lhandle,'foo');
   /* Register the return type for function foo */
   ora_ffi.register_return
          (foo_fhandle, ora_ffi.C_SHORT);
   ...
   /* Generate PL/SQL package containing all
      functions defined in test library */
   ora_ffi.generate_foreign
           (testlib_lhandle, 'test_ffi_pkg');
   ...
END;
```

Ora_Ffi.Unload_Library

説明

指定した動的ライブラリがアンロードされます。ライブラリのファンクションは、そのライブラリを再ロードするまで、アクセスできなくなります。

構文

```
PROCEDURE Ora_Ffi.Unload_Library
   (libHandle  libHandleType);
```

パラメータ

libHandle	**アンロードされるライブラリに対するハンドル。**

Ora_Ffi.Unload_Libraryの例

```
/* First load a dll library */

PROCEDURE register_libs IS
   test_lib  Ora_Ffi.Libhandletype;
BEGIN
   /* Load the testlib.dll library
      from directory C:¥libs¥ */
   testlib_lhandle := ora_ffi.load_library
          ('C:¥libs¥', 'testlib.dll');
END;

/* Generate PL/SQL Package containing
   funtions from the test library.*/

PROCEDURE define_c_funcs IS
   ...
   Ora_Ffi.Genereate_Foreign (testlib_lhandle,
         'test_Ffi_Pkg')';
   ...
END;

/* Unload the library */

PROCEDURE unload_libs IS
BEGIN
   /* Unload the dll library assigned to the
      library handle 'test_lib.'*/
   Ora_Ffi.Unload_library(testlib_lhandle);
   ...
END;
```

Ora_Ffiの例1A

Microsoft Windowsのランタイム・ライブラリC:¥windows¥system¥msvcrt.dllにあるCファンクションpowのためのインタフェースを作成する場合を考えます（powファンクションは、xをy乗します）。

```
int pow(int x, int y)
```

まず、ライブラリを表すパッケージ仕様を作成し、起動するPL/SQLファンクションを定義します。

```
PACKAGE mathlib IS
   FUNCTION pow(x NUMBER, y NUMBER)
     RETURN NUMBER;
END;
```

上で定義したPL/SQLファンクションmathlib.powをコールし、動的ライブラリmsvcrt.dllから外部ファンクション*pow*をコールします。

このサブプログラムは、ライブラリまたは外部ファンクションへのハンドルを必要としないことに注意してください。便宜を図るために、さまざまな登録は下記で定義するパッケージ本体で処理されます。

注意: この例はPRAGMAコンパイラ・ディレクティブを使用して、ファンクションff_to_powerがPL/SQLファンクションではなく、実際にはCファンクションとしてコンパイルされることをPL/SQLコンパイラに伝えます。Ora_Ffiの例1Bでは、Ora_Ffi.Generate_Foreignファンクションを使用してPL/SQLのmathlibパッケージを生成し、同じ結果を得る方法を示します。例1Bでは、PRAGMAディレクティブは、Ora_Ffi.Generate_Foreignファンクションによって自動的に処理されます。

```
PACKAGE BODY mathlib IS
  /* Declare the library and function handles.*/
  mathlib_lhandle   Ora_Ffi.Libhandletype ;
  to_power_fhandle  Ora_Ffi.Funchandletype ;

  /* Create the PL/SQL function that will actually */
  /* invoke the foreign function.                  */
  FUNCTION ff_to_power(fhandle Ora_Ffi.Funchandletype,
        x NUMBER, y NUMBER)RETURN NUMBER;
  PRAGMA interface(C, ff_to_power, 11265);

  /* Create the PL/SQL function that is defined in */
  /* the package spec.This function simply      */
  /* passes along the arguments it receives to    */
  /* ff_to_power (defined above), prepending the   */
  /* foreign function handle to the argument List.*/
  FUNCTION pow(x NUMBER, y NUMBER) RETURN NUMBER IS
  BEGIN
    RETURN(ff_to_power(to_power_fhandle, x, y));
```

```
  END pow;

/* Define the body of package mathlib */
BEGIN

  /* Load the library.*/
     mathlib_lhandle := Ora_Ffi.Load_Library
        ('C:¥WINDOWS¥SYSTEM¥', 'msvcrt.dll');

     /* Register the foreign function.*/
  to_power_fhandle := Ora_Ffi.Register_Function
        (mathlib_lhandle, 'pow', Ora_Ffi.C_Std);

  /* Register both parameters of function to_power.*/
  Ora_Ffi.Register_Parameter (to_power_fhandle,
                         Ora_Ffi.C_DOUBLE);
  Ora_Ffi.Register_Parameter(to_power_fhandle,
                         Ora_Ffi.C_DOUBLE);

  /* Register the return type.*/
  Ora_Ffi.Register_Return (to_power_fhandle, Ora_Ffi.C_DOUBLE);

END;  /* Package Body Mathlib */
```

mscvrt.dllからのCファンクションpowを起動するには、次の例のように、mathlibパッケージ仕様部に定義されたPL/SQLファンクションpowをコールします。次に例を示します。

```
PL/SQL>
PROCEDURE raise_to_power (a in number, b in number) IS
   BEGIN
      text_io.put_line(mathlib.pow(a,b));
   END;
PL/SQL> raise_to_power(2,9);
512
```

Ora_Ffiの例1B

次に示すのは、Ora_Ffiの例1Aで示したCファンクションpowを実装する別の方法です。この例では、Ora_Ffi.Generate_Foreignファンクションを使用してPL/SQLパッケージを生成します。外部Cファンクションのコンパイルに必要なPRAGMAコンパイラ・ディレクティブは、生成されるパッケージに自動的にインクルードされるため、下記のパッケージ本体の中では使用されていません。

```
/* Create package mathlib that will generate a PL/SQL
package using a foreign file C function to raise a
```

```
number to a power.The parameter, pkg_name, lets you
specify the name of the generated package.*/
PACKAGE mathgen IS
   PROCEDURE gen(pkg_name IN VARCHAR2);
END;


PACKAGE BODY mathgen IS
   /* Define the 'gen' procedure that will generate the
      foreign file package.*/
   PROCEDURE gen(pkg_name IN VARCHAR2) IS
      /* Declare the library and function handles.*/
      mathlib_lhandle   Ora_Ffi.Libhandletype ;
      to_power_fhandle  Ora_Ffi.Funchandletype ;

