142 DBMS_OUTPUT

The `DBMS_OUTPUT` package enables you to send messages from stored procedures, packages, and triggers. The package is especially useful for displaying PL/SQL debugging information.

This chapter contains the following topics:

DBMS_OUTPUT Overview

The package is typically used for debugging, or for displaying messages and reports to SQL*DBA or SQL*Plus (such as are produced by applying the SQL command `DESCRIBE` to procedures).

The PUT Procedure and PUT_LINE Procedure in this package enable you to place information in a buffer that can be read by another trigger, procedure, or package. In a separate PL/SQL procedure or anonymous block, you can display the buffered information by calling the GET_LINE Procedure and GET_LINES Procedure.

If the package is disabled, all calls to subprograms are ignored. In this way, you can design your application so that subprograms are available only when a client is able to process the information.

DBMS_OUTPUT Security Model

The `dbmsotpt.sql` script must be run as user `SYS`. This creates the public synonym `DBMS_OUTPUT`, and `EXECUTE` permission on this package is granted to `public`.

DBMS_OUTPUT Operational Notes

The following operational notes apply to DBMS_OUTPUT.

- If you do not call `GET_LINE`, or if you do not display the messages on your screen in SQL*Plus, the buffered messages are ignored.
- SQL*Plus calls `GET_LINES` after issuing a SQL statement or anonymous PL/SQL calls.
- Typing `SET SERVEROUTPUT ON` in SQL*Plus has the effect of invoking

```
DBMS_OUTPUT.ENABLE (buffer_size => NULL);
```

with no limit on the output.

- You should generally avoid having application code invoke either the DISABLE Procedure or ENABLE Procedure because this could subvert the attempt of an external tool like SQL*Plus to control whether or not to display output.

Note:

Messages sent using `DBMS_OUTPUT` are not actually sent until the sending subprogram or trigger completes. There is no mechanism to flush output during the execution of a procedure.

DBMS_OUTPUT Exceptions

`DBMS_OUTPUT` subprograms raise the application error `ORA-20000` and return errors.

The output procedures can return the following errors:

Table 142-1 DBMS_OUTPUT Errors

Error	Description
`ORU-10027:`	Buffer overflow
`ORU-10028:`	Line length overflow

DBMS_OUTPUT Rules and Limits

The following are limits on DBMS_OUTPUT line and buffer size.

- The maximum line size is 32767 bytes.
- The default buffer size is 20000 bytes. The minimum size is 2000 bytes and the maximum is unlimited.

DBMS_OUTPUT Examples

This topic contains three examples of using DBMS_OUTPUT.

Example 1: Using a Trigger to Produce Output

You can use a trigger to print out some output from the debugging process. For example, you could code the trigger to invoke:

```
DBMS_OUTPUT.PUT_LINE('I got here:'||:new.col||' is the new value');
```

If you have enabled the `DBMS_OUTPUT` package, then the text produced by this `PUT_LINE` would be buffered, and you could, after executing the statement (presumably some `INSERT`, `DELETE`, or `UPDATE` that caused the trigger to fire), retrieve the line of information. For example:

```
BEGIN
   DBMS_OUTPUT.GET_LINE(:buffer, :status);
END;
```

You could then optionally display the buffer on the screen. You repeat calls to `GET_LINE` until `status` comes back as nonzero. For better performance, you should use calls to GET_LINES Procedure which can return an array of lines.

Example 2: Debugging Stored Procedures and Triggers

The `DBMS_OUTPUT` package is commonly used to debug stored procedures and triggers. This package can also be used to enable you to retrieve information about an object and format this output, as shown in "Example 3: Retrieving Information About an Object".

