215
DBMS_UTILITY

The `DBMS_UTILITY` package provides various utility subprograms.

This chapter contains the following topics:

DBMS_UTILITY Deprecated Subprograms

These `DBMS_UTILITY` subprograms are deprecated in Oracle Database 12*c* release 12.2.

> **Note:**
>
> Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

- GET_PARAMETER_VALUE Function

  Query `v$_parameter` directly to find the value of an `init.ora` parameter.
- ANALYZE_PART_OBJECT Procedure

  Use `DBMS_STATS` to gather statistics.
- GET_DEPENDENCY Procedure

  There is no replacement for this subprogram. However, you can directly query the dictionary views.

There are no replacements for these subprograms.

DBMS_UTILITY Security Model

`DBMS_UTILITY` runs with the privileges of the calling user for the `NAME_RESOLVE` procedure and the `COMPILE_SCHEMA` procedure. This is necessary so that the SQL works correctly.

The package does not run as `SYS`. The privileges are checked using `DBMS_DDL`.

Related Topics

- NAME_RESOLVE Procedure
  This procedure resolves the given name, including synonym translation and authorization checking as necessary.
- COMPILE_SCHEMA Procedure
  This procedure compiles all procedures, functions, packages, views and triggers in the specified schema.

DBMS_UTILITY Constants

The `DBMS_UTILITY` package defines one constant to use when specifying parameter values.

This constant is shown in the following table.

Table 215-1 DBMS_UTILITY Constants

Name	Type	Value	Description
`INV_ERROR_ON_RESTRICTIONS`	`PLS_INTEGER`	`1`	This constant is the only legal value for the `p_option_flags` parameter of the `INVALIDATE` subprogram

DBMS_UTILITY Exceptions

This table lists the exceptions raised by `DBMS_UTILITY`.

Table 215-2 Exceptions Raised by DBMS_UTILITY

Exception	Error Code	Description
`INV_NOT_EXIST_OR_NO_PRIV`	-24237	Raised by the `INVALIDATE` subprogram when the `object_id` argument is `NULL` or invalid, or when the caller does not have `CREATE` privileges on the object being invalidated
`INV_MALFORMED_SETTINGS`	-24238	Raised by the `INVALIDATE` subprogram if a compiler setting is specified more than once in the `p_plsql_object_settings` parameter
`INV_RESTRICTED_OBJECT`	-24239	Raised by the `INVALIDATE` subprogram when different combinations of conditions pertaining to the `p_object_id` parameter are contravened

DBMS_UTILITY Data Structures

The `DBMS_UTILITY` package defines a single `RECORD` type and `TABLE` types.

Record Types

- INSTANCE_RECORD Record Type

Table Types

- DBLINK_ARRAY TABLE Type

DBMS_UTILITY INSTANCE_RECORD Record Type

This type describes a list of active instance number-name pairs.

Syntax

```
TYPE INSTANCE_RECORD IS RECORD (
     inst_number   NUMBER,
     inst_name     VARCHAR2(60));
```

Fields

Table 215-3 INSTANCE_RECORD Record Type Fields

Field	Description
inst_number	Active instance number
inst_name	Instance name

DBMS_UTILITY DBLINK_ARRAY TABLE Type

This type stores a list of database links.

Syntax

```
TYPE DBLINK_ARRAY IS TABLE OF VARCHAR2(128) INDEX BY BINARY_INTEGER;
```

DBMS_UTILITY INDEX_TABLE_TYPE Table Type

This type describes the order in which generated objects are returned to a user.

Syntax

```
TYPE INDEX_TABLE_TYPE IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

DBMS_UTILITY INSTANCE_TABLE Table Type

This type describes a table of `INSTANCE_RECORD` Record Type.

Syntax

```
TYPE INSTANCE_TABLE IS TABLE OF INSTANCE_RECORD INDEX BY BINARY_INTEGER;
```

Usage Notes

The starting index of `INSTANCE_TABLE` Is 1; `INSTANCE_TABLE` Is Dense.

Related Topics

- DBMS_UTILITY INSTANCE_RECORD Record Type
  This type describes a list of active instance number-name pairs.

DBMS_UTILITY LNAME_ARRAY Table Type

This type stores lists of `LONG NAME` including fully qualified attribute names.

Syntax

```
TYPE LNAME_ARRAY IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

DBMS_UTILITY NAME_ARRAY Table Type

This type stores lists of `NAME`.

Syntax

```
TYPE NAME_ARRAY IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```

DBMS_UTILITY NUMBER_ARRAY Table Type

This type describes the order in which generated objects are returned to users.

Syntax

```
TYPE NUMBER_ARRAY IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

DBMS_UTILITY UNCL_ARRAY Table Type

This type stores lists of `"user"."name"."column"@link`

Syntax

```
TYPE UNCL_ARRAY IS TABLE OF VARCHAR2(227) INDEX BY BINARY_INTEGER;
```

Summary of DBMS_UTILITY Subprograms

This table lists the `DBMS_UTILITY` subprograms and briefly describes them.

Table 215-4 *DBMS_UTILITY Package Subprograms*

Subprogram	Description
ACTIVE_INSTANCES Procedure	Returns the active instance
ANALYZE_DATABASE Procedure	Analyzes all the tables, clusters and indexes in a database
ANALYZE_PART_OBJECT Procedure	Analyzes the given tables and indexes This procedure is deprecated from the `DBMS_UTILITY` package with Oracle Database 12c release 12.2 and later. Use `DBMS_STATS` to gather statistics.
ANALYZE_SCHEMA Procedure	Analyzes all the tables, clusters and indexes in a schema
CANONICALIZE Procedure	Canonicalizes a given string