   BEGIN   /* package body mathlib */
      /* Load the library.*/
      mathlib_lhandle := Ora_Ffi.Load_Library
         ('C:¥WINDOWS¥SYSTEM¥', 'msvcrt.dll');

      /* Register the foreign function.*/
      to_power_fhandle := Ora_Ffi.Register_Function
         (mathlib_lhandle, 'pow', Ora_Ffi.C_Std);

      /* Register both parameters of the foreign function.*/
      Ora_Ffi.Register_Parameter (to_power_fhandle,
                           Ora_Ffi.C_DOUBLE);
      Ora_Ffi.Register_Parameter(to_power_fhandle,
                           Ora_Ffi.C_DOUBLE);

      /* Register the return type of the foreign function.*/
      Ora_Ffi.Register_Return (to_power_fhandle, Ora_Ffi.C_DOUBLE);

      /* Generate a PL/SQL package containing the foreign C function,
         'pow.'You can name the new package by specifying a value
          for the parameter, pkg_name, when you generate the package.*/
      Ora_Ffi.generate_foreign(mathlib_lhandle, pkg_name);

   END; /* Procedure gen */
END; /* Package Body mathgen */
```

このメソッドを使用して、数値を累乗するには、最初にパッケージmathgenとプロシージャ*gen.*を使用してPL/SQLパッケージを生成します。たとえば、生成されたPL/SQL累乗パッケージをmathlibと呼ぶ場合、次のようにそのパッケージを生成します。

```
PL/SQL> mathgen.gen('mathlib');
```

次に、パッケージmathlibから累乗ファンクションを起動するには、次のようなプロシージャを記述します。

```
PROCEDURE raise_to_power (a in number, b in number) IS
   BEGIN
     text_io.put_line(mathlib.pow(a,b));
   END;
PL/SQL> raise_to_power(5,2);
25
```

Ora_Ffiの例2

ライブラリC:¥oralibs¥imglib.dllにある、次のCファンクションへのインタフェースを作成することを想定します。

```
void *get_image(char *imgkey)
void show_image(void *binimage, float iscale)
```

get_imageファンクションはキーワード引数を使用して画像データをロードし、次にそのバイナリ・データを指す汎用ポインタ（型指定のないポインタ）を戻します。次に、画像を画面に表示するshow_imageにそのポインタと拡大係数を渡します。

まず、ライブラリを表現するパッケージ仕様部を作成し、起動するPL/SQLファンクションを定義します。

```
PACKAGE imglib IS
  FUNCTION get_image(ikey IN OUT VARCHAR2)
    RETURN Ora_Ffi.pointerType;

  PROCEDURE show_image(idata Ora_Ffi.pointerType,
                       iscale NUMBER);
END;   /* package imglib */
```

パッケージ本体は、次のように定義されます

```
PACKAGE BODY imglib IS
  /* Declare the library and function handles.*/
  imglib_lhandle     Ora_Ffi.libHandleType;
  get_image_fhandle   Ora_Ffi.funcHandleType;
  show_image_fhandle  Ora_Ffi.funcHandleType;

  /* Create the PL/SQL function that will actually  */
  /* invoke the 'get_image' foreign function.      */
  FUNCTION ff_get_image(fhandle Ora_Ffi.funcHandleType,
                        ikey IN OUT VARCHAR2)
    RETURN Ora_Ffi.handleType;
    PRAGMA interface(C, ff_get_image, 11265);
```

```
  /* Create the 'get_image' PL/SQL function that is */
  /* defined in the package spec.                   */
  FUNCTION get_image(ikey IN OUT VARCHAR2)
    RETURN Ora_Ffi.pointerType IS
    ptr Ora_Ffi.pointerType;
  BEGIN
    ptr.handle := ff_get_image(get_image_fhandle, ikey);
    RETURN(ptr);
  END;   /* function get_image */

  /* Create the PL/SQL procedure that will actually */
  /* invoke the 'show_image' foreign function.      */
  PROCEDURE ff_show_image(fhandle Ora_Ffi.funcHandleType,
                      idata Ora_Ffi.handleType,
                      iscale NUMBER);
    PRAGMA interface(C, ff_show_image, 11265);

  /* Create the 'show_image' PL/SQL procedure that is */
  /* defined in the package spec.                     */
  PROCEDURE show_image(idata Ora_Ffi.pointerType,
                    iscale NUMBER) IS
  BEGIN
    ff_show_image(show_image_fhandle, idata.handle, iscale);
  END;   /* procedure show_image */

BEGIN   /* package body imglib */

  /* Load the library.*/
  imglib_lhandle := Ora_Ffi.Load_Library
        ('C:¥oralibs¥', 'imglib.dll');

  /* Register the foreign functions.*/
  get_image_fhandle := Ora_Ffi.Register_Function
        (imglib_lhandle, 'get_image', Ora_Ffi.C_Std);
  show_image_fhandle := Ora_Ffi.Register_Function
        (imglib_lhandle, 'show_image', Ora_Ffi.C_Std);

  /* Register the parameters.*/
  Ora_Ffi.Register_Parameter(get_image_fhandle,
                         Ora_Ffi.C_Char_Ptr);

  Ora_Ffi.Register_Parameter(show_image_fhandle,
                         Ora_Ffi.C_Void_Ptr);
  Ora_Ffi.Register_Parameter(show_image_fhandle,
                         Ora_Ffi.C_Float);

  /* Register the return type ('get_image' only).*/
  Ora_Ffi.Register_Return(get_image_fhandle,
                       Ora_Ffi.C_Void_Ptr);
```

```
END;   /* package body imglib */
```

外部ファンクションを起動するには、次の例に示すように、パッケージ仕様部で定義されたPL/SQLプロシージャをコールします。

```
PROCEDURE display_image(keywrd IN OUT VARCHAR2) IS
  img_ptr  Ora_Ffi.Pointertype;
BEGIN
  img_ptr := imglib.get_image(keywrd);
  imglib.show_image(img_ptr, 2);
END;   /* procedure display_image */
```

ORA_NLSパッケージ

ORA_NLSパッケージ

Ora_Nls.American

説明

現行のキャラクタ・セットが"U.S."ならばTRUE、そうでなければFALSEが戻されます。

構文

```
FUNCTION Ora_Nls.American
RETURN BOOLEAN;
```

戻り値

TRUEまたはFALSE。

Ora_Nls.Americanの例

