Database Programming with PL/SQL

11-2
Using Oracle-Supplied Packages
Objectives

This lesson covers the following objectives:

• Describe two common uses for the `DBMS_OUTPUT` server-supplied package

• Recognize the correct syntax to specify messages for the `DBMS_OUTPUT` package

• Describe the purpose for the `UTL_FILE` server-supplied package

• Recall the exceptions used in conjunction with the `UTL_FILE` server-supplied package

• Describe the main features of the `UTL_MAIL` server-supplied package
Purpose

- You already know that Oracle supplies a number of SQL functions (UPPER, TO_CHAR, and so on) that you can use in your SQL statements when required.

- It would be wasteful for you to have to “re-invent the wheel” by writing your own functions to do these things.

- In the same way, Oracle supplies a number of ready-made PL/SQL packages to do things that most application developers and/or database administrators need to do from time to time.
Purpose

• In this lesson, you learn how to use two of the Oracle-supplied PL/SQL packages.
• These packages focus on generating text output and manipulating text files.
Using Oracle-Supplied Packages

- You can use these packages directly by invoking them from your own application, exactly as you would invoke packages that you had written yourself.
- Or, you can use these packages as ideas when you create your own subprograms.
- Think of these packages as ready-made “building blocks” that you can invoke from your own applications.
## List of Some Oracle-Supplied Packages

<table>
<thead>
<tr>
<th>Tab</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBMS_LOB</strong></td>
<td>Enables manipulation of Oracle Large Object column datatypes: CLOB, BLOB and BFILE</td>
</tr>
<tr>
<td><strong>DBMS_LOCK</strong></td>
<td>Used to request, convert, and release locks in the database through Oracle Lock Management services</td>
</tr>
<tr>
<td><strong>DBMS_OUTPUT</strong></td>
<td>Provides debugging and buffering of messages</td>
</tr>
<tr>
<td><strong>HTP</strong></td>
<td>Writes HTML-tagged data into database buffers</td>
</tr>
<tr>
<td><strong>UTL_FILE</strong></td>
<td>Enables reading and writing of operating system text files</td>
</tr>
<tr>
<td><strong>UTL_MAIL</strong></td>
<td>Enables composing and sending of e-mail messages</td>
</tr>
<tr>
<td><strong>DBMS_SCHEDULER</strong></td>
<td>Enables scheduling of PL/SQL blocks, stored procedures, and external procedures or executables</td>
</tr>
</tbody>
</table>
The **DBMS_OUTPUT** Package

- The **DBMS_OUTPUT** package sends text messages from any PL/SQL block into a private memory area, from which the message can be displayed on the screen.

- Common uses of **DBMS_OUTPUT** include:
  - You can output results back to the developer during testing for debugging purposes.
  - You can trace the code execution path for a function or procedure.
How the **DBMS_OUTPUT** Package Works

- The **DBMS_OUTPUT** package enables you to send messages from stored subprograms and anonymous blocks.
- **PUT** places text in the buffer.
- **NEW_LINE** sends the buffer to the screen.
- **PUT_LINE** **does a PUT followed by a NEW_LINE**.
- **GET_LINE** and **GET_LINES** read the buffer.
- Messages are not sent until after the calling block finishes.
How the `DBMS_OUTPUT` Package Works

```plsql
BEGIN
    DBMS_OUTPUT...;
END;
```
Using `DBMS_OUTPUT`: Example 1

- You have already used `DBMS_OUTPUT.PUT_LINE`.
- This writes a text message into a buffer, then displays the buffer on the screen:

```sql
BEGIN
   DBMS_OUTPUT.PUT_LINE('The cat sat on the mat');
END;
```

- If you wanted to build a message a little at a time, you could code:

```sql
BEGIN
   DBMS_OUTPUT.PUT('The cat sat ');
   DBMS_OUTPUT.PUT('on the mat');
   DBMS_OUTPUT.NEW_LINE;
END;
```
Using **DBMS_OUTPUT**: Example 2

You can trace the flow of execution of a block with complex **IF...ELSE, CASE, or looping structures**:

```plsql
DECLARE
  v_bool1    BOOLEAN := true;
  v_bool2    BOOLEAN := false;
  v_number   NUMBER;
BEGIN
  ...
  IF v_bool1 AND NOT v_bool2 AND v_number < 25 THEN
    DBMS_OUTPUT.PUT_LINE('IF branch was executed');
  ELSE
    DBMS_OUTPUT.PUT_LINE('ELSE branch was executed');
  END IF;
  ...
END;
```
**DBMS_OUTPUT** Is Designed for Debugging Only

- You would not use **DBMS_OUTPUT** in PL/SQL programs that are called from a “real” application, which can include its own application code to display results on the user’s screen.