Table 215-4 (Cont.) *DBMS_UTILITY Package Subprograms*

Subprogram	Description
COMMA_TO_TABLE Procedures	Converts a comma-delimited list of names into a PL/SQL table of names
COMPILE_SCHEMA Procedure	Compiles all procedures, functions, packages, views and triggers in the specified schema
CREATE_ALTER_TYPE_ERROR_TABLE Procedure	Creates an error table to be used in the `EXCEPTION` clause of the `ALTER TYPE` statement
CURRENT_INSTANCE Function	Returns the current connected instance number
DATA_BLOCK_ADDRESS_BLOCK Function	Gets the block number part of a data block address
DATA_BLOCK_ADDRESS_FILE Function	Gets the file number part of a data block address
DB_VERSION Procedure	Returns version information for the database
EXEC_DDL_STATEMENT Procedure	Executes the DDL statement in `parse_string`
EXPAND_SQL_TEXT Procedure	Recursively replaces any view references in the input SQL query with the corresponding view subquery
FORMAT_CALL_STACK Function	Formats the current call stack
FORMAT_ERROR_BACKTRACE Function	Formats the backtrace from the point of the current error to the exception handler where the error has been caught
FORMAT_ERROR_STACK Function	Formats the current error stack
GET_CPU_TIME Function	Returns the current CPU time in 100th's of a second
GET_DEPENDENCY Procedure	Shows the dependencies on the object passed in. This procedure is deprecated from the `DBMS_UTILITY` package with Oracle Database 12c release 12.2 and later. There is no replacement for this subprogram.
GET_ENDIANNESS Function	Gets the endianness of the database platform
GET_HASH_VALUE Function	Computes a hash value for the given string
GET_PARAMETER_VALUE Function	Gets the value of specified init.ora parameter. This function is deprecated from the `DBMS_UTILITY` package with Oracle Database 12c release 12.2 and later. You can query `v$_parameter` directly.
GET_SQL_HASH Function	Computes a hash value for the given string using MD5 algorithm
GET_TIME Function	Returns the current time in 100th's of a second
GET_TZ_TRANSITIONS Procedure	Returns time zeone transitions by `regionid` from the `timezone.dat` file
INVALIDATE Procedure	Invalidates a database object and (optionally) modifies its PL/SQL compiler parameter settings
IS_BIT_SET Function	Checks the bit setting for the given bit in the given `RAW` value
IS_CLUSTER_DATABASE Function	Determines if the database is running in cluster database mode
MAKE_DATA_BLOCK_ADDRESS Function	Creates a data block address given a file number and a block number
NAME_RESOLVE Procedure	Resolves the given name
NAME_TOKENIZE Procedure	Calls the parser to parse the given name

Table 215-4 (Cont.) *DBMS_UTILITY Package Subprograms*

Subprogram	Description
OLD_CURRENT_SCHEMA Function	Returns the session value from `SYS_CONTEXT ('USERENV', 'CURRENT_SCHEMA')`
OLD_CURRENT_USER Function	Returns the session value from `SYS_CONTEXT ('USERENV', 'CURRENT_USER')`
PORT_STRING Function	Returns a string that uniquely identifies the version of Oracle and the operating system
SQLID_TO_SQLHASH Function	Converts a SQL ID into a hash value
TABLE_TO_COMMA Procedures	Converts a PL/SQL table of names into a comma-delimited list of names
VALIDATE Procedure	Makes invalid database objects valid
WAIT_ON_PENDING_DML Function	Waits until all transactions (other than the caller's own) that have locks on the listed tables and began prior to the specified SCN have either committed or been rolled back

ACTIVE_INSTANCES Procedure

This procedure returns the active instance.

Syntax

```
DBMS_UTILITY.ACTIVE_INSTANCES (
   instance_table   OUT INSTANCE_TABLE,
   instance_count   OUT NUMBER);
```

Parameters

Table 215-5 ACTIVE_INSTANCES Procedure Parameters

Procedure	Description
`instance_table`	Contains a list of the active instance numbers and names. When no instance is up, the list is empty.
`instance_count`	Number of active instances

ANALYZE_DATABASE Procedure

This procedure analyzes all the tables, clusters and indexes in a database.

Syntax

```
DBMS_UTILITY.ANALYZE_DATABASE (
   method             IN  VARCHAR2,
   estimate_rows      IN  NUMBER DEFAULT NULL,
   estimate_percent   IN  NUMBER DEFAULT NULL,
   method_opt         IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 215-6 ANALYZE_DATABASE Procedure Parameters

Parameter	Description
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE then either estimate_rows or estimate_percent must be nonzero.
estimate_rows	Number of rows to estimate
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified ignore this parameter.
method_opt	Method options of the following format: [FOR TABLE] [FOR ALL [INDEXED] COLUMNS] [SIZE n] [FOR ALL INDEXES]

Exceptions

ORA-20000: Insufficient privileges for some object in this database

ANALYZE_PART_OBJECT Procedure

This procedure is equivalent to SQL: "ANALYZE TABLE|INDEX [<schema>.]<object_name> PARTITION <pname> [<command_type>] [<command_opt>] [<sample_clause>]

> **Note:**
>
> This subprogram has been deprecated and replaced by improved technology. It is maintained only for purposes of backward compatibility. As an alternative, you can use DBMS_STATS to gather statistics.

Syntax

```
DBMS_UTILITY.ANALYZE_PART_OBJECT (
   schema        IN VARCHAR2 DEFAULT NULL,
   object_name   IN VARCHAR2 DEFAULT NULL,
   object_type   IN CHAR     DEFAULT 'T',
   command_type  IN CHAR     DEFAULT 'E',
   command_opt   IN VARCHAR2 DEFAULT NULL,
   sample_clause IN VARCHAR2 DEFAULT 'sample 5 percent ');
```

Parameters

Table 215-7 ANALYZE_PART_OBJECT Procedure Parameters

Parameter	Description
schema	Schema of the object_name
object_name	Name of object to be analyzed, must be partitioned
object_type	Type of object, must be T (table) or I (index)

Table 215-7 (Cont.) ANALYZE_PART_OBJECT Procedure Parameters

Parameter	Description
command_type	Must be V (validate structure)
command_opt	Other options for the command type. For C, E it can be FOR table, FOR all LOCAL indexes, FOR all columns or combination of some of the 'for' options of analyze statistics (table). For V, it can be CASCADE when object_type is T.
sample_clause	Sample clause to use when command_type is 'E'

Usage Notes

For each partition of the object, run in parallel using job queues.

ANALYZE_SCHEMA Procedure

This procedure analyzes all the tables, clusters and indexes in a schema.

Syntax