```
 /*
** Determine if you're dealing with an American
```

```
** set or not
*/
PROCEDURE is_american (out Text_IO.File_Type) IS
  us  BOOLEAN;
BEGIN
  us := Ora_Nls.American;
  IF us = TRUE THEN
    Text_IO.Put (out, 'Character set it American');
  ELSE
    change_char_set;
  END IF;
END;
```

Ora_Nls.American_Date

説明

現行の日付書式が"U.S."ならばTRUEが、そうでなければFALSEが戻されます。

構文

```
FUNCTION Ora_Nls.American_Date
RETURN BOOLEAN;
```

戻り値

TRUEまたはFALSE。

Ora_Nls.American_Dateの例

```
 /*
** Determine if date format is American
*/
PROCEDURE is_amerdate (out Text_IO.File_Type) IS
  usd  BOOLEAN;
BEGIN
  usd := Ora_Nls.American_Date;
  IF usd = TRUE THEN
    Text_IO.Put (out, 'Date format is American');
  ELSE
    change_date_to_us;
  END IF;
END;
```

Ora_Nls.Bad_Attribute

説明

Ora_Nls.Get_Lang_ScalarまたはOra_Nls.Get_Lang_Strに属性を指定しないと呼び出されます。

構文

```
Ora_Nls.Bad_Attribute  EXCEPTION;
```

Ora_Nls.Bad_Attributeの例

```
 /*
** Handle the Bad Attribute exception
*/
EXCEPTION
  WHEN Ora_Nls.Bad_Attribute THEN
    Text_IO.Put_Line('Check calls to Get_Lang_Scalar
      and Get_Lang_Str.A bad attribute name was found.');
```

Ora_Nls.Get_Lang_Scalar

説明

現行言語について要求した情報が戻されます。GET_LANG_SCALARを使用して、数値データを取り出すことができます。

構文

```
FUNCTION Ora_Nls.Get_Lang_Scalar
   (attribute PLS_INTEGER)
RETURN NUMBER;
```

パラメータ

attribute	**ORA_NLS定数またはそれに関連付けられた整数値。定数リストは、「ORA_NLS数値定数」を参照してください。**

戻り値

数値。

Ora_Nls.Get_Lang_Scalarの例

```
 /*
** Retrieve and print out the language number
*/
PROCEDURE lang_num (out Text_IO.File_Type) IS
```

```
  lang_num  NUMBER;
BEGIN
  lang_num := Ora_Nls.Get_Lang_Scalar
    (Ora_Nls.Iso_Alphabet);
  Text_IO.Putf (out, "Current Language numer is %s¥n",
    lang_num);
END;
```

Ora_Nls.Get_Lang_Str

説明

現行言語について要求した情報が戻されます。GET_LANG_STRを使用して、文字情報を取り出すことができます。

構文

```
FUNCTION Ora_Nls.Get_Lang_Str
   (attribute PLS_INTEGER)
RETURN VARCHAR2;
```

パラメータ

attribute	**ORA_NLS定数またはそれに関連付けられた整数値。定数リストは、「ORA_NLS文字定数」を参照してください。**

戻り値

文字値。

Ora_Nls.Get_Lang_Strの例

```
 /*
** Retrieve and print out the language name
*/
PROCEDURE lang_name (out Text_IO.File_Type) IS
  lang_name  VARCHAR2(80);
BEGIN
  lang_name := Ora_Nls.Get_Lang_Str
    (Ora_Nls.Language);
  Text_IO.Putf (out, "Current Language is %s¥n",
    lang_name);
END;
```

Ora_Nls.Linguistic_Collate

説明

特別な言語情報に従って現行キャラクタ・セットの文字を照合する必要があればTRUEが、そうでなければFALSEが戻されます。

構文

```
FUNCTION Ora_Nls.Linguistic_Collate
RETURN BOOLEAN;
```

戻り値

TRUEまたはFALSE。

使用上の注意

TRUEが戻されても、2文字のバイナリ・ソートによって正しい値が戻されるとは限りません。それは、現在使用しているキャラクタ・セットのコード構造が、必ずしもすべての文字を数値の昇順に定義しないからです。

なお、言語によって、文字のソート位置が異なる場合があります。たとえば、ドイツ語では"?"は"b"より前にソートされますが、スウェーデン語では"z"の後ろにソートされます。

Ora_Nls.Linguistic_Collateの例

```
 /*
** Determine whether or not special collating is
** needed.
*/
collate := Ora_Nls.Linguistic_Collate;
  IF collate = TRUE THEN
    lang_name (langinfo.txt);
    Text_IO.Put ('This needs special collating.');
  END IF;
```

Ora_Nls.Linguistic_Specials

説明

言語上の特例を使用している場合はTRUEが、そうでなければFALSEが戻されます。

構文

```
FUNCTION Ora_Nls.Linguistic_Specials
RETURN BOOLEAN;
```

戻り値

TRUEまたはFALSE。

使用上の注意

言語上の特例とは、大/小文字の照合および変換用の言語固有の特殊なケースです。たとえば、ドイツ語のシャープ"s"（1バイト）の大文字は"SS" （2バイト）です。ソートも2バイト値に従って行われます。

言語上の特例は、通常の照合とともに言語定義の中で定義されます。特定の言語ハンドルに対して有効な言語定義に、言語上の特例の定義が含まれていると、言語上の特例を扱うファンクションの出力サイズは、入力文字列のサイズより大きくなることがあります。

Ora_Nls.Linguistic_Specialsの例

```
 /*
** Determine whether or not specials are in use
** and how to deal with them if so
*/
specials := Ora_Nls.Linguistic_Specials;
  IF specials = TRUE THEN
    lang_name (langinfo.txt);
    Text_IO.Put ('Specials are in use.');
  END IF;
```

Ora_Nls.Modified_Date_Fmt

説明

日付書式が変更されている場合はTRUEが、そうでなければFALSEが戻されます。

構文

```
FUNCTION Ora_Nls.Modified_Date_Fmt
RETURN BOOLEAN;
```

戻り値

TRUEまたはFALSE。

Ora_Nls.Modified_Date_Fmtの例

```
 /*
** Determine whether or not the date format has been
** modified
*/
modify := Ora_Nls.Modified_Date_Fmt;
```

```
IF modify = TRUE THEN
  Text_IO.Putf (langinfo.txt, 'The date format
    has been modified.');
END IF;
```

Ora_Nls.No_Item

説明

ユーザーが指定した属性を属性定数リストで見つけられないと呼び出されます。

構文