- Instead, you would return the text to be displayed as an **OUT** argument from the subprogram.
DBMS_OUTPUT Is Designed for Debugging Only

• For example:

```
CREATE OR REPLACE PROCEDURE do_some_work (...) IS
BEGIN
    ... DBMS_OUTPUT.PUT_LINE('string'); ...
END;
```

• Would be converted to:

```
CREATE OR REPLACE PROCEDURE do_some_work
    (... p_output OUT VARCHAR2)
IS BEGIN
    ... p_output := 'string'; ...
END;
```
DBMS_OUTPUT Is Designed for Debugging Only

- For this reason, you should not use DBMS_OUTPUT in subprograms, but only in anonymous PL/SQL blocks for testing purposes.
- Instead of:

```sql
CREATE OR REPLACE PROCEDURE do_some_work IS BEGIN
  ... DBMS_OUTPUT.PUT_LINE('string');
  ... END;
BEGIN do_some_work;
END; -- Test the procedure
```
DBMS_OUTPUT is designed for debugging only

You should use:

```sql
CREATE OR REPLACE PROCEDURE do_some_work
  (p_output OUT VARCHAR2) IS BEGIN
    ... p_output := 'string';
    ... END;

DECLARE v_output VARCHAR2(100);
BEGIN  --Test
do_some_work(v_output);
  DBMS_OUTPUT.PUT_LINE(v_output);
END;
```
The **UTL_FILE** Package

- Allows PL/SQL programs to read and write operating system text files.
- Can access text files in operating system directories defined by a `CREATE DIRECTORY` statement.
- You can also use the `utl_file_dir` database parameter.

BEGIN
    UTL_FILE...;
END;
File Processing Using the **UTL_FILE** Package

• Reading a file

```
f:=FOPEN(dir,file,'r')
```

<table>
<thead>
<tr>
<th>Open for reading</th>
<th>Get lines from the text file</th>
<th>More to read?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GET_LINE(f,buf,len)</td>
<td>Yes/No</td>
</tr>
</tbody>
</table>

• Writing or appending to a file

```
f:=FOPEN(dir,file,'w')
f:=FOPEN(dir,file,'a')
```

<table>
<thead>
<tr>
<th>Open for write/append</th>
<th>Put lines into the text file</th>
<th>More to write?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PUT(f,buf) PUT_LINE(f,buf)</td>
<td>Yes/No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Close the text file</th>
<th>FCLOSE(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>More to write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>More to read?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>f:=FOPEN(dir,file,'r')</th>
</tr>
</thead>
<tbody>
<tr>
<td>f:=FOPEN(dir,file,'w')</td>
</tr>
<tr>
<td>f:=FOPEN(dir,file,'a')</td>
</tr>
</tbody>
</table>
File Processing Using the UTL_FILE Package

• The GET_LINE procedure reads a line of text from the file into an output buffer parameter.
• The maximum input record size is 1,023 bytes.
• The PUT and PUT_LINE procedures write text to the opened file.
• The NEW_LINE procedure terminates a line in an output file.
• The FCLOSE procedure closes an opened file.
Exceptions in the **UTL_FILE** Package

**UTL_FILE** has its own set of exceptions that are applicable only when using this package:

- **INVALID_PATH**
- **INVALID_MODE**
- **INVALID_FILEHANDLE**
- **INVALID_OPERATION**
- **READ_ERROR**
- **WRITE_ERROR**
- **INTERNAL_ERROR**
Exceptions in the UTL_FILE Package

The other exceptions not specific to the UTL_FILE package are:

- `NO_DATA_FOUND`
- `VALUE_ERROR`
**FOPEN and IS_OPEN Function Parameters**

```sql
FUNCTION FOPEN (location IN VARCHAR2,
    filename IN VARCHAR2,
    open_mode IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;

FUNCTION IS_OPEN (file IN FILE_TYPE)
RETURN BOOLEAN;
```

**Example:**

```sql
PROCEDURE read(dir VARCHAR2, filename VARCHAR2) IS
    file UTL_FILE.FILE_TYPE;
BEGIN
    IF NOT UTL_FILE.IS_OPEN(file) THEN
        file := UTL_FILE.FOPEN (dir, filename, 'r');
    END IF; ...
END read;
```
Using **UTL_FILE** Example

- In this example, the `sal_status` procedure uses **UTL_FILE** to create a text report of employees for each department, along with their salaries.