```
DBMS_UTILITY.ANALYZE_SCHEMA (
   schema             IN  VARCHAR2,
   method             IN  VARCHAR2,
   estimate_rows      IN  NUMBER DEFAULT NULL,
   estimate_percent   IN  NUMBER DEFAULT NULL,
   method_opt         IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 215-8 ANALYZE_SCHEMA Procedure Parameters

Parameter	Description
schema	Name of the schema
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE then either estimate_rows or estimate_percent must be nonzero.
estimate_rows	Number of rows to estimate
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified ignore this parameter.
method_opt	Method options of the following format: [FOR TABLE] [FOR ALL [INDEXED] COLUMNS] [SIZE n] [FOR ALL INDEXES]

Exceptions

ORA-20000: Insufficient privileges for some object in this schema

CANONICALIZE Procedure

This procedure canonicalizes the given string. The procedure handles a single reserved or key word (such as 'table'), and strips off white spaces for a single identifier so that ' table ' becomes TABLE.

Syntax

```
DBMS_UTILITY.CANONICALIZE(
   name           IN    VARCHAR2,
   canon_name     OUT   VARCHAR2,
   canon_len      IN    BINARY_INTEGER);
```

Parameters

Table 215-9 CANONICALIZE Procedure Parameters

Parameter	Description
`name`	String to be canonicalized
`canon_name`	Canonicalized string
`canon_len`	Length of the string (in bytes) to canonicalize

Return Values

Returns the first `canon_len` bytes in `canon_name`.

Usage Notes

- If `name` is NULL, `canon_name` becomes NULL.
- If `name` is not a dotted name, and if `name` begins and ends with a double quote, remove both quotes. Alternatively, convert to upper case with NLS_UPPER. Note that this case does not include a name with special characters, such as a space, but is not doubly quoted.
- If name is a dotted name (such as a."b".c), for each component in the dotted name in the case in which the component begins and ends with a double quote, no transformation will be performed on this component. Alternatively, convert to upper case with NLS_UPPER and apply begin and end double quotes to the capitalized form of this component. In such a case, each canonicalized component will be concatenated together in the input position, separated by ".".
- Any other character after a[.b]* will be ignored.
- The procedure does not handle cases like 'A B.'

Examples

- `a` becomes `A`
- `"a"` becomes `a`
- `"a".b` becomes `"a"."B"`
- `"a".b,c.f` becomes `"a"."B"` with`",c.f"` ignored.

COMMA_TO_TABLE Procedures

These procedures convert a comma-delimited list of names into a PL/SQL table of names. The second version supports fully-qualified attribute names.

Syntax

```
DBMS_UTILITY.COMMA_TO_TABLE (
   list   IN  VARCHAR2,
   tablen OUT BINARY_INTEGER,
   tab    OUT uncl_array);

DBMS_UTILITY.COMMA_TO_TABLE (
   list   IN  VARCHAR2,
   tablen OUT BINARY_INTEGER,
   tab    OUT lname_array);
```

Parameters

Table 215-10 COMMA_TO_TABLE Procedure Parameters

Parameter	Description
`list`	Comma separated list of list of 'names', where a name should have the following format for the first overloading: `a [. b [. c ]][ @ d ]` and the following format for the second overloading: `a [. b]*` where `a`, `b`, `c`, `d` are simple identifiers (quoted or unquoted).
`tablen`	Number of tables in the PL/SQL table
`tab`	PL/SQL table which contains list of names

Return Values

A PL/SQL table is returned, with values `1..n` and `n+1 is null`.

Usage Notes

- The `list` must be a non-empty comma-delimited list: Anything other than a comma-delimited list is rejected. Commas inside double quotes do not count.
- Entries in the comma-delimited list cannot include multibyte characters.
- The values in `tab` are copied from the original list, with no transformations.
- The procedure fails if the string between separators is longer than 30 bytes.

COMPILE_SCHEMA Procedure

This procedure compiles all procedures, functions, packages, views and triggers in the specified schema.

Syntax

```
DBMS_UTILITY.COMPILE_SCHEMA (
   schema          IN VARCHAR2,
   compile_all     IN BOOLEAN DEFAULT TRUE,
   reuse_settings  IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 215-11 COMPILE_SCHEMA Procedure Parameters

Parameter	Description
schema	Name of the schema
compile_all	If TRUE, will compile everything within the schema regardless of whether it is VALID If FALSE, will compile only INVALID objects
reuse_settings	Indicates whether the session settings in the objects should be reused, or whether the current session settings should be adopted instead

Exceptions

Table 215-12 COMPILE_SCHEMA Procedure Exceptions

Exception	Description
ORA-20000	Insufficient privileges for some object in this schema
ORA-20001	Cannot recompile SYS objects
ORA-20002	Maximum iterations exceeded. Some objects may not have been recompiled.

Usage Notes

- Note that this subprogram is a wrapper for the RECOMP_SERIAL Procedure included with the UTL_RECOMP package.
- After calling this procedure, you should select from view ALL_OBJECTS for items with status of INVALID to see if all objects were successfully compiled.
- To see the errors associated with INVALID objects, you may use the Enterprise Manager command:

```
SHOW ERRORS <type> <schema>.<name>
```

CREATE_ALTER_TYPE_ERROR_TABLE Procedure

This procedure creates an error table to be used in the EXCEPTION clause of the ALTER TYPE statement.

Syntax

```
DBMS_UTILITY.CREATE_ALTER_TYPE_ERROR_TABLE(
   schema_name      IN     VARCHAR2,
   table_name       IN     VARCHAR2);
```

Parameters

Table 215-13 CREATE_ALTER_TYPE_ERROR_TABLE Procedure Parameters

Parameter	Description
schema_name	Name of the schema
table_name	Name of the table created

Exceptions

An error is returned if the table already exists.

CURRENT_INSTANCE Function

This function returns the current connected instance number. It returns NULL when connected instance is down.

Syntax

```
DBMS_UTILITY.CURRENT_INSTANCE
   RETURN NUMBER;
```

DATA_BLOCK_ADDRESS_BLOCK Function

This function gets the block number part of a data block address.

Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_BLOCK (
   dba NUMBER)
  RETURN NUMBER;
```

Parameters

Table 215-14 DATA_BLOCK_ADDRESS_BLOCK Function Parameters

Parameter	Description
dba	Data block address

Pragmas

```
pragma restrict_references(data_block_address_block, WNDS, RNDS, WNPS, RNPS);
```

Return Values

Block offset of the block.