```
Ora_Nls.No_Item  EXCEPTION;
```

Ora_Nls.No_Itemの例

```
 /*
** Hand the exception for an unidentified attribute constant
*/
EXCEPTION
  WHEN Ora.Nls.No_Item THEN
    Text_IO.Put ('An attribute supplied is not valid.');
```

Ora_Nls.Not_Found

説明

要求した項目を見つけられないと呼び出されます。この状況は、主に、Ora_Nls.Get_Lang_Scalarを使った文字情報の取出しや、Ora_Nls.Get_Lang_Strを使った数値情報の取出しの際に発生します。

構文

```
Ora_Nls.Not_Found  EXCEPTION;
```

Ora_Nls.Not_Foundの例

```
/*
** Hand the exception for an item that was not found
*/
EXCEPTION
  WHEN Ora.Nls.Not_Found THEN
    Text_IO.Put ('The item was not found, check calls to Get_Lang.');
```

Ora_Nls.Right_to_Left

説明

現行言語の書き込み方向が"右から左"の場合はTRUEが、そうでなければFALSEが戻されます。

構文

```
FUNCTION Ora_Nls.Right_To_Left
RETURN BOOLEAN;
```

戻り値

TRUEまたはFALSE。

Ora_Nls.Right_To_Leftの例

```
 /*
** Verify that the language is a right-to-left language
*/
rtl := Ora_Nls.Right_To_Left;
IF rtl = FALSE THEN
  Text_IO.Put (langinfo.txt, 'This is not a right to left
    language.');
END IF;
```

Ora_Nls.Simple_Cs

説明

現行キャラクタ・セットが単純（つまり、1バイト、特殊文字なし、特殊な処理なし）ならばTRUEが、そうでなければFALSEが戻されます。

構文

```
FUNCTION Ora_Nls.Simple_Cs
RETURN BOOLEAN;
```

戻り値

TRUEまたはFALSE。

Ora_Nls.Simple_Csの例

```
 /*
**  Determine if the language is simple or not
*/
```

```
simplecs := Ora_Nls.Simple_Cs;
IF simplecs = TRUE THEN
  lang_name (langinfo.txt);
  Text_IO.Put ('This language uses a simple
    character set.');
ELSE
  lang_name (langinfo.txt);
  Text_IO.Put ('This language uses a complex
    character set.');
END IF;
```

Ora_Nls.Single_Byte

説明

現行キャラクタ・セットのすべての文字を1バイトで表せる場合はTRUEが、そうでなければFALSEが戻されます。

構文

```
FUNCTION Ora_Nls.Single_Byte
RETURN BOOLEAN;
```

戻り値

TRUEまたはFALSE。

Ora_Nls.Single_Byteの例

```
 /*
** Determine if the character set is single or multi-byte
*/
bytes := Ora_Nls.Single_Byte;
IF bytes = FALSE THEN
  lang_name (langinfo.txt);
  Text_IO.Put ('This is a multi-byte character set.');
END IF;
```

ORA_PROFパッケージ

ORA_PROFパッケージ

Ora_Prof.Bad_Timer

説明

別のORA_PROFパッケージ・プロシージャまたはファンクションに無効なタイマー名を渡すと呼び出されます。

構文

```
Ora_Prof.Bad_Timer  EXCEPTION;
```

Ora_Prof.Bad_Timerの例

```
  /*
** Create a timer, start it, run a subprogram,
** stop the timer, then display the time in
** seconds.Destroy the timer when finished.
*/
PROCEDURE timed_proc (test VARCHAR2) IS
  i PLS_INTEGER;
BEGIN
  Ora_Prof.Create_Timer('loop2');
  Ora_Prof.Start_Timer('loop2');
  test;
  Ora_Prof.Stop_Timer('loop2');
  Text_IO.Putf('Loop executed in %s seconds.¥n',
              Ora_Prof.Elapsed_Time('loop2'));
  Ora_Prof.Destroy_Timer('loop2');
EXCEPTION
  WHEN ORA_PROF.BAD_TIMER THEN
```

```
  text_io.put_line('Invalid timer name');

END;
```

Ora_Prof.Create_Timer

説明

名前付きタイマーが割り当てられます。このプロシージャを使用する前に名前付きタイマーを参照すると、エラーになります。

構文

```
PROCEDURE Ora_Prof.Create_Timer
   (timer VARCHAR2);
```

パラメータ

timer	**タイマーの名前。**

Ora_Prof.Create_Timerの例

```
 /*
**Allocate the timer 'LOOPTIME'.
*/
Ora_Prof.Create_Timer('LOOPTIME');
```

Ora_Prof.Destroy_Timer

説明

名前付きタイマーが破棄されます。そのタイマーに関連付けられたメモリーはすべて、その時点で解放されます。このプロシージャを使った後に、名前付きタイマーを参照すると、エラーになります。

構文

```
PROCEDURE Ora_Prof.Destroy_Timer
   (timer VARCHAR2);
```

パラメータ

timer	**タイマーの名前。**

Ora_Prof.Destroy_Timerの例

```
 /*
```

```
**Destroy the timer 'LOOPTIME'.
*/
Ora_Prof.Destroy_Timer('LOOPTIME');
```

Ora_Prof.Elapsed_Time

説明

Ora_Prof.Reset_Timerを最後にコールしてからコード・タイマーに累積された合計経過時間が戻されます。

構文

```
FUNCTION Ora_Prof.Elapsed_Time
   (timer PLS_INTEGER)
RETURN PLS_INTEGER;
```

パラメータ

timer	**タイマーの名前。**

戻り値

コード・タイマーに累積された経過時間（ミリ秒単位）。

Ora_Prof.Elapsed_Timeの例

```
 /*
** Create a timer, start it, run a subprogram,
** stop the timer, then display the time in
** seconds.Destroy the timer when finished.
*/
PROCEDURE timed_proc (test VARCHAR2) IS
  i PLS_INTEGER;
BEGIN
  Ora_Prof.Create_Timer('loop2');
  Ora_Prof.Start_Timer('loop2');
  test;
  Ora_Prof.Stop_Timer('loop2');
  Text_IO.Putf('Loop executed in %s seconds.¥n',
     Ora_Prof.Elapsed_Time('loop2'));
  Ora_Prof.Destroy_Timer('loop2');
END;
```

Ora_Prof.Reset_Timer

説明

タイマーの経過時間を0（ゼロ）にリセットします。

構文

```
PROCEDURE Ora_Prof.Reset_Timer
   (timer VARCHAR2);
```

パラメータ

timer	**タイマーの名前。**

Ora_Prof.Reset_Timerの例