- In the code, the variable `v_file` is declared as **UTL_FILE.FILE_TYPE**, a **BINARY_INTEGER** datatype that is declared globally by the **UTL_FILE** package.
Using UTL_FILE Example

• The `sal_status` procedure accepts two `IN` parameters: `p_dir` for the name of the directory in which to write the text file, and `p_filename` to specify the name of the file.

• To invoke the procedure, use (for example):

```
BEGIN sal_status('MY_DIR', 'salreport.txt');
END;
```
Using **UTL_FILE** Example

```sql
CREATE OR REPLACE PROCEDURE sal_status(
    p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
    v_file UTL_FILE.FILE_TYPE;
    CURSOR empc IS
        SELECT last_name, salary, department_id
        FROM employees ORDER BY department_id;
    v_newdeptno employees.department_id%TYPE;
    v_olddeptno employees.department_id%TYPE := 0;
BEGIN
    v_file := UTL_FILE.FOPEN (p_dir, p_filename, 'w');  -- 1
    UTL_FILE.PUT_LINE (v_file,                           -- 2
        'REPORT: GENERATED ON ' || SYSDATE);
    UTL_FILE.NEW_LINE (v_file); ...                     -- 3
END;
```
Using **UTL_FILE** Example

```plsql
FOR emp_rec IN empc LOOP
    UTL_FILE.PUT_LINE (v_file, -- 4
        '  EMPLOYEE: ' || emp_rec.last_name ||
        'earns: ' || emp_rec.salary);
END LOOP;

UTL_FILE.PUT_LINE(v_file,'*** END OF REPORT ***'); -- 5
UTL_FILE.FCLOSE (v_file); -- 6

EXCEPTION
    WHEN UTL_FILE.INVALID_FILEHANDLE THEN -- 7
        RAISE_APPLICATION_ERROR(-20001,'Invalid File.');
    WHEN UTL_FILE.WRITE_ERROR THEN -- 8
        RAISE_APPLICATION_ERROR (-20002, 'Unable to
                                        write to file');

END sal_status;
```
Using **UTL_FILE** Invocation and Output Report Example

• Suppose you invoke your procedure by:

```sql
BEGIN   sal_status('MYDIR', 'salreport.txt');   END;
```

• The output contained in the file is:

```sql
SALARY REPORT: GENERATED ON 29-NOV-06
   EMPLOYEE: Whalen earns: 4400
   EMPLOYEE: Hartstein earns: 13000
   EMPLOYEE: Fay earns: 6000
   ...  
   EMPLOYEE: Higgins earns: 12000
   EMPLOYEE: Gietz earns: 8300
   EMPLOYEE: Grant earns: 7000

*** END OF REPORT ***
```
The **UTL_MAIL** Package

- The **UTL_MAIL** package allows sending email from the Oracle database to remote recipients.
- Contains three procedures:
  - `SEND` for messages without attachments
  - `SEND_ATTACHMENT_RAW` for messages with binary attachments
  - `SEND_ATTACHMENT_VARCHAR2` for messages with text attachments
Using Oracle-Supplied Packages

The `UTL_MAIL SEND` Procedure Example

- Sends an email to one or more recipients.
- No attachments are allowed.

```sql
UTL_MAIL.SEND (  
    sender IN VARCHAR2,  
    recipients IN VARCHAR2,  
    cc IN VARCHAR2 DEFAULT NULL,  
    bcc IN VARCHAR2 DEFAULT NULL,  
    subject IN VARCHAR2 DEFAULT NULL,  
    message IN VARCHAR2,  
    ...);  
BEGIN  
    UTL_MAIL.SEND('database@oracle.com',  
                  'joe43@yahoo.com',  
                  message => 'Friday’s meeting will be at 10:30 in Room 6',  
                  subject => 'Our PL/SQL meeting');  
END;
```
The **UTL_MAIL.SEND_ATTACH_RAW** Procedure