Usage Notes

This function should not be used with datablocks which belong to bigfile tablespaces.

DATA_BLOCK_ADDRESS_FILE Function

This function gets the file number part of a data block address.

Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_FILE (
   dba NUMBER)
  RETURN NUMBER;
```

Parameters

Table 215-15 DATA_BLOCK_ADDRESS_FILE Function Parameters

Parameter	Description
dba	Data block address

Pragmas

```
pragma restrict_references (data_block_address_file, WNDS, RNDS, WNPS, RNPS);
```

Return Values

File that contains the block.

Usage Notes

This function should not be used with datablocks which belong to bigfile tablespaces.

DB_VERSION Procedure

This procedure returns version information for the database.

Syntax

```
DBMS_UTILITY.DB_VERSION (
   version       OUT VARCHAR2,
   compatibility OUT VARCHAR2);
```

Parameters

Table 215-16 DB_VERSION Procedure Parameters

Parameter	Description
version	A string which represents the internal software version of the database (for example, 7.1.0.0.0). The length of this string is variable and is determined by the database version.
compatibility	The compatibility setting of the database determined by the "compatible" init.ora parameter. If the parameter is not specified in the init.ora file, then NULL is returned.

EXEC_DDL_STATEMENT Procedure

This procedure executes the DDL statement in `parse_string`.

Syntax

```
DBMS_UTILITY.EXEC_DDL_STATEMENT (
   parse_string IN VARCHAR2);
```

Parameters

Table 215-17 EXEC_DDL_STATEMENT Procedure Parameters

Parameter	Description
`parse_string`	DDL statement to be executed

EXPAND_SQL_TEXT Procedure

This procedure recursively replaces any view references in the input SQL query with the corresponding view subquery.

Syntax

```
DBMS_UTILITY.EXPAND_SQL_TEXT (
   input_sql_text     IN           CLOB,
   output_sql_text    OUT NOCOPY   CLOB);
```

Parameters

Table 215-18 EXPAND_SQL_TEXT Procedure Parameters

Parameter	Description
`input_sql_text`	Input SQL query text
`output_sql_text`	View-expanded query text

Exceptions

Table 215-19 EXPAND_SQL_TEXT Procedure Exceptions

Exception	Description
`ORA-00942`	Current user does not have select privileges on all the views and tables recursively referenced in the `input_sql_text`
`ORA-24251`	`input_sql_text` is not a `SELECT` statement
`ORA-00900`	Input is not valid
`ORA-29477`	Input LOB size exceeds maximum size of 4GB -1

Usage Notes

The expanded and merged SQL statement text is copied to output_sql_text on successful completion. The resulting query text only contains references to underlying tables and is semantically equivalent with some caveats:

- If there are invoker rights functions called from any of the views, they may be called as a different user in the resulting query text if the view owner is different from the user who will eventually compile/run the expanded SQL text.
- The VPD policy expands differently if there is a function supplied to generate the dynamic `WHERE` clause. This function would return differently, for example, if the userid caused the expansion to be different.
- If there are references to remote objects, results are undetermined.

FORMAT_CALL_STACK Function

This function formats the current call stack. This can be used on any stored procedure or trigger to access the call stack. This can be useful for debugging.

Syntax

```
DBMS_UTILITY.FORMAT_CALL_STACK
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(format_call_stack,WNDS);
```

Return Values

This returns the call stack, up to 2000 bytes.

FORMAT_ERROR_BACKTRACE Function

This function displays the call stack at the point where an exception was raised, even if the subprogram is called from an exception handler in an outer scope.

The output is similar to the output of the `SQLERRM` function, but not subject to the same size limitation.

Syntax

```
DBMS_UTILITY.FORMAT_ERROR_BACKTRACE
  RETURN VARCHAR2;
```

Return Values

The backtrace string. A `NULL` string is returned if no error is currently being handled.

Examples

```
CREATE OR REPLACE PROCEDURE Log_Errors ( i_buff in varchar2 ) IS
  g_start_pos integer := 1;
  g_end_pos  integer;

  FUNCTION Output_One_Line RETURN BOOLEAN IS
  BEGIN
```

```
    g_end_pos := Instr ( i_buff, Chr(10), g_start_pos );

    CASE g_end_pos > 0
      WHEN true THEN
        DBMS_OUTPUT.PUT_LINE ( Substr ( i_buff, g_start_pos,
g_end_pos-g_start_pos ) );
        g_start_pos := g_end_pos+1;
        RETURN TRUE;

      WHEN FALSE THEN
        DBMS_OUTPUT.PUT_LINE ( Substr ( i_buff, g_start_pos,
(Length(i_buff)-g_start_pos)+1 ) );
        RETURN FALSE;
    END CASE;
  END Output_One_Line;

BEGIN
  WHILE Output_One_Line() LOOP NULL;
  END LOOP;
END Log_Errors;
/

Set Doc Off
Set Feedback off
Set Echo Off

CREATE OR REPLACE PROCEDURE P0 IS
  e_01476 EXCEPTION; pragma exception_init ( e_01476, -1476 );
BEGIN
  RAISE e_01476;
END P0;
/
Show Errors

CREATE OR REPLACE PROCEDURE P1 IS
BEGIN
  P0();
END P1;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P2 IS
BEGIN
  P1();
END P2;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P3 IS
BEGIN
  P2();
END P3;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P4 IS
  BEGIN P3(); END P4;
/
CREATE OR REPLACE PROCEDURE P5 IS
  BEGIN P4(); END P5;
/
SHOW ERRORS
```