```
 PROCEDURE multi_time IS
  i PLS_INTEGER;
BEGIN
  Ora_Prof.Create_Timer('loop');
   --
   -- First loop...
   --
  Ora_Prof.Start_Timer('loop');
    FOR i IN 1..10 LOOP
      Text_IO.Put_Line('Hello');
    END LOOP;
  Ora_Prof.Stop_Timer('loop');
   --
   -- Second loop...
   --
  Ora_Prof.Start_Timer('loop');
  FOR i IN 1..10 LOOP
    Text_IO.Put_Line('Hello');
  END LOOP;
  Ora_Prof.Stop_Timer('loop');
  Ora_Prof.Destroy_Timer('loop');
END;
```

Ora_Prof.Start_Timer

説明

タイマーが起動されます。Ora_Prof.Timer_Startをコールしてから Ora_Prof.Timer_Stopを呼び出すまでの累積経過時間がタイマーの合計経過時間に加算されます。

構文

```
PROCEDURE Ora_Prof.Start_Timer
   (timer VARCHAR2);
```

パラメータ

timer	タイマーの名前。

Ora_Prof.Start_Timerの例

```
 PROCEDURE multi_time IS
  i PLS_INTEGER;
BEGIN
  Ora_Prof.Create_Timer('loop');
   --
   -- First loop...
  --
  Ora_Prof.Start_Timer('loop');
    FOR i IN 1..10 LOOP
      Text_IO.Put_Line('Hello');
    END LOOP;
  Ora_Prof.Stop_Timer('loop');
   --
   -- Second loop...
   --
  Ora_Prof.Start_Timer('loop');
  FOR i IN 1..10 LOOP
    Text_IO.Put_Line('Hello');
  END LOOP;
  Ora_Prof.Stop_Timer('loop');
  Ora_Prof.Destroy_Timer('loop');
END;
```

Ora_Prof.Stop_Timer

説明

タイマーが停止されます。Ora_Prof.Timer_Startをコールしてから Ora_Prof.Timer_Stopを呼び出すまでの累積経過時間がタイマーの合計経過時間に加算されます。

構文

```
PROCEDURE Ora_Prof.Stop_Timer
   (timer VARCHAR2);
```

パラメータ

timer	タイマーの名前。

Ora_Prof.Stop_Timerの例

```
 PROCEDURE multi_time IS
  i PLS_INTEGER;
BEGIN
  Ora_Prof.Create_Timer('loop');
   --
   -- First loop...
   --
  Ora_Prof.Start_Timer('loop');
    FOR i IN 1..10 LOOP
      Text_IO.Put_Line('Hello');
    END LOOP;
  Ora_Prof.Stop_Timer('loop');
   --
   -- Second loop...
   --
  Ora_Prof.Start_Timer('loop');
  FOR i IN 1..10 LOOP
    Text_IO.Put_Line('Hello');
  END LOOP;
  Ora_Prof.Stop_Timer('loop');
  Ora_Prof.Destroy_Timer('loop');
END;
```

Txet_IOパッケージ

TEXT_IOパッケージ

Text_IO.Fclose

説明

ファイルがクローズされます。

構文

```
PROCEDURE Text_IO.Fclose
   (file file_type);
```

パラメータ

file	クローズするファイルを指定する変数。

Text_IO.Fcloseの例

```
 /*
**  Close the output file.
*/
Text_IO.Fclose (out_file);
```

Text_IO.File_Type

説明

ファイルに対するハンドルが指定されます。

構文

```
TYPE Text_IO.File_Type;
```

Text_IO.File_Typeの例

```
 /*
** Declare a local variable to represent
** the output file you will write to.
*/
out_file  Text_IO.File_Type;
```

Text_IO.Fopen

説明

ファイルを指定したモードでオープンします。

構文

```
FUNCTION Text_IO.Fopen
   (spec     VARCHAR2,
    filemode VARCHAR2)
RETURN Text_IO.File_Type;
```

パラメータ

spec	ファイル名に相当する大/小文字の区別なしの文字列。
filemode	ファイルをオープンするときのモードを指定する、大/小文字の区別なしの1つの文字であり、次の文字のうちの一つになります。 R　読込み専用として、ファイルをオープンします。 W　ファイル内の既存の行をすべて削除した後、読込みおよび書込みを行うために、ファイルをオープンします。 A　ファイル内の既存の行を削除しないで（追加して）、読込みおよび書込みを行うために、ファイルをオープンします。

戻り値

指定したファイルに対するハンドル。

Text_IO.Fopenの例

```
 /*
** Declare two local variables to represent two files:
** one to read from, the other to write to.
*/
in_file   Text_IO.File_Type;
out_file  Text_IO.File_Type;
in_file  := Text_IO.Fopen('salary.txt', 'r');
out_file := Text_IO.Fopen('bonus.txt', 'w');
```

Text_IO.Is_Open

説明

指定したファイルが現在オープンされているかどうかチェックされます。

構文

```
FUNCTION Text_IO.Is_Open
   (file file_type)
RETURN BOOLEAN;
```

パラメータ

file	チェックするファイルを指定する変数。

戻り値

TRUEまたはFALSE。

Text_IO.Is_Openの例

```
 /*
** Determine if the output file is open.If so,
** then close it.
*/
IF Text_IO.Is_Open(out_file) THEN
   Text_IO.Fclose(out_file);
END IF;
```

Text_IO.Get_Line

説明

オープンしているファイルの行を検索し、*item*内に入れます。Text_IO.Get_Lineは、改行文字（キャリッジ・リターン）が読み込まれるか、またはファイルの終わり（EOF）条件が検出されるまで、文字を読み込みます。

読み込む行が*item*のサイズを超えると、Value_Error例外が呼び出されます。ファイル内に読み込む文字がなくなると、No_Data_Found例外が呼び出されます。

構文

```
PROCEDURE Text_IO.Get_Line
  (file file_type,
   item OUT VARCHAR2);
```

パラメータ

file	**オープン・ファイルを指定する変数。**
item	**読み込んだ次の行を格納しておく変数。**

Text_IO.Get_Lineの例

```
/*
** Open a file and read the first line
** into linebuf.
*/
declare
  in_file  Text_IO.File_Type;
  linebuf  VARCHAR2(80);
begin
  in_file := Text_IO.Fopen('salary.txt', 'r');
  Text_IO.Get_Line(in_file,linebuf);
end;
```

Text_IO.New_Line

説明

オープン・ファイルの現行の行に、指定した数の改行文字（キャリッジ・リターン）が連結されるか、インタプリタに出力されます。デフォルト値は1です。つまり、値を指定しない（たとえばText_IO.New_Line）場合、改行文字が1つ作成されます。

構文

