Using Oracle-Supplied Packages

Objectives

After completing this lesson, you should be able to do the following:

- Describe how the `DBMS_OUTPUT` package works
- Use `UTL_FILE` to direct output to operating system files
- Describe the main features of `UTL_MAIL`

In this lesson, you learn how to use some of the Oracle-supplied packages and their capabilities.

Using Oracle-Supplied Packages

- The Oracle-supplied packages:
  - Are provided with the Oracle server
  - Extend the functionality of the database
  - Enable access to certain SQL features that are normally restricted for PL/SQL
- For example, the `DBMS_OUTPUT` package was originally designed to debug PL/SQL programs.
- Most of the standard packages are created by running `catproc.sql`.

Packages are provided with the Oracle server to allow either of the following:

- PL/SQL access to certain SQL features
- The extension of the functionality of the database

You can use the functionality provided by these packages when creating your application, or you may simply want to use these packages as ideas when you create your own stored procedures.

Most of the standard packages are created by running `catproc.sql`. The `DBMS_OUTPUT` package is the one that you will be most familiar with during this course.

Oracle Supplied Packages

- The list of PL/SQL packages provided with an Oracle database grows with the release of new versions.
- It would be impossible to cover the exhaustive set of packages and their functionality in this course.
  - **DBMS_ALERT** supports asynchronous notification of database events. Messages or alerts are sent on a `COMMIT` command.
  - **DBMS_LOCK** is used to request, convert, and release locks through Oracle Lock Management services.
  - **DBMS_SESSION** enables programmatic use of the `ALTER SESSION` SQL statement and other session-level commands.

Oracle Supplied Packages

- **DBMS_OUTPUT** provides debugging and buffering of text data.
- **HTTP** package writes HTML-tagged data into database buffers.
- **UTL_FILE** enables reading and writing of operating system text files.
- **UTL_MAIL** enables composing and sending of email messages.
- **DBMS_SCHEDULER** enables scheduling and automated execution of PL/SQL blocks, stored procedures, and external procedures and executables

Examples of Some Oracle-Supplied Packages

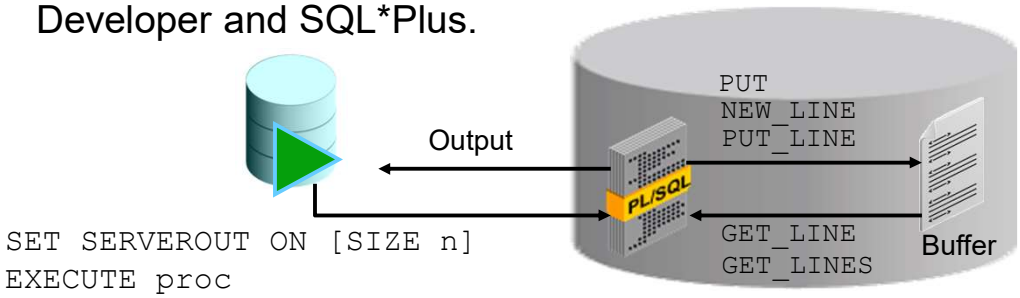
Here is an abbreviated list of some Oracle-supplied packages:

- `DBMS_OUTPUT`
- `UTL_FILE`
- `UTL_MAIL`

How the DBMS_OUTPUT Package Works

The DBMS_OUTPUT package enables you to send messages from stored subprograms and triggers.

- PUT and PUT_LINE place text in the buffer.
- GET_LINE and GET_LINES read the buffer.
- Messages are not sent until the sending subprogram or trigger completes.
- Use SET SERVEROUTPUT ON to display messages in SQL Developer and SQL*Plus.



ORACLE

The DBMS_OUTPUT package sends textual messages from any PL/SQL block into a buffer in the database. Procedures provided by the package include the following:

- PUT appends text from the procedure to the current line of the line output buffer.
- NEW_LINE places an end-of-line marker in the output buffer.
- PUT_LINE combines the action of PUT and NEW_LINE (to trim leading spaces).
- GET_LINE retrieves the current line from the buffer into a procedure variable.
- GET_LINES retrieves an array of lines into a procedure-array variable.
- ENABLE/DISABLE enables and disables calls to DBMS_OUTPUT procedures.