This function queries the employee table and returns the total salary for a specified department. The function includes several calls to the `PUT_LINE` procedure:

```
CREATE FUNCTION dept_salary (dnum NUMBER) RETURN NUMBER IS
   CURSOR emp_cursor IS
      SELECT sal, comm FROM emp WHERE deptno = dnum;
   total_wages    NUMBER(11, 2) := 0;
   counter        NUMBER(10) := 1;
BEGIN

   FOR emp_record IN emp_cursor LOOP
      emp_record.comm := NVL(emp_record.comm, 0);
      total_wages := total_wages + emp_record.sal
         + emp_record.comm;
      DBMS_OUTPUT.PUT_LINE('Loop number = ' || counter ||
         '; Wages = '|| TO_CHAR(total_wages));    /* Debug line */
      counter := counter + 1; /* Increment debug counter */
   END LOOP;
   /* Debug line */
   DBMS_OUTPUT.PUT_LINE('Total wages = ' ||
     TO_CHAR(total_wages));
   RETURN total_wages;

END dept_salary;
```

Assume the `EMP` table contains the following rows:

```
EMPNO           SAL     COMM     DEPT
-----        ------- -------- -------
1002            1500      500      20
1203            1000               30
1289            1000               10
1347            1000      250      20
```

Assume the user executes the following statements in SQL*Plus:

```
SET SERVEROUTPUT ON
VARIABLE salary NUMBER;
EXECUTE :salary := dept_salary(20);
```

The user would then see the following information displayed in the output pane:

```
Loop number = 1; Wages = 2000
Loop number = 2; Wages = 3250
Total wages = 3250

PL/SQL procedure successfully executed.
```

Example 3: Retrieving Information About an Object

In this example, the user has used the `EXPLAIN PLAN` command to retrieve information about the execution plan for a statement and has stored it in `PLAN_TABLE`. The user has also assigned a statement ID to this statement. The example `EXPLAIN_OUT` procedure retrieves the information from this table and formats the output in a nested manner that more closely depicts the order of steps undergone in processing the SQL statement.

```
 /****************************************************************/
/* Create EXPLAIN_OUT procedure. User must pass STATEMENT_ID to */
/* to procedure, to uniquely identify statement.                */
/****************************************************************/
CREATE OR REPLACE PROCEDURE explain_out
   (statement_id IN VARCHAR2) AS

   -- Retrieve information from PLAN_TABLE into cursor EXPLAIN_ROWS.

   CURSOR explain_rows IS
      SELECT level, id, position, operation, options,
         object_name
      FROM plan_table
      WHERE statement_id = explain_out.statement_id
      CONNECT BY PRIOR id = parent_id
         AND statement_id = explain_out.statement_id
      START WITH id = 0
       ORDER BY id;

BEGIN

   -- Loop through information retrieved from PLAN_TABLE:

   FOR line IN explain_rows LOOP

      -- At start of output, include heading with estimated cost.

      IF line.id = 0 THEN
         DBMS_OUTPUT.PUT_LINE ('Plan for statement '
            || statement_id
            || ', estimated cost = ' || line.position);
      END IF;

      -- Output formatted information. LEVEL determines indention level.

      DBMS_OUTPUT.PUT_LINE (lpad(' ',2*(line.level-1)) ||
         line.operation || ' ' || line.options || ' ' ||
         line.object_name);
   END LOOP;

END;
```

See Also:

UTL_FILE

DBMS_OUTPUT Data Structures

The `DBMS_OUTPUT` package declares 2 collection types for use with the GET_LINES Procedure.

TABLE Types

CHARARR Table Type

OBJECT Types

DBMSOUTPUT_LINESARRAY Object Type

Related Topics

- GET_LINES Procedure
  This procedure retrieves an array of lines from the buffer.

CHARARR Table Type

This package type is to be used with the GET_LINES Procedure to obtain text submitted through the PUT Procedure and PUT_LINE Procedure.

Syntax

```
TYPE CHARARR IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

Related Topics

- GET_LINES Procedure
  This procedure retrieves an array of lines from the buffer.
- PUT Procedure
  This procedure places a partial line in the buffer.
- PUT_LINE Procedure
  This procedure places a line in the buffer.

DBMS_OUTPUT DBMSOUTPUT_LINESARRAY Object Type

This type, defined outside the package, is to be used with the GET_LINES Procedure to obtain text submitted through the PUT Procedure and PUT_LINE Procedure.

Syntax