- Similar to **UTL_MAIL.SEND**, but allows sending an attachment of data type **RAW** (for example, a small picture).

```sql
UTL_MAIL.SEND_ATTACH_RAW (  
    sender           IN    VARCHAR2,  
    recipients       IN    VARCHAR2,  
    cc               IN    VARCHAR2 DEFAULT NULL,  
    bcc              IN    VARCHAR2 DEFAULT NULL,  
    subject          IN    VARCHAR2 DEFAULT NULL,  
    message          IN    VARCHAR2 DEFAULT NULL,  
    ...  
    attachment       IN    RAW,  
    ...);
```

- The maximum size of a **RAW** file is 32,767 bytes, so you cannot use this to send a large JPEG, MP3, or WAV file.
The UTL_MAIL.SEND_ATTACHMENT_RAW Example

• In this example, the attachment is read from an operating system image file (named company_logo.gif) by a PL/SQL function (GET_IMAGE) which returns a RAW data type.

• Notice that all UTL_MAIL procedures allow you to send to more than one recipient.

• The recipients must be separated by commas.

```
BEGIN
    UTL_MAIL.SEND_ATTACHMENT_RAW(
        sender => 'marketing@ourcompany.com',
        recipients => 'sally@ourcompany.com,bill@ourcompany.com',
        message => 'Please display this logo on our website',
        subject => 'Display Logo',
        attachment => get_image('company_logo.gif'),
    );
END;
```
The **UTL_MAIL.SEND_ATTACH_RAW** Example

• Notice that all **UTL_MAIL** procedures allow you to send to more than one recipient.

• The recipients must be separated by commas.

```sql
BEGIN
    UTL_MAIL.SEND_ATTACH_RAW(
        sender    => 'marketing@ourcompany.com',
        recipients => 'sally@ourcompany.com,bill@ourcompany.com',
        message   => 'Please display this logo on our website',
        subject   => 'Display Logo',
        attachment => get_image('company_logo.gif'),
    )
END;
```
The **UTL_MAIL.SEND_ATTACH_VARCHAR2** Procedure

- This is identical to **UTL_MAIL.SEND_ATTACH_RAW**, but the attachment is a **VARCHAR2** text.

- Again, the maximum size of a **VARCHAR2** argument is 32,767 bytes, but this can be quite a large document.

```sql
UTL_MAIL.SEND_ATTACH_VARCHAR2 (  
    sender     IN    VARCHAR2,  
    recipients IN    VARCHAR2,  
    cc         IN    VARCHAR2 DEFAULT NULL,  
    bcc        IN    VARCHAR2 DEFAULT NULL,  
    subject    IN    VARCHAR2 DEFAULT NULL,  
    message    IN    VARCHAR2 DEFAULT NULL,  
    ...  
    attachment IN    VARCHAR2,  
    ...  
);  
```
In this example, the attachment is passed to a procedure as an argument, instead of being read from an operating system file.

```
CREATE OR REPLACE PROCEDURE send_mail_with_text
    (p_text_attachment IN VARCHAR2)
IS BEGIN
    UTL_MAIL.SEND_ATTACH_VARCHAR2(
        sender     => 'me@here.com',
        recipients => 'you@somewhere.net',
        message    => 'See attachment',
        subject    => 'Useful document for our project',
        attachment => p_text_attachment,
    )
END send_mail_with_text;
BEGIN
    send_mail_with_text('This document is designed to help in creating a project ...');
END;
```
Terminology

Key terms used in this lesson included:

• **DBMS_OUTPUT** package
• **UTL_FILE** package
• **UTL_MAIL** package
Summary

In this lesson, you should have learned how to:

• Describe two common uses for the DBMS_OUTPUT server-supplied package

• Recognize the correct syntax to specify messages for the DBMS_OUTPUT package

• Describe the purpose for the UTL_FILE server-supplied package

• Recall the exceptions used in conjunction with the UTL_FILE server-supplied package

• Describe the main features of the UTL_MAIL server-supplied package