The buffer size can be set by using:

- The SIZE n option appended to the SET SERVEROUTPUT ON command where n is the number of characters. The minimum is 2,000 and the maximum is unlimited. The default is 20,000.
- An integer parameter between 2,000 and 1,000,000 in the ENABLE procedure

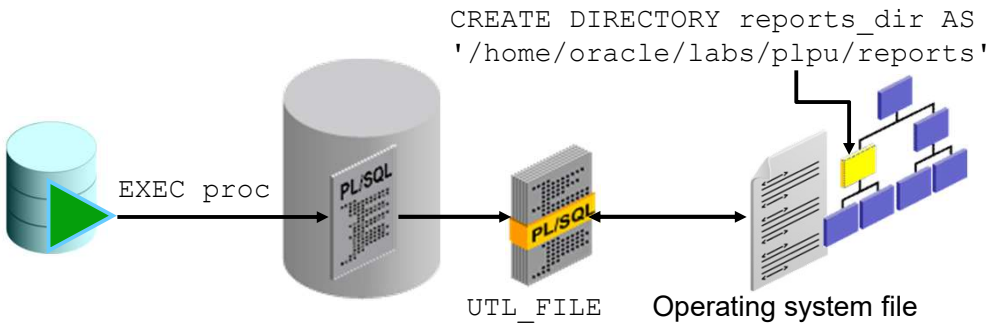
You can output results to the window for debugging purposes. You can trace a code execution path for a function or procedure. You can send messages between subprograms and triggers.

Note: There is no mechanism to flush output during the execution of a procedure.

Using the UTL_FILE Package to Interact with Operating System Files

The UTL_FILE package extends PL/SQL programs to read and write operating system text files:

- Provides a restricted version of operating system stream file I/O for text files
- Can access files in operating system directories defined by a CREATE DIRECTORY statement



ORACLE

Interacting with Operating System Files

The Oracle-supplied UTL_FILE package is used to access text files in the operating system of the database server. The database provides read and write access to specific operating system directories by using:

- A CREATE DIRECTORY statement that associates an alias with an operating system directory. The database directory alias can be granted the READ and WRITE privileges to control the type of access to files in the operating system. For example:

```
CREATE DIRECTORY my_dir AS '/temp/my_files';
GRANT READ, WRITE ON DIRECTORY my_dir TO public;
```

- The paths specified in the utl_file_dir database initialization parameter

It is recommended that you use the CREATE DIRECTORY feature instead of UTL_FILE_DIR for directory access verification. Directory objects offer more flexibility and granular control to the UTL_FILE application administrator, can be maintained dynamically (that is, without shutting down the database), and are consistent with other Oracle tools. The CREATE DIRECTORY privilege is granted only to SYS and SYSTEM by default.