```
TYPE DBMSOUTPUT_LINESARRAY IS
     VARRAY(2147483647) OF VARCHAR2(32767);
```

Related Topics

- GET_LINES Procedure
  This procedure retrieves an array of lines from the buffer.
- PUT Procedure
  This procedure places a partial line in the buffer.
- PUT_LINE Procedure
  This procedure places a line in the buffer.

Summary of DBMS_OUTPUT Subprograms

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

Table 142-2 DBMS_OUTPUT Package Subprograms

Subprogram	Description
DISABLE Procedure	Disables message output
ENABLE Procedure	Enables message output
GET_LINE Procedure	Retrieves one line from buffer
GET_LINES Procedure	Retrieves an array of lines from buffer
NEW_LINE Procedure	Terminates a line created with `PUT`
PUT Procedure	Places a partial line in the buffer
PUT_LINE Procedure	Places line in buffer

Note:

The PUT Procedure that take a number are obsolete and, while currently supported, are included in this release for legacy reasons only.

DISABLE Procedure

This procedure disables calls to `PUT`, `PUT_LINE`, `NEW_LINE`, `GET_LINE`, and `GET_LINES`, and purges the buffer of any remaining information.

As with the ENABLE Procedure, you do not need to call this procedure if you are using the `SERVEROUTPUT` option of SQL*Plus.

Syntax

```
DBMS_OUTPUT.DISABLE;
```

Pragmas

```
pragma restrict_references(disable,WNDS,RNDS);
```

ENABLE Procedure

This procedure enables calls to `PUT`, `PUT_LINE`, `NEW_LINE`, `GET_LINE`, and `GET_LINES`.

Calls to these procedures are ignored if the `DBMS_OUTPUT` package is not activated.

Syntax

```
DBMS_OUTPUT.ENABLE (
   buffer_size IN INTEGER DEFAULT 20000);
```

Pragmas

```
pragma restrict_references(enable,WNDS,RNDS);
```

Parameters

Table 142-3 ENABLE Procedure Parameters

Parameter	Description
buffer_size	Upper limit, in bytes, the amount of buffered information. Setting buffer_size to NULL specifies that there should be no limit.

Usage Notes

- It is not necessary to call this procedure when you use the SET SERVEROUTPUT option of SQL*Plus.
- If there are multiple calls to ENABLE, then buffer_size is the last of the values specified. The maximum size is 1,000,000, and the minimum is 2,000 when the user specifies buffer_size (NOT NULL).
- NULL is expected to be the usual choice. The default is 20,000 for backwards compatibility with earlier database versions that did not support unlimited buffering.

GET_LINE Procedure

This procedure retrieves a single line of buffered information.

Syntax

```
DBMS_OUTPUT.GET_LINE (
   line     OUT VARCHAR2,
   status   OUT INTEGER);
```

Parameters

Table 142-4 GET_LINE Procedure Parameters

Parameter	Description
line	Returns a single line of buffered information, excluding a final newline character. You should declare the actual for this parameter as VARCHAR2 (32767) to avoid the risk of "ORA-06502: PL/SQL: numeric or value error: character string buffer too small".
status	If the call completes successfully, then the status returns as 0. If there are no more lines in the buffer, then the status is 1.

Usage Notes

- You can choose to retrieve from the buffer a single line or an array of lines. Call the GET_LINE procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the GET_LINES procedure to retrieve an array of lines from the buffer.
- You can choose to automatically display this information if you are using SQL*Plus by using the special SET SERVEROUTPUT ON command.
- After calling GET_LINE or GET_LINES, any lines not retrieved before the next call to PUT, PUT_LINE, or NEW_LINE are discarded to avoid confusing them with the next message.

GET_LINES Procedure

This procedure retrieves an array of lines from the buffer.

Syntax

```
DBMS_OUTPUT.GET_LINES (
   lines       OUT     CHARARR,
   numlines    IN OUT  INTEGER);