```
CREATE OR REPLACE PROCEDURE Top_Naive IS
BEGIN
  P5();
END Top_Naive;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE Top_With_Logging IS
  -- NOTE: SqlErrm in principle gives the same info as Format_Error_Stack.
  -- But SqlErrm is subject to some length limits,
  -- while Format_Error_Stack is not.
BEGIN
  P5();
EXCEPTION
  WHEN OTHERS THEN
    Log_Errors ( 'Error_Stack...' || Chr(10) ||
      DBMS_UTILITY.FORMAT_ERROR_STACK() );
    Log_Errors ( 'Error_Backtrace...' || Chr(10) ||
      DBMS_UTILITY.FORMAT_ERROR_BACKTRACE() );
    DBMS_OUTPUT.PUT_LINE ( '----------' );
END Top_With_Logging;
/
SHOW ERRORS

--------------------------------------------------------------------------------

Set ServerOutput On
call Top_Naive()
  /*
  ERROR at line 1:
  ORA-01476: divisor is equal to zero
  ORA-06512: at "U.P0", line 4
  ORA-06512: at "U.P1", line 3
  ORA-06512: at "U.P2", line 3
  ORA-06512: at "U.P3", line 3
  ORA-06512: at "U.P4", line 2
  ORA-06512: at "U.P5", line 2
  ORA-06512: at "U.TOP_NAIVE", line 3
  */
  ;

Set ServerOutput On
call Top_With_Logging()
  /*
  Error_Stack...
  ORA-01476: divisor is equal to zero
  Error_Backtrace...
  ORA-06512: at "U.P0", line 4
  ORA-06512: at "U.P1", line 3
  ORA-06512: at "U.P2", line 3
  ORA-06512: at "U.P3", line 3
  ORA-06512: at "U.P4", line 2
  ORA-06512: at "U.P5", line 2
  ORA-06512: at "U.TOP_WITH_LOGGING", line 6
  ----------
  */
  ;

/*
  ORA-06512:
  Cause:
```

```
    Backtrace message as the stack is
    unwound by unhandled exceptions.
  Action:
    Fix the problem causing the exception
    or write an exception handler for this condition.
    Or you may need to contact your application administrator
    or database administrator.
*/
```

FORMAT_ERROR_STACK Function

This function formats the current error stack. This can be used in exception handlers to look at the full error stack.

Syntax

```
DBMS_UTILITY.FORMAT_ERROR_STACK
  RETURN VARCHAR2;
```

Return Values

This returns the error stack, up to 2000 bytes.

GET_CPU_TIME Function

This function returns a measure of current CPU processing time in hundredths of a second. The difference between the times returned from two calls measures the CPU processing time (not the total elapsed time) between those two points.

Syntax

```
  DBMS_UTILITY.GET_CPU_TIME
   RETURN NUMBER;
```

Return Values

Time is the number of 100th's of a second from some arbitrary epoch.

Usage Notes

The amount of work performed is calculated by measuring the difference between a start point and end point for a particular operation.

GET_DEPENDENCY Procedure

This **deprecated procedure** shows the dependencies on the object passed in.

> **Note:**
>
> This subprogram has been deprecated and replaced in Oracle Database 12c release 12.2 and later. Oracle recommends that you do not use deprecated subprograms. It is maintained only for purposes of backward compatibility.

Syntax

```
DBMS_UTILITY.GET_DEPENDENCY
 type      IN     VARCHAR2,
 schema    IN     VARCHAR2,
 name      IN     VARCHAR2);
```

Parameters

Table 215-20 GET_DEPENDENCY Procedure Parameters

Parameter	Description
type	Type of the object, for example if the object is a table give the type as 'TABLE'
schema	Schema name of the object
name	Name of the object

Usage Notes

This procedure uses the DBMS_OUTPUT package to display results, and so you must declare SET SERVEROUTPUT ON if you wish to view dependencies. Alternatively, any application that checks the DBMS_OUTPUT output buffers can invoke this subprogram and then retrieve the output by means of DBMS_OUTPUT subprograms such as GET_LINES.

GET_ENDIANNESS Function

This function gets the endianness of the database platform.

Syntax

```
DBMS_UTILITY.GET_ENDIANNESS
 RETURN NUMBER;
```

Return Values

A NUMBER value indicating the endianness of the database platform: 1 for big-endian or 2 for little-endian.

GET_HASH_VALUE Function

This function computes a hash value for the given string.

Syntax

```
DBMS_UTILITY.GET_HASH_VALUE (
   name      VARCHAR2,
   base      NUMBER,
   hash_size NUMBER)
  RETURN NUMBER;
```

Parameters

Table 215-21 GET_HASH_VALUE Function Parameters

Parameter	Description
name	String to be hashed.
base	Base value for the returned hash value at which to start
hash_size	Desired size of the hash table

Pragmas

```
pragma restrict_references(get_hash_value, WNDS, RNDS, WNPS, RNPS);
```

Return Values

A hash value based on the input string. For example, to get a hash value on a string where the hash value should be between 1000 and 3047, use 1000 as the base value and 2048 as the hash_size value. Using a power of 2 for the hash_size parameter works best.

GET_PARAMETER_VALUE Function

This deprecated function gets the value of specified init.ora parameter.

> **Note:**
>
> This subprogram has been deprecated and replaced by improved technology. It is maintained only for purposes of backward compatibility. As an alternative, you can query v$_parameter directly.

Syntax

```
DBMS_UTILITY.GET_PARAMETER_VALUE (
   parnam     IN        VARCHAR2,
   intval     IN OUT    BINARY_INTEGER,
   strval     IN OUT    VARCHAR2,
   listno     IN        BINARY_INTEGER DEFAULT 1)
  RETURN BINARY_INTEGER;
```

Parameters

Table 215-22 GET_PARAMETER_VALUE Function Parameters

Parameter	Description
parnam	Parameter name
intval	Value of an integer parameter or the value length of a string parameter
strval	Value of a string parameter
listno	List item number. If retrieving parameter values for a parameter that can be specified multiple times to accumulate values, use this parameter to get each individual parameter.

Return Values

Parameter type:

- 0 if parameter is an `INTEGER`/`BOOLEAN` parameter
- 1 if parameter is a string/file parameter

Usage Notes

- To execute the this function, you must have the `SELECT` privilege on the V$PARAMETER dynamic view.

Examples