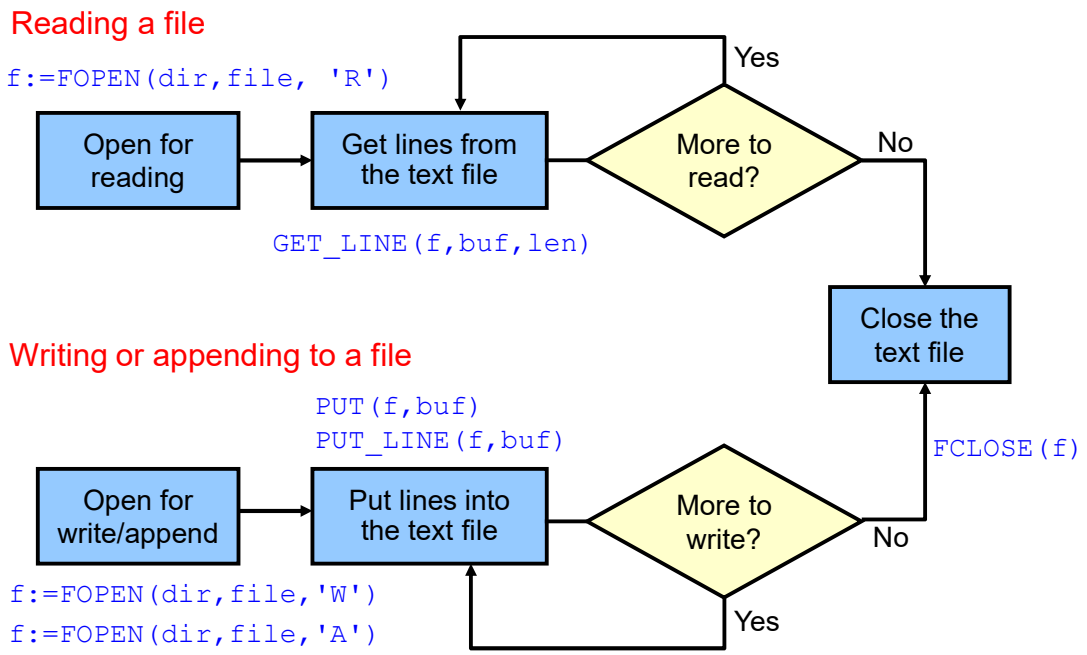
The operating system directories specified by using either of these techniques should be accessible to and on the same machine as the database server processes. The path (directory) names may be case-sensitive for some operating systems.

Some of the UTL_FILE Procedures and Functions

Subprogram	Description
ISOPEN function	Determines if a file handle refers to an open file
FOPEN function	Opens a file for input or output
FCLOSE function	Closes all open file handles
FCOPY procedure	Copies a contiguous portion of a file to a newly created file
FGETATTR procedure	Reads and returns the attributes of a disk file
GET_LINE procedure	Reads text from an open file
FREMOVE procedure	Deletes a disk file, if you have sufficient privileges
FRENAME procedure	Renames an existing file to a new name
PUT procedure	Writes a string to a file
PUT_LINE procedure	Writes a line to a file, and so appends an operating system-specific line terminator

The table in the slide lists some of the UTL_FILE Package subprograms. For a complete list of the package's subprograms, see *Oracle Database PL/SQL Packages and Types Reference.*

File Processing Using the UTL_FILE Package: Overview



You can use the procedures and functions in the `UTL_FILE` package to open files with the `FOPEN` function. You can then either read from or write or append to the file until processing is done. After you finish processing the file, close the file by using the `FCLOSE` procedure. The following are the subprograms:

- The `FOPEN` function opens a file in a specified directory for input/output (I/O) and returns a file handle used in subsequent I/O operations.
- The `IS_OPEN` function returns a Boolean value whenever a file handle refers to an open file. Use `IS_OPEN` to check whether the file is already open before opening the file.
- 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 unless you specify a larger size in the overloaded version of `FOPEN`.)
- The `PUT` and `PUT_LINE` procedures write text to the opened file.
- The `PUTF` procedure provides formatted output with two format specifiers: `%s` to substitute a value into the output string and `\n` for a new line character.
- The `NEW_LINE` procedure terminates a line in an output file.
- The `FFLUSH` procedure writes all data buffered in memory to a file.
- The `FCLOSE` procedure closes an opened file.
- The `FCLOSE_ALL` procedure closes all opened file handles for the session.

Using the Available Declared Exceptions in the UTL_FILE Package

Exception Name	Description
INVALID_PATH	File location invalid
INVALID_MODE	The open_mode parameter in FOPEN is invalid
INVALID_FILEHANDLE	File handle invalid
INVALID_OPERATION	File could not be opened or operated on as requested
READ_ERROR	Operating system error occurred during the read operation
WRITE_ERROR	Operating system error occurred during the write operation
INTERNAL_ERROR	Unspecified PL/SQL error

The UTL_FILE package declares fifteen exceptions that indicate an error condition in the operating system file processing. You may have to handle one of these exceptions when using UTL_FILE subprograms.

A subset of the exceptions are displayed in the slide. For additional information about the remaining exceptions, refer to *Oracle Database PL/SQL Packages and Types Reference*.