```
PROCEDURE Text_IO.New_Line
   (file file_type,
```

```
   n    PLS_INTEGER := 1);
PROCEDURE Text_IO.New_Line
   (n PLS_INTEGER := 1);
```

パラメータ

file	オープン・ファイルを指定する変数。
n	整数。

Text_IO.New_Lineの例

```
 /*
** Write a string to the output file, then
** create a newline after it.
*/
Text_IO.Put(out_file, SYSDATE);
Text_IO.New_Line(out_file, 2);
```

Text_IO.Put

説明

指定したデータがオープンしているファイルの現行の行に連結されるか、インタプリタに出力されます。*item*についてVARCHAR2、DATE、NUMBERおよびPLS_INTEGERの値を受け付けるText_IO.Putプロシージャがいくつかあることに注意してください。すべてのプロシージャ（VARCHAR2を除く）は、指定したデータを文字列に変換します。改行文字（キャリッジ・リターン）は追加されません。

構文

```
PROCEDURE Text_IO.Put
   (file file_type,
    item VARCHAR2);
PROCEDURE Text_IO.Put
   (item VARCHAR2);
PROCEDURE Text_IO.Put
   (item DATE);
PROCEDURE Text_IO.Put
   (file file_type,
    item DATE);
PROCEDURE Text_IO.Put
   (file file_type,
    item NUMBER);
PROCEDURE Text_IO.Put
   (item NUMBER);
PROCEDURE Text_IO.Put
```

```
   (file file_type,
    item PLS_INTEGER);
PROCEDURE Text_IO.Put
   (item PLS_INTEGER);
```

パラメータ

file	**オープン・ファイルを指定する変数。**
item	**バッファとして使用される変数。**

Text_IO.Putの例

```
 /*
** Write a line to a specified output file, create
** a newline, then write another line to the output
** file.
*/
Text_IO.Put(out_file, SYSDATE);
Text_IO.New_Line(out_file);
Text_IO.Put('Processing ends...');
```

Text_IO.Putf

説明

メッセージがフォーマットされてオープン・ファイルに書き込まれるか、メッセージがインタプリタに出力されます。*format*には5個の"%s"パターン（たとえば'%s %s %s'）まで埋め込むことができます。"%s"パターンは、連続した文字*arg*値（たとえばCheck', 'each', 'value'）で置換されます。"¥n"パターンは改行文字（つまりキャリッジ・リターン）で置換されます。

構文

```
PROCEDURE Text_IO.Putf
   (arg    VARCHAR2);
PROCEDURE Text_IO.Putf
   (file   file_type,
    arg    VARCHAR2);
PROCEDURE Text_IO.Putf
   (file   file_type,
    format VARCHAR2,
   [arg1 [,..., arg5] VARCHAR2]);
PROCEDURE Text_IO.Putf
   (format VARCHAR2,
   [arg1 [,..., arg5] VARCHAR2]);
```

パラメータ

arg	表示する値（文字列、変数など）を指定する引数。
format	表示するメッセージの書式を指定します。
file	オープン・ファイルを指定する変数。

使用上の注意

文字以外に置換されたデータが入っているメッセージをフォーマットするには、引数でTO_CHARファンクションを使います（次の例を参照してください）。

Text_IO.Putfの例

```
 /*
** Write a line to the output file, using the
** TO_CHAR(SYSDATE) call to represent the substituted
** character variable.
*/
Text_IO.Putf(out_file,'Today is %s¥n',
  TO_CHAR(SYSDATE));
```

Text_IO.Put_Line

説明

*item*で指定した文字データがオープンしているファイルの現行の行に連結されるか、インタプリタに出力されます。改行文字（キャリッジ・リターン）は、文字列の末尾に自動的に追加されます。

構文

```
PROCEDURE Text_IO.Put_Line
   (file file_type,
    item VARCHAR2);
```

パラメータ

file	オープン・ファイルを指定する変数。
item	表示する文字データを指定する変数。

Text_IO.Put_Lineの例

```
 /*
** Print two complete lines to the output file.
*/
Text_IO.Put_Line(out_file, TO_CHAR(SYSDATE));
Text_IO.Put_Line('Starting test procedures...');
```

TOOL_ENVパッケージ

TOOL_ENVパッケージ

Tool_Env.Getvar

説明

VARCHAR2変数に環境変数をインポートできます。

構文

```
PROCEDURE Tool_Env.Getvar
   (varname  VARCHAR2,
    varvalue VARCHAR2);
```

パラメータ

varname	**環境変数名。**
varvalue	**環境変数の値。**

Tool_Env.Getvarの例

```
 /*
** Retrieve the environment variable USER into a
** variable named :userid so you can use it in a
** connect string or other call.
*/
Tool_Env.Getvar('USER', :userid);
```

TOOL_ERRパッケージ

TOOL_ERRパッケージ

Tool_Err.Clear

説明

エラー・スタック上の現行のすべてのエラーが破棄されます。

構文

```
PROCEDURE Tool_Err.Clear;
```

Tool_Err.Code

説明

エラー・スタックの*i*番目（デフォルトは1番上のエラー）のエラーのエラー・コードが戻されます。スタック上にエラーがない場合、0（ゼロ）が戻されます。

構文

```
FUNCTION Tool_Err.Code
   (i PLS_INTEGER := TOPERROR)
RETURN NUMBER;
```

パラメータ

i	**エラー・スタックにあるエラーを指定する整数。**

戻り値

指定したエラーのエラー・コード。

Tool_Err.Codeの例

```
/*
** Check for unexpected error, disregard it,
** and print any other error.
*/
PROCEDURE check_err IS
BEGIN
  IF (TOOL_ERR.CODE != pkg_a.not_found) THEN
    TEXT_IO.PUT_LINE(TOOL_ERR.MESSAGE);
  END IF;
  TOOL_ERR.POP;
END;
```

Tool_Err.Encode

説明

指定した接頭辞とオフセットをもつ、パッケージ内で使用するための一意のエラー・コードが構成されます。

注意: これはPL/SQL例外ではありません。

構文