```
DECLARE
  parnam VARCHAR2(256);
  intval BINARY_INTEGER;
  strval VARCHAR2(256);
  partyp BINARY_INTEGER;
BEGIN
  partyp := dbms_utility.get_parameter_value('max_dump_file_size',
                                             intval, strval);
  dbms_output.put('parameter value is: ');
  IF partyp = 1 THEN
    dbms_output.put_line(strval);
  ELSE
    dbms_output.put_line(intval);
  END IF;
  IF partyp = 1 THEN
    dbms_output.put('parameter value length is: ');
    dbms_output.put_line(intval);
  END IF;
  dbms_output.put('parameter type is: ');
  IF partyp = 1 THEN
    dbms_output.put_line('string');
  ELSE
    dbms_output.put_line('integer');
  END IF;
END;
```

GET_SQL_HASH Function

This function computes a hash value for the given string using MD5 algorithm.

Syntax

```
Dbms_utility.get_sql_hash (
   name          IN   VARCHAR2,
   hash          OUT  RAW,
   pre10ihash    OUT  NUMBER)
  RETURN NUMBER;
```

Pragmas

Pragma Restrict_references(`Get_sql_hash`, `Wnds`, `Rnds`, `Wnps`, `Rnps`);

Parameters

Table 215-23 GET_SQL_HASH Procedure Parameters

Parameter	Description
`name`	String to be hashed
`hash`	Stores all 16 bytes of returned hash value
`pre10ihash`	Stores the pre 10i database version hash value

Return Values

A hash value (last 4 bytes) based on the input string. the MD5 hash algorithm computes a 16 byte hash value, but we only return the last 4 bytes so that we can return an actual number. Use the `hash` parameter to get all 16 bytes and pre10*i* hash parameter to store the pre 10*i* hash value of 4 bytes.

GET_TIME Function

This function determines the current time in hundredths of a second. This subprogram is primarily used for determining elapsed time. The subprogram is called twice – at the beginning and end of some process – and then the first (earlier) number is subtracted from the second (later) number to determine the time elapsed.

Syntax

```
DBMS_UTILITY.GET_TIME
  RETURN NUMBER;
```

Return Values

Time is the number of hundredths of a second from the point in time at which the subprogram is invoked.

Usage Notes

Numbers are returned in the range -2147483648 to 2147483647 depending on platform and machine, and your application must take the sign of the number into account in determining the interval. For instance, in the case of two negative numbers, application logic must allow that the first (earlier) number will be larger than the second (later) number which is closer to zero. By the same token, your application should also allow that the first (earlier) number be negative and the second (later) number be positive.

GET_TZ_TRANSITIONS Procedure

This procedure returns time zone transitions by `regionid` from the `timezone.dat` file.

Syntax

```
DBMS_UTILITY.GET_TZ_TRANSITIONS
   regionid      IN     NUMBER,
   transitions   OUT    MAXRAW);
```

Parameters

Table 215-24 GET_TZ_TRANSITIONS Procedure Parameters

Parameter	Description
regionid	Number corresponding to the region
transitions	Raw bytes from the timezone.dat file

Exceptions

Table 215-25 GET_TZ_TRANSITIONS Procedure Exceptions

Exception	Description
ORA-6502: PL/SQL: NUMERIC OR VALUE ERROR	For an invalid regionid

INVALIDATE Procedure

This procedure invalidates a database object and (optionally) modifies its PL/SQL compiler parameter settings. It also invalidates any objects that (directly or indirectly) depend on the object being invalidated.

Syntax

```
DBMS_UTILITY.INVALIDATE (
   p_object_id              NUMBER,
   p_plsql_object_settings  VARCHAR2 DEFAULT NULL,
   p_option_flags           PLS_INTEGER DEFAULT 0);
```

Parameters

Table 215-26 INVALIDATE Procedure Parameters

Parameter	Description
p_object_id	ID number of object to be invalidated. This is the same as the value of the OBJECT_ID column from ALL_OBJECTS. If the object_id argument is NULL or invalid then the exception inv_not_exist_or_no_priv is raised. The caller of this procedure must have create privileges on the object being invalidated else the inv_not_exist_or_no_priv exception is raised.

Table 215-26 (Cont.) INVALIDATE Procedure Parameters

Parameter	Description
p_plsql_object_settings	Optional parameter that ignored if the object specified by p_object_id is not a PL/SQL object. If no value is specified for this parameter then the PL/SQL compiler settings are left unchanged, that is, equivalent to REUSE SETTINGS. If a value is provided, it must specify the values of the PL/SQL compiler settings separated by one or more spaces. Each setting can be specified only once else inv_malformed_settings exception will be raised. The setting values are changed only for the object specified by p_object_id and do not affect dependent objects that may be invalidated. The setting names and values are case insensitive. If a setting is omitted and REUSE SETTINGS is specified, then if a value was specified for the compiler setting in an earlier compilation of this library unit, Oracle Database uses that earlier value. If a setting is omitted and REUSE SETTINGS was not specified or no value has been specified for the parameter in an earlier compilation, then the database will obtain the value for that setting from the session environment.
p_option_flags	Optional parameter defaults to zero (no flags). Option flags supported by invalidate. • inv_error_on_restrictions (see Constants): The subprogram imposes various restrictions on the objects that can be invalidated. For example, the object specified by p_object_id cannot be a table. By default, invalidate quietly returns on these conditions (and does not raise an exception). If the caller sets this flag, the exception inv_restricted_object is raised.

Exceptions

Table 215-27 INVALIDATE Exceptions

Exception	Description
INV_NOT_EXIST_OR_NO_PRIV	Raised when the object_id argument is NULL or invalid, or when the caller does not have CREATE privileges on the object being invalidated
INV_MALFORMED_SETTINGS	Raised if a compiler setting is specified more than once in the p_plsql_object_settings parameter
INV_RESTRICTED_OBJECT	Raised when different combinations of conditions pertaining to the p_object_id parameter are contravened

Usage Notes

The object type (object_type column from ALL_OBJECTS) of the object specified by p_object_id must be a PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, TYPE, TYPE BODY, LIBRARY, VIEW, OPERATOR, SYNONYM, or JAVA CLASS. If the object is not one of these types and the flag inv_error_on_restrictions is specified in p_option_flags then the exception inv_restricted_object is raised, else no action is taken.

If the object specified by p_object_id is the package specification of STANDARD, DBMS_STANDARD, or specification or body of DBMS_UTILITY and the flag

inv_error_on_restrictions is specified in p_option_flags then the exception inv_restricted_object is raised, else no action is taken.

If the object specified by p_object_id is an object type specification and there exist tables which depend on the type and the flag inv_error_on_restrictions is specified in p_option_flags then the exception inv_restricted_object is raised, else no action is taken.

Examples

Example 1