Note: These exceptions must always be prefixed with the package name. UTL_FILE procedures can also raise predefined PL/SQL exceptions such as NO_DATA_FOUND or VALUE_ERROR.

The NO_DATA_FOUND exception is raised when reading past the end of a file by using UTL_FILE.GET_LINE or UTL_FILE.GET_LINES.

FOPEN and IS_OPEN Functions: Example

- This FOPEN function opens a file for input or output.

```
FUNCTION FOPEN (p_location  IN VARCHAR2,
                p_filename  IN VARCHAR2,
                p_open_mode IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

- The IS_OPEN function determines whether a file handle refers to an open file.

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

The parameters include the following:

- `p_location` parameter: Specifies the name of a directory alias defined by a `CREATE DIRECTORY` statement, or an operating system–specific path specified by using the `utl_file_dir` database parameter
- `p_filename` parameter: Specifies the name of the file, including the extension, without any path information
- `open_mode` string: Specifies how the file is to be opened. Values are:
  - `'R'` for reading text (use `GET_LINE`)
  - `'W'` for writing text (`PUT`, `PUT_LINE`, `NEW_LINE`, `PUTF`, `FFLUSH`)
  - `'A'` for appending text (`PUT`, `PUT_LINE`, `NEW_LINE`, `PUTF`, `FFLUSH`)
- The return value from `FOPEN` is a file handle whose type is `UTL_FILE.FILE_TYPE`.

ORACLE

The return value from `FOPEN` is a file handle whose type is `UTL_FILE.FILE_TYPE`. The handle must be used on subsequent calls to routines that operate on the opened file.

The `IS_OPEN` function parameter is the file handle. The `IS_OPEN` function tests a file handle to see whether it identifies an opened file. It returns a Boolean value of `TRUE` if the file has been opened; otherwise it returns a value of `FALSE` indicating that the file has not been opened. The example in the slide shows how to combine the use of the two subprograms. For the full syntax, refer to *Oracle Database PL/SQL Packages and Types Reference*.

Ensure that the external file and the database are on the same PC.

```
CREATE OR REPLACE PROCEDURE read_file(p_dir VARCHAR2, p_filename
   VARCHAR2) IS
  f_file UTL_FILE.FILE_TYPE;
  buffer VARCHAR2(200);
  lines  PLS_INTEGER := 0;
BEGIN
  DBMS_OUTPUT.PUT_LINE(' Start ');
  IF NOT UTL_FILE.IS_OPEN(f_file) THEN
    DBMS_OUTPUT.PUT_LINE(' Open ');
    f_file := UTL_FILE.FOPEN (p_dir, p_filename, 'R');
    DBMS_OUTPUT.PUT_LINE(' Opened ');
    BEGIN
      LOOP
        UTL_FILE.GET_LINE(f_file, buffer);
        lines := lines + 1;
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(lines, '099')||' '||buffer);
      END LOOP;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE(' ** End of File **');
    END;
    DBMS_OUTPUT.PUT_LINE(lines||' lines read from file');
    UTL_FILE.FCLOSE(f_file);
  END IF;
END read_file;
/
SHOW ERRORS
SET SERVEROUTPUT ON
EXECUTE read_file('REPORTS_DIR', 'instructor.txt')
```

The partial output of the above code is as follows:

```
PROCEDURE READ_FILE compiled
No Errors.
anonymous block completed
 Start
 Open
 Opened
 001 SALARY REPORT: GENERATED ON
 002                                08-MAR-01
 003
 004 DEPARTMENT: 10
 005   EMPLOYEE: Whalen earns: 4400
 006 DEPARTMENT: 20
 007   EMPLOYEE: Hartstein earns: 13000
 008   EMPLOYEE: Fay earns: 6000
 009 DEPARTMENT: 30
```

...