DBMS_OUTPUT.GET_LINES (
   lines       OUT     DBMSOUTPUT_LINESARRAY,
   numlines    IN OUT INTEGER);
```

Parameters

Table 142-5 GET_LINES Procedure Parameters

Parameter	Description
`lines`	Returns an array of lines of buffered information. The maximum length of each line in the array is 32767 bytes. It is recommended that you use the `VARRAY` overload version in a 3GL host program to execute the procedure from a PL/SQL anonymous block.
`numlines`	Number of lines you want to retrieve from the buffer. After retrieving the specified number of lines, the procedure returns the number of lines actually retrieved. If this number is less than the number of lines requested, then there are no more lines in the buffer.

Usage Notes

- You can choose to retrieve from the buffer a single line or an array of lines. Call the `GET_LINE` procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the `GET_LINES` procedure to retrieve an array of lines from the buffer.
- You can choose to automatically display this information if you are using SQL*Plus by using the special `SET SERVEROUTPUT ON` command.
- After calling `GET_LINE` or `GET_LINES`, any lines not retrieved before the next call to `PUT`, `PUT_LINE`, or `NEW_LINE` are discarded to avoid confusing them with the next message.

NEW_LINE Procedure

This procedure puts an end-of-line marker.

The GET_LINE Procedure and the GET_LINES Procedure return "lines" as delimited by "newlines". Every call to the PUT_LINE Procedure or NEW_LINE Procedure generates a line that is returned by `GET_LINE`(`S`).

Syntax

```
DBMS_OUTPUT.NEW_LINE;
```

PUT Procedure

This procedure places a partial line in the buffer.

Note:

The PUT procedure that takes a NUMBER is obsolete and, while currently supported, is included in this release for legacy reasons only.

Syntax

```
DBMS_OUTPUT.PUT (
    item IN VARCHAR2);
```

Parameters

Table 142-6 PUT Procedure Parameters

Parameter	Description
item	Item to buffer.

Exceptions

Table 142-7 PUT Procedure Exceptions

Error	Description
ORA-20000, ORU-10027:	Buffer overflow, limit of <buf_limit> bytes.
ORA-20000, ORU-10028:	Line length overflow, limit of 32767 bytes for each line.

Usage Notes

- You can build a line of information piece by piece by making multiple calls to PUT, or place an entire line of information into the buffer by calling PUT_LINE.
- When you call PUT_LINE the item you specify is automatically followed by an end-of-line marker. If you make calls to PUT to build a line, then you must add your own end-of-line marker by calling NEW_LINE. GET_LINE and GET_LINES do not return lines that have not been terminated with a newline character.
- If your lines exceed the line limit, you receive an error message.
- Output that you create using PUT or PUT_LINE is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

  For example, SQL*Plus does not display DBMS_OUTPUT messages until the PL/SQL program completes. There is no mechanism for flushing the DBMS_OUTPUT buffers within the PL/SQL program.

  ```
  SQL> SET SERVEROUTPUT ON
  SQL> BEGIN
  ```

```
  2  DBMS_OUTPUT.PUT_LINE ('hello');
  3  DBMS_LOCK.SLEEP (10);
  4  END;
```

PUT_LINE Procedure

This procedure places a line in the buffer.

> **Note:**
>
> The `PUT_LINE` procedure that takes a `NUMBER` is obsolete and, while currently supported, is included in this release for legacy reasons only.

Syntax

```
DBMS_OUTPUT.PUT_LINE (
   item IN VARCHAR2);
```

Parameters

Table 142-8   PUT_LINE Procedure Parameters

Parameter	Description
`item`	Item to buffer.

Exceptions

Table 142-9   PUT_LINE Procedure Exceptions

Error	Description
`ORA-20000, ORU-10027:`	Buffer overflow, limit of <`buf_limit`> bytes.
`ORA-20000, ORU-10028:`	Line length overflow, limit of 32767 bytes for each line.

Usage Notes

- You can build a line of information piece by piece by making multiple calls to `PUT`, or place an entire line of information into the buffer by calling `PUT_LINE`.
- When you call `PUT_LINE` the item you specify is automatically followed by an end-of-line marker. If you make calls to `PUT` to build a line, then you must add your own end-of-line marker by calling `NEW_LINE`. `GET_LINE` and `GET_LINES` do not return lines that have not been terminated with a newline character.
- If your lines exceeds the line limit, you receive an error message.
- Output that you create using `PUT` or `PUT_LINE` is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

  For example, SQL*Plus does not display `DBMS_OUTPUT` messages until the PL/SQL program completes. There is no mechanism for flushing the `DBMS_OUTPUT` buffers within the PL/SQL program. For example:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
   2 DBMS_OUTPUT.PUT_LINE ('hello');
   3 DBMS_LOCK.SLEEP (10);
   4 END;
```