```
DBMS_UTILITY.INVALIDATE (1232, 'PLSQL_OPTIMIZE_LEVEL = 2 REUSE SETTINGS');
```

Assume that the object_id 1232 refers to the procedure remove_emp in the HR schema. Then the above call will mark the remove_emp procedure invalid and change it's PLSQL_OPTIMIZE_LEVEL compiler setting to 2. The values of other compiler settings will remain unchanged since REUSE SETTINGS is specified.

Objects that depend on hr.remove_emp will also get marked invalid. Their compiler parameters will not be changed.

Example 2

```
DBMS_UTILITY.INVALIDATE (40775, 'plsql_code_type = native');
```

Assume that the object_id 40775 refers to the type body leaf_category_typ in the OE schema. Then the above call will mark the type body invalid and change its PLSQL_CODE_TYPE compiler setting to NATIVE. The values of other compiler settings will be picked up from the current session environment since REUSE SETTINGS has not been specified.

Since no objects can depend on bodies, there are no cascaded invalidations.

Example 3

```
DBMS_UTILITY.INVALIDATE (40796);
```

Assume that the object_id 40796 refers to the view oc_orders in the OE schema. Then the above call will mark the oc_orders view invalid.

Objects that depend on oe.oc_orders will also get marked invalid.

IS_BIT_SET Function

This function checks the bit setting for the given bit in the given RAW value.

Syntax

```
DBMS_UTILITY.IS_BIT_SET (
   r     IN    RAW,   n     IN    NUMBER)
 RETURN NUMBER;
```

Parameters

Table 215-28 IS_BET_SET Function Parameters

Parameter	Description
r	RAW source

Table 215-28 (Cont.) IS_BET_SET Function Parameters

Parameter	Description
n	Bit in r to check

Return Values

This function returns 1 if bit n in raw r is set, zero otherwise. Bits are numbered high to low with the lowest bit being bit number 1.

IS_CLUSTER_DATABASE Function

This function finds out if this database is running in cluster database mode.

Syntax

```
DBMS_UTILITY.IS_CLUSTER_DATABASE
  RETURN BOOLEAN;
```

Return Values

This function returns TRUE if this instance was started in cluster database mode; FALSE otherwise.

MAKE_DATA_BLOCK_ADDRESS Function

This function creates a data block address given a file number and a block number.

A data block address is the internal structure used to identify a block in the database. This function is useful when accessing certain fixed tables that contain data block addresses.

Syntax

```
DBMS_UTILITY.MAKE_DATA_BLOCK_ADDRESS (
   file  NUMBER,
   block NUMBER)
  RETURN NUMBER;
```

Parameters

Table 215-29 MAKE_DATA_BLOCK_ADDRESS Function Parameters

Parameter	Description
file	File that contains the block
block	Offset of the block within the file in terms of block increments

Pragmas

```
pragma restrict_references (make_data_block_address, WNDS, RNDS, WNPS, RNPS);
```

Return Values

Data block address.

NAME_RESOLVE Procedure

This procedure resolves the given name, including synonym translation and authorization checking as necessary.

Syntax

```
DBMS_UTILITY.NAME_RESOLVE (
   name          IN  VARCHAR2,
   context       IN  NUMBER,
   schema        OUT VARCHAR2,
   part1         OUT VARCHAR2,
   part2         OUT VARCHAR2,
   dblink        OUT VARCHAR2,
   part1_type    OUT NUMBER,
   object_number OUT NUMBER);
```

Parameters

Table 215-30 NAME_RESOLVE Procedure Parameters

Parameter	Description
`name`	Name of the object. This can be of the form [[a.]b.]c[@d], where a, b, c are SQL identifier and d is a dblink. No syntax checking is performed on the dblink. If a dblink is specified, or if the name resolves to something with a dblink, then object is not resolved, but the `schema`, `part1`, `part2` and `dblink` `OUT` parameters are filled in. a, b and c may be delimited identifiers, and may contain Globalization Support (NLS) characters (single and multibyte).
`context`	Must be an integer between 0 and 9. • 0 - table • 1 - PL/SQL (for 2 part names) • 2 - sequences • 3 - trigger • 4 - Java Source • 5 - Java resource • 6 - Java class • 7 - type • 8 - Java shared data • 9 - index
`schema`	Schema of the object: c. If no schema is specified in `name`, then the `schema` is determined by resolving the name.
`part1`	First part of the name. The type of this name is specified `part1_type` (synonym or package).
`part2`	If this is non-`NULL`, then this is a subprogram name. If part1 is non-`NULL`, then the subprogram is within the package indicated by part1. If part1 is `NULL`, then the subprogram is a top-level subprogram.
`dblink`	If this is non-`NULL`, then a database link was either specified as part of `name` or `name` was a synonym which resolved to something with a database link. In this case, if further name translation is desired, then you must call the `DBMS_UTILITY.NAME_RESOLVE` procedure on this remote node.

Table 215-30 (Cont.) NAME_RESOLVE Procedure Parameters

Parameter	Description
part1_type	Type of part1 is: • 5 - synonym • 7 - procedure (top level) • 8 - function (top level) • 9 - package
object_number	Object identifier

Exceptions

All errors are handled by raising exceptions. A wide variety of exceptions are possible, based on the various syntax error that are possible when specifying object names.

NAME_TOKENIZE Procedure

This procedure calls the parser to parse the given name as a [. b [. c]][@ dblink].

It strips double quotes, or converts to uppercase if there are no quotes. It ignores comments of all sorts, and does no semantic analysis. Missing values are left as NULL.

Syntax