```
FUNCTION Tool_Err.Encode
   (prefix VARCHAR2,
    offset PLS_INTEGER)
RETURN NUMBER;
```

パラメータ

prefix	**5文字の文字列。**
offset	**1～127の整数。**

戻り値

エラー・コード。

Tool_Err.Encodeの例

```
/*
** Define a list of errors for a package
** called pkg_a.
*/
```

```
PACKAGE pkg_a IS
  not_found CONSTANT pls_integer := TOOL_ERR.ENCODE('pkg_a', 1);
  bad_value CONSTANT pls_integer := TOOL_ERR.ENCODE('pkg_a', 2);
  too_big   CONSTANT pls_integer := TOOL_ERR.ENCODE('pkg_a', 3);
  too_small CONSTANT pls_integer := TOOL_ERR.ENCODE('pkg_a', 4);
  . . .    /* Rest of pkg_a specification */
END;
```

Tool_Err.Message

説明

エラー・スタックの*i*番目のエラー（デフォルトは1番上のエラー）のフォーマットされたメッセージが戻されます。

構文

```
FUNCTION Tool_Err.Message
   (i PLS_INTEGER := TOPERROR)
RETURN VARCHAR2;
```

パラメータ

i	**エラー・スタックにあるエラーを指定する整数。**

戻り値

エラー・メッセージ。

Tool_Err.Messageの例

```
/*
** Determine the number of errors
** on the stack.Then, loop through stack,
** and print out each error message.
*/
PROCEDURE print_all_errors IS
  number_of_errors   PLS_INTEGER;
BEGIN
EXCEPTION
  WHEN OTHERS THEN
  number_of_errors := TOOL_ERR.NERRORS;
  FOR i IN 1..number_of_errors LOOP
    TEXT_IO.PUT_LINE(TOOL_ERR.MESSAGE(i-1));
  END LOOP;
END;
```

Tool_Err.Nerrors

説明

エラー・スタック上の現行エラーの数が戻されます。

構文

```
FUNCTION Tool_Err.Nerrors
RETURN PLS_INTEGER;
```

戻り値

エラー・スタック上のエラーの数。

Tool_Err.Nerrorsの例

```
/*
** Determine the number of errors
** on the stack.Then, loop through stack,
** and print out each error message.
*/
PROCEDURE print_all_errors IS
  number_of_errors   PLS_INTEGER;
BEGIN
EXCEPTION
  WHEN OTHERS THEN
  number_of_errors := TOOL_ERR.NERRORS;
  FOR i IN 1..number_of_errors LOOP
    TEXT_IO.PUT_LINE(TOOL_ERR.MESSAGE(i-1));
  END LOOP;
END;
```

Tool_Err.Pop

説明

エラー・スタック上の1番上のエラーが破棄されます。

構文

```
PROCEDURE Tool_Err.Pop;
```

Tool_Err.Popの例

```
/*
** Loop through each message in the stack,
```

```
** print it, then clear the top most error.
*/
BEGIN
  . . .

EXCEPTION
  WHEN OTHERS THEN
  FOR i IN 1..Tool_Err.Nerrors LOOP
    TEXT_IO.PUT_LINE(TOOL_ERR.MESSAGE);
    TOOL_ERR.POP;
  END LOOP;
  . . .

END;
```

Tool_Err.Tool_Error

説明

エラー・スタックに1つ以上のエラーをプッシュしたことを示すために使用する一般エラーが定義されます。

構文

```
Tool_Err.Tool_Error  EXCEPTION;
```

Tool_Err.Tool_Errorの例

```
/*
** Raise a generic internal error if a function
** argument is out of range.
*/
PROCEDURE my_proc(count PLS_INTEGER) IS
BEGIN
  IF (count < 0) THEN
    RAISE TOOL_ERR.TOOL_ERROR;
  END IF;
  . . .

END;
```

Tool_Err.Toperror

説明

エラー・スタックの1番上のエラーが識別されます。

構文

```
Tool_Err.Toperror CONSTANT PLS_INTEGER;
```

Tool_Err.Toperrorの例

```
/*
**  Print top-most error on the stack.The same
**  results are produced by calling Tool_Err.Message
**  with no arguments.
*/
BEGIN
  . . .

  TEXT_IO.PUT_LINE(TOOL_ERR.MESSAGE(TOOL_ERR.TOPERROR));
  . . .

END;
```

TOOL_RESパッケージ

TOOL_RESパッケージ

Tool_Res.Bad_File_Handle

説明

Tool_Res.Rfcloseに渡したファイル・ハンドルが無効な場合に呼び出されます。

構文

```
Tool_Res.Bad_File_Handle  EXCEPTION;
```

Tool_Res.Bad_File_Handleの例

```
/*
** This examples uses Tool_Res.Bad_File_Handle
*/
PRODEDURE res_test IS
  resfileh   TOOL_RES.RFHANDLE;
  resfileh1  TOOL_RES.RFHANDLE;
  res1       VARCHAR2(20);
BEGIN
  /*  Open a resource file */
  resfileh := TOOL_RES.RFOPEN('C:¥resource¥test.res');
  ...
  /* Used wrong handle to close
      the resource file.*/
  TOOL_RES.RFCLOSE(resfileh1);
  ...
EXCEPTION
  WHEN TOOL_RES.BAD_FILE_HANDLE THEN
```

```
    /* print error message */
    TEXT_IO.PUT_LINE('Invalid file handle.');
    /* discard the error */
    TOOL_ERR.POP;
END;
```

Tool_Res.Buffer_Overflow

説明

提供されたバッファより長いリソースを取得しようとすると呼び出されます。

構文

```
Tool_Res.Buffer_Overflow  EXCEPTION;
```

Tool_Res.Buffer_Overflowの例

```
/*
** This example uses Tool_Res.Buffer_Overflow
*/
PRODEDURE res_buf_test IS
  resfileh   TOOL_RES.RFHANDLE;
  res1       VARCHAR2(20);
BEGIN
  /*  Open a resource file */
  resfileh := TOOL_RES.RFOPEN
          ('C:¥resource¥test.res');
  /*  Attempt to read very large string
        which overflows buffer.*/
  res1 := TOOL_RES.RFREAD(resfileh,'res_1');
   ...