```
 120 DEPARTMENT: 110
 121   EMPLOYEE: Higgins earns: 12000
 122   EMPLOYEE: Gietz earns: 8300
 123   EMPLOYEE: Grant earns: 7000
 124 *** END OF REPORT ***
 ** End of File **
124 lines read from file
```

Using UTL_FILE: Example

```
CREATE OR REPLACE PROCEDURE sal_status(
  p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
  f_file UTL_FILE.FILE_TYPE;
  CURSOR cur_emp 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
  f_file:= UTL_FILE.FOPEN (p_dir, p_filename, 'W');
  UTL_FILE.PUT_LINE(f_file,
    'REPORT: GENERATED ON ' || SYSDATE);
  UTL_FILE.NEW_LINE (f_file);
. . .
```


In the slide example, the `sal_status` procedure creates a report of employees for each department, along with their salaries. The data is written to a text file by using the `UTL_FILE` package. In the code example, the `file` variable is declared as `UTL_FILE.FILE_TYPE`, a package type that is a record with a field called `ID` of the `BINARY_INTEGER` data type. For example:

```
TYPE file_type IS RECORD (id BINARY_INTEGER);
```

The field of `FILE_TYPE` record is private to the `UTL_FILE` package and should never be referenced or changed. The `sal_status` procedure accepts two parameters:

- The `p_dir` parameter for the name of the directory in which to write the text file
- The `p_filename` parameter to specify the name of the file

For example, to call the procedure, use the following after ensuring that the external file and the database are on the same PC:

```
EXECUTE sal_status('REPORTS_DIR', 'salreport2.txt')
```

Note: The directory location used (`REPORTS_DIR`) must be in uppercase characters if it is a directory alias created by a `CREATE DIRECTORY` statement. When reading a file in a loop, the loop should exit when it detects the `NO_DATA_FOUND` exception. The `UTL_FILE` output is sent synchronously. A `DBMS_OUTPUT` procedure does not produce an output until the procedure is completed.

Using UTL_FILE: Example

```
. . .
FOR emp_rec IN cur_emp LOOP
    IF emp_rec.department_id <> v_olddeptno THEN
      UTL_FILE.PUT_LINE (f_file,
       'DEPARTMENT: ' || emp_rec.department_id);
     UTL_FILE.NEW_LINE (f_file);
    END IF;
    UTL_FILE.PUT_LINE (f_file,
       '  EMPLOYEE: ' || emp_rec.last_name ||
            ' earns: ' || emp_rec.salary);
    v_olddeptno := emp_rec.department_id;
    UTL_FILE.NEW_LINE (f_file);
  END LOOP;
  UTL_FILE.PUT_LINE(f_file,'*** END OF REPORT ***');
  UTL_FILE.FCLOSE (f_file);
EXCEPTION
 WHEN UTL_FILE.INVALID_FILEHANDLE THEN
  RAISE_APPLICATION_ERROR(-20001,'Invalid File.');
 WHEN UTL_FILE.WRITE_ERROR THEN
  RAISE_APPLICATION_ERROR (-20002, 'Unable to write to file');
END sal_status;/
```

ORACLE

Execute the `sal_status` procedure:

```
EXECUTE sal_status('REPORTS_DIR', 'salreport2.txt')
```

The following is a sample of the salreport2.txt output file:

salreport2.txt