```
DBMS_UTILITY.NAME_TOKENIZE (
   name    IN  VARCHAR2,
   a       OUT VARCHAR2,
   b       OUT VARCHAR2,
   c       OUT VARCHAR2,
   dblink  OUT VARCHAR2,
   nextpos OUT BINARY_INTEGER);
```

Parameters

Table 215-31 NAME_RESOLVE Procedure Parameters

Parameter	Description
name	Input name, consisting of SQL identifiers (for example, scott.foo@dblink)
a	Output for the first token of the name
b	Output for the second token of the name (if applicable)
c	Output for the third token of the name (if applicable)
dblink	Output for the dblink of the name
nextpos	Next position after parsing the input name

OLD_CURRENT_SCHEMA Function

This function returns the session value from `sys_context` ('`userenv`', '`current_schema`').

Syntax

```
DBMS_UTILITY.OLD_CURRENT_SCHEMA
 RETURN VARCHAR2;
```

OLD_CURRENT_USER Function

This function returns the session value from `sys_context` ('`userenv`', '`current_user`').

Syntax

```
DBMS_UTILITY.OLD_CURRENT_USER
 RETURN VARCHAR2;
```

PORT_STRING Function

This function returns a string that identifies the operating system and the `TWO TASK PROTOCOL` version of the database. For example, "`VAX/VMX-7.1.0.0`"

The maximum length is port-specific.

Syntax

```
DBMS_UTILITY.PORT_STRING
   RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(port_string, WNDS, RNDS, WNPS, RNPS);
```

SQLID_TO_SQLHASH Function

This function converts a SQL ID into a hash value.

Syntax

```
DBMS_UTILITY.SQLID_TO_SQLHASH (
   sql_id    IN   VARCHAR2)
 RETURN NUMBER;
```

Parameters

Table 215-32 SQLID_TO_SQLHASH Function Parameters

Parameter	Description
`sql_id`	SQL ID of a SQL statement. Must be `VARCHAR2(13)`.

TABLE_TO_COMMA Procedures

This procedure converts a PL/SQL table of names into a comma-delimited list of names.

This takes a PL/SQL table, `1..n`, terminated with `n+1 null`. The second version supports fully-qualified attribute names.

Syntax

```
DBMS_UTILITY.TABLE_TO_COMMA (
   tab    IN  UNCL_ARRAY,
   tablen OUT BINARY_INTEGER,
   list   OUT VARCHAR2);

DBMS_UTILITY.TABLE_TO_COMMA (
   tab    IN  lname_array,
   tablen OUT BINARY_INTEGER,
   list   OUT VARCHAR2);
```

Parameters

Table 215-33 TABLE_TO_COMMA Procedure Parameters

Parameter	Description
`tab`	PL/SQL table which contains list of table names
`tablen`	Number of tables in the PL/SQL table
`list`	Comma separated list of tables

Return Values

A comma-delimited list and the number of elements found in the table.

VALIDATE Procedure

This procedure makes invalid database objects valid.

Syntax

```
DBMS_UTILITY.VALIDATE(
    object_id       NUMBER);

DBMS_UTILITY.VALIDATE(
   owner          VARCHAR2,
   objname        VARCHAR2,
   namespace      NUMBER,   edition_name   := SYS_CONTEXT ('USERENV',
'CURRENT_EDITION'));
```

Parameters

Table 215-34 VALIDATE Procedure Parameters

Parameter	Description
`owner`	Name of the user who owns the object. Same as the `OWNER` field in `ALL_OBJECTS`.

Table 215-34 (Cont.) VALIDATE Procedure Parameters

Parameter	Description
objname	Name of the object to be validated. Same as the OBJECT_NAME field in ALL_OBJECTS.
namespace	Namespace of the object. Same as the namespace field in obj$. Equivalent numeric values are as follows: • 1 — TABLE/PROCEDURE/TYPE • 2 — BODY • 3 — TRIGGER • 4 — INDEX • 5 — CLUSTER • 8 — LOB • 9 — DIRECTORY • 10 — QUEUE • 11 — REPLICATION OBJECT GROUP • 12 — REPLICATION PROPAGATOR • 13 — JAVA SOURCE • 14 — JAVA RESOURCE • 58 — (Data Mining) MODEL
edition_name	[Note: Currently not operable. Reserved for future use]

Usage Notes

- No errors are raised if the object does not exist or is already valid or is an object that cannot be validated.
- If the object being validated is not actual in the specified edition, the subprogram automatically switches into the edition in which the object is actual prior to validation. That is, a call to VALIDATE will not actualize the object in the specified edition.
- The INVALIDATE Procedure invalidates a database object and optionally changes its PL/SQL compiler parameter settings. The object to be invalidated is specified by its object_id. The subprogram automatically switches to the edition in which the object is actual prior to invalidation. That is, a call to INVALIDATE will not actualize the object in the current edition.

WAIT_ON_PENDING_DML Function

This function waits until all transactions (other than the caller's own) that have locks on the listed tables and began prior to the specified scn have either committed or been rolled back.

Syntax

```
DBMS_UTILITY.WAIT_ON_PENDING_DML (
    tables     IN       VARCHAR2,
    timeout    IN       BINARY_INTEGER,
    scn        IN OUT   NUMBER)
  RETURN BOOLEAN;
```

Parameters

Table 215-35 WAIT_ON_PENDING_DML Function Parameters

Parameter	Description
tables	Comma-separated list of one or more table names. The list must be valid for COMMA_TO_TABLE Procedures, and each item valid to the NAME_RESOLVE Procedure. Neither column specifiers nor DBLINK (database link) specifiers are allowed in the names, and each name must resolve to an existing table in the local database.
timeout	Maximum number of seconds to wait, totalled across all tables/ transactions. A NULL or negative value will cause a very long wait.
scn	SCN prior to which transactions must have begun to be considered relevant to this request. If the value is NULL or not recognized as a meaningful scn on input, the most current SCN across all instances will be used and will be set into the passed argument as an output. If a meaningful value is passed in, its value will be preserved in the output.

Return Values

TRUE if all relevant transactions have committed or been rolled back, FALSE if the timeout occurred prior to all relevant transactions committing or being rolled back