EXCEPTION
  WHEN TOOL_RES.BUFFER_OVERFLOW THEN
    /* print error message */
    TEXT_IO.PUT_LINE('Buffer overflow.');
    /* discard the error */
    TOOL_ERR.POP;
END;
```

Tool_Res.File_Not_Found

説明

指定したファイルをオープンできないと呼び出されます。 ほとんどの場合、次の理由のいずれかが原因です。

- ファイル名
- ファイルに対する許可
- システム・エラー

構文

```
Tool_Res.File_Not_Found  EXCEPTION;
```

Tool_Res.File_Not_Foundの例

```
/*
** This example uses Tool_Res.File_Not_Found
*/
PRODEDURE res_test IS
  resfileh   TOOL_RES.RFHANDLE;
  res1       VARCHAR2(20);
BEGIN
  /* Open a resource file */
  resfileh := TOOL_RES.RFOPEN
       ('C:¥resource¥twst.res');
  /* File name is misspelled. */
  ...

EXCEPTION
  WHEN TOOL_RES.FILE_NOT_FOUND THEN
    /* print error message */
    TEXT_IO.PUT_LINE('Cannot find the file.');
    /* discard the error */
    TOOL_ERR.POP;
END;
```

Tool_Res.No_Resource

説明

指定したリソースが見つからないと呼び出されます。ファイルを指定した場合、指定したリソースはそのファイル内にはありません。ファイルを指定しなかった場合、そのリソースは、現在オープンしているリソース・ファイル内には存在しません。

構文

```
Tool_Res.No_Resource  EXCEPTION;
```

Tool_Res.No_Resourceの例

```
/*
** This examples uses Tool_Res.No_Resource
*/
PRODEDURE res_test IS
  resfileh   TOOL_RES.RFHANDLE;
  res1       VARCHAR2(20);
BEGIN
  /* Open a resource file */
  resfileh := TOOL_RES.RFOPEN
        ('C:¥resource¥test.res');
  /* Attempt to read nonexistant
     resource from file.*/
  res1 := TOOL_RES.RFREAD(resfileh,'Res_1');
   ...

EXCEPTION
  WHEN TOOL_RES.NO_RESOURCE THEN
    /* print error message */
    TEXT_IO.PUT_LINE('Cannot find the resource.');
    /* discard the error */
    TOOL_ERR.POP;
END;
```

Tool_Res.Rfclose

説明

指定したリソース・ファイルがクローズされます。Tool_Res.Rfopenでオープンしたすべてのファイルは、アプリケーションを終了する前に、Tool_Res.Rfcloseを使用してクローズする必要があります。

構文

```
PROCEDURE Tool_Res.Rfclose
   (file rfhandle);
```

パラメータ

file	クローズするファイル。

使用上の注意

RFCLOSEが呼び出す例外は次のとおりです。

BAD_FILE_HANDLE	ファイル・ハンドルが有効なファイルを指し示していない場合に呼び出されます。
Tool_Err.Tool_Error	内部エラーが検出された場合に呼び出されます。

Tool_Res.Rfcloseの例

```
/*
** This examples uses Tool_Res.Rfclose
*/
PROCEDURE my_cleanup
     (my_file_handle IN OUT TOOL_RES.RFHANDLE)IS
BEGIN
  /* Assign a resource file to 'fhandle.' */
  ...

  /* Close the resource file with the
            handle 'fhandle.'*/
  TOOL_RES.RFCLOSE(my_file_handle);
  ...
END;
```

Tool_Res.Rfhandle

説明

ファイルに対するハンドルが指定されます。

構文

```
TYPE Tool_Res.Rfhandle;
```

Tool_Res.Rfhandleの例

```
/*
** This examples uses Tool_Res.Rfhandle
*/
```

```
PRODEDURE res_test IS
  /* Specify the handle 'resfileh'.*/
  resfileh   TOOL_RES.RFHANDLE;
BEGIN
  /* Assign handle to a resource file */
  resfileh := TOOL_RES.RFOPEN('C:¥test.res');
  ...
END;
```

Tool_Res.Rfopen

説明

指定したファイルがリソース・ファイルとしてオープンされます。

構文

```
FUNCTION Tool_Res.Rfopen
   (spec VARCHAR2)
RETURN rfhandle;
```

パラメータ

spec	**オープンするファイル。 *spec*は大/小文字を区別しません。**

戻り値

指定したファイルに対するハンドル。

使用上の注意

Tool_Res.Rfopenが呼び出す例外は次のとおりです。

File_Not_Found	***spec*に有効なファイルを指していないかファイルをオープンできない場合に呼び出されます。**
Tool_Err.Tool_Error	**内部エラーが検出された場合に呼び出されます。**

Tool_Res.Rfopenの例

```
/*
** This example uses Tool_Res.Rfopen
*/
PROCEDURE my_init
  (fhandle OUT TOOL_RES.RFHANDLE) IS
BEGIN
  /* Open a resource file and assign it to
           the handle, 'fhandle.'*/
  fhandle := TOOL_RES.RFOPEN('C;¥my_app.res');
```

```
   ...
END;
```

Tool_Res.Rfread

説明

指定したリソースが読み込まれます。ファイル・ハンドルを含んでいる場合、指定したリソース・ファイルのみが検索されます。ファイル・ハンドルを含んでいない場合は、現在オープンしているリソース・ファイルをすべて検索します。

構文

```
FUNCTION Tool_Res.Rfread
   (rfile   rfhandle,
    resid   VARCHAR2,
    restype VARCHAR2 := 'string')
RETURN VARCHAR2;

FUNCTION Tool_Res.Rfread
   (resid   VARCHAR2,
    restype VARCHAR2 := 'string')
RETURN VARCHAR2;
```

パラメータ

rfile	読み込むファイル。
resid	リソースのID。
restype	リソースの型。

戻り値

指定したファイルに対するハンドル。

使用上の注意

Rfreadが呼び出す例外は次のとおりです。

No_Resource	指定したリソースが見つからなかった場合に呼び出されます。
Buffer_Overflow	提供される"buffer"が要求したリソースより小さい場合に呼び出されます。
Tool_Err.Tool_Error	内部エラーが検出された場合に呼び出されます。

Tool_Res.Rfreadの例

```
/*
** This examples uses Tool_Res.Rfread
*/
```

```
PRODEDURE ban_res IS
  resfileh   TOOL_RES.RFHANDLE;
  res1       VARCHAR2(20);
BEGIN
  /*  Open a resource file */
  resfileh := TOOL_RES.RFOPEN
        ('C:¥resource¥test.res');
  /*  Read resource string 'banner' from file */
  res1 := TOOL_RES.RFREAD(resfileh,'banner');
  ...

  TEXT_IO.PUT_LINE(res1);
  ...

END;
```

索引

せ

て

ひ

ふ

り

れ