```
REPORT: GENERATED ON 19-NOV-12

DEPARTMENT: 10

  EMPLOYEE: Whalen earns: 4400

DEPARTMENT: 20

  EMPLOYEE: Hartstein earns: 13000

  EMPLOYEE: Fay earns: 6000

DEPARTMENT: 30

  EMPLOYEE: Baida earns: 2900

  EMPLOYEE: Khoo earns: 3100
```

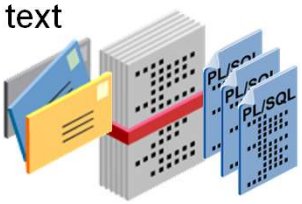
What Is the UTL_MAIL Package?

- The UTL_MAIL package is a utility for managing email that includes commonly used email features such as attachments, CC, BCC, and return receipt.
- The UTL_MAIL package is not installed by default because of the SMTP_OUT_SERVER configuration requirement and the security exposure this involves.
- When installing UTL_MAIL, you should take steps to prevent the port defined by SMTP_OUT_SERVER being swamped by data transmissions.

- To install UTL_MAIL, log in as a DBA user in SQL*Plus and execute the following scripts:
  - @$ORACLE_HOME/rdbms/admin/utlmail.sql
  - @$ORACLE_HOME/rdbms/admin/prvtmail.plb

What Is the UTL_MAIL Package?

- A utility for managing email
- Requires the setting of the SMTP_OUT_SERVER database initialization parameter
- Provides the following procedures:
  - SEND for messages without attachments
  - SEND_ATTACH_RAW for messages with binary attachments
  - SEND_ATTACH_VARCHAR2 for messages with text attachments

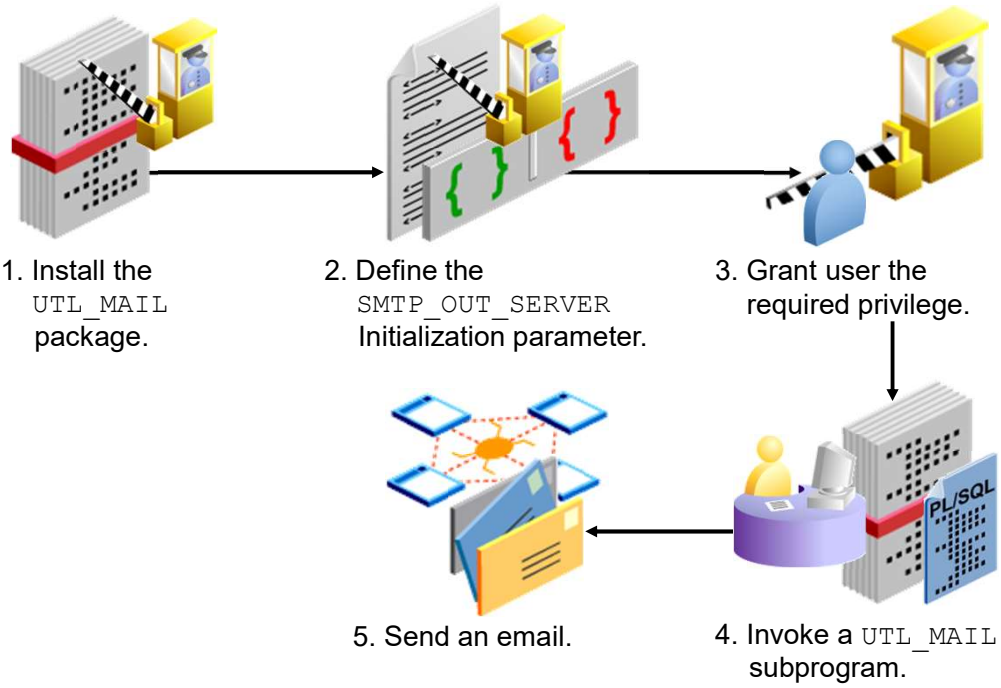


- Due to firewall restrictions, the UTL_MAIL examples in this lesson cannot be demonstrated; therefore, no labs were designed to use UTL_MAIL.

The SMTP_OUT_SERVER parameter specifies the SMTP host and port to which UTL_MAIL delivers outbound email. Multiple servers can be specified, separated by commas. If the first server in the list is unavailable, then UTL_MAIL tries the second server, and so on. If SMTP_OUT_SERVER is not defined, then this invokes a default setting derived from DB_DOMAIN, which is a database initialization parameter specifying the logical location of the database within the network structure. For example:

```
db_domain=mydomain.com
```

Setting Up and Using the UTL_MAIL: Overview



In Oracle Database, the UTL_MAIL package is now an invoker's rights package and the invoking user will need the connect privilege granted in the access control list assigned to the remote network host to which he wants to connect. The Security Administrator performs this task.

Note

- For information about how a user with SYSDBA capabilities grants a user the required fine-grained privileges required for using this package, refer to the "Managing Fine-Grained Access to External Network Services" topic in *Oracle Database Security Guide* and the *Oracle Database Advanced PL/SQL* instructor-led training course.
- Due to firewall restrictions, the UTL_MAIL examples in this lesson cannot be demonstrated; therefore, no labs were designed to use UTL_MAIL.

Summary of UTL_MAIL Subprograms

Subprogram	Description
SEND procedure	Packages an email message, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients
SEND_ATTACH_RAW Procedure	Represents the SEND procedure overloaded for RAW attachments
SEND_ATTACH_VARCHAR2 Procedure	Represents the SEND procedure overloaded for VARCHAR2 attachments

Installing and Using UTL_MAIL

- As SYSDBA, using SQL Developer or SQL*Plus:
  - Install the UTL_MAIL package

```
@?/rdbms/admin/utlmail.sql
@?/rdbms/admin/prvtmail.plb
```

  - Set the SMTP_OUT_SERVER

```
ALTER SYSTEM SET SMTP_OUT_SERVER='smtp.server.com'
SCOPE=SPFILE
```

- As a developer, invoke a UTL_MAIL procedure:

```
BEGIN
 UTL_MAIL.SEND('otn@oracle.com','user@oracle.com',
   message => 'For latest downloads visit OTN',
   subject => 'OTN Newsletter');
END;
```

ORACLE

The slide shows how to configure the SMTP_OUT_SERVER parameter to the name of the SMTP host in your network, and how to install the UTL_MAIL package that is not installed by default. Changing the SMTP_OUT_SERVER parameter requires restarting the database instance. These tasks are performed by a user with SYSDBA capabilities.

The last example in the slide shows the simplest way to send a text message by using the UTL_MAIL.SEND procedure with at least a subject and a message. The first two required parameters are the following :

- The sender email address (in this case, otn@oracle.com)
- The recipients email address (for example, user@oracle.com). The value can be a comma-separated list of addresses.

The UTL_MAIL.SEND procedure provides several other parameters, such as cc, bcc, and priority with default values, if not specified. In the example, the message parameter specifies the text for the email, and the subject parameter contains the text for the subject line. To send an HTML message with HTML tags, add the mime_type parameter (for example, mime_type=>'text/html').

Note: For details about all the UTL_MAIL procedure parameters, refer to *Oracle Database PL/SQL Packages and Types Reference*.

SQL>show parameter smtp_out_server

Utlmail.sql

Prvtmail.plb (privatemail)

Run the above scripts in sys user account

SQL>ALTER SESSION SET SMTP_OUT_SERVER = "127.0.0.1"';

Or

SQL>ALTER SESSION SET SMTP_OUT_SERVER = 'mailhost.oracle.com';

```
CREATE OR REPLACE PROCEDURE test_mail IS
BEGIN
  UTL_MAIL.send(
  sender      => 'arif@iitcoman.com',
  recipients  => 'arif@gmail.com',
  cc          => 'ahmed@gmail.com',
  bcc         => 'said@gmail.com',
  subject     => 'Test message',
  message     =>
  ‘Sending email in pl/sql test’);
END;
/
```

Having 2 mail servers

```
create or replace
procedure my_mailer(p_recip varchar2, p_mode varchar2) is
begin
  --
  -- blah blah
  --

  if p_mode = 'internal' then
    execute immediate
       'alter session set smtp_out_server = ''mailhost1.oracle.com''';
  elsif p_mode = 'internal' then
    execute immediate
       'alter session set smtp_out_server = ''mailhost2.oracle.com''';
  end if;

  utl_mail.send(
    sender=>'connor@oracle.com',
    recipients=>p_recip,
    subject=>'My Subject',
    message=>'My Message');
end;
/
```

The SEND Procedure Syntax

Packages an email message into the appropriate format, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients.

```
UTL_MAIL.SEND (
   sender       IN    VARCHAR2 CHARACTER SET ANY_CS,
   recipients   IN    VARCHAR2 CHARACTER SET ANY_CS,
   cc           IN    VARCHAR2 CHARACTER SET ANY_CS
                        DEFAULT NULL,
   bcc          IN    VARCHAR2 CHARACTER SET ANY_CS
                        DEFAULT NULL,
   subject      IN    VARCHAR2 CHARACTER SET ANY_CS
                        DEFAULT NULL,
   message      IN    VARCHAR2 CHARACTER SET ANY_CS,
   mime_type    IN    VARCHAR2
                        DEFAULT 'text/plain; charset=us-ascii',
   priority     IN    PLS_INTEGER DEFAULT NULL);
```

ORACLE

The SEND Procedure

This procedure packages an email message into the appropriate format, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients. It hides the SMTP API and exposes a one-line email facility for ease of use.

The SEND Procedure Parameters

- **sender:** The email address of the sender.
- **recipients:** The email addresses of the recipient(s), separated by commas.
- **cc:** The email addresses of the CC recipient(s), separated by commas. The default is `NULL`.
- **bcc:** The email addresses of the BCC recipient(s), separated by commas. The default is `NULL`.
- **subject:** A string to be included as email subject string. The default is `NULL`.
- **message:** A text message body.
- **mime_type:** The mime type of the message, default is `'text/plain; charset=us-ascii'`.
- **priority:** The message priority. The default is `NULL`.

The SEND_ATTACH_RAW Procedure

This procedure is the SEND procedure overloaded for RAW attachments.

```
UTL_MAIL.SEND_ATTACH_RAW (
   sender           IN    VARCHAR2 CHARACTER SET ANY_CS,
   recipients       IN    VARCHAR2 CHARACTER SET ANY_CS,
   cc               IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   bcc              IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   subject          IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   message          IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   mime_type        IN    VARCHAR2 DEFAULT CHARACTER SET ANY_CS
                            DEFAULT 'text/plain; charset=us-ascii',
   priority         IN    PLS_INTEGER DEFAULT 3,
   attachment       IN    RAW,
   att_inline       IN    BOOLEAN DEFAULT TRUE,
   att_mime_type    IN    VARCHAR2 CHARACTER SET ANY_CS
                            DEFAULT 'text/plain; charset=us-ascii',
   att_filename     IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```


The SEND_ATTACH_RAW Procedure Parameters

- **sender:** The email address of the sender
- **recipients:** The email addresses of the recipient(s), separated by commas
- **cc:** The email addresses of the CC recipient(s), separated by commas. The default is NULL.
- **bcc:** The email addresses of the BCC recipient(s), separated by commas. The default is NULL.
- **subject:** A string to be included as email subject string. The default is NULL.
- **message:** A text message body
- **mime_type:** The mime type of the message, default is 'text/plain; charset=us-ascii'
- **priority:** The message priority. The default is NULL.
- **attachment:** A RAW attachment
- **att_inline:** Specifies whether the attachment is viewable inline with the message body. The default is TRUE.

Sending Email with a Binary Attachment: Example

```
CREATE OR REPLACE PROCEDURE send_mail_logo IS
BEGIN
  UTL_MAIL.SEND_ATTACH_RAW(
    sender => 'me@oracle.com',
    recipients => 'you@somewhere.net',
    message =>
      '<HTML><BODY>See attachment</BODY></HTML>',
    subject => 'Oracle Logo',
    mime_type => 'text/html'
    attachment => get_image('oracle.gif'),
    att_inline => true,
    att_mime_type => 'image/gif',
    att_filename => 'oralogo.gif');
END;
/
```


The slide shows a procedure calling the `UTL_MAIL.SEND_ATTACH_RAW` procedure to send a textual or an HTML message with a binary attachment. In addition to the `sender`, `recipients`, `message`, `subject`, and `mime_type` parameters that provide values for the main part of the email message, the `SEND_ATTACH_RAW` procedure has the following highlighted parameters:

- The `attachment` parameter (required) accepts a `RAW` data type, with a maximum size of 32,767 binary characters.
- The `att_inline` parameter (optional) is Boolean (default `TRUE`) to indicate that the attachment is viewable with the message body.
- The `att_mime_type` parameter (optional) specifies the format of the attachment. If not provided, it is set to `application/octet`.
- The `att_filename` parameter (optional) assigns any file name to the attachment. It is `NULL` by default, in which case, the name is assigned a default name.

The `get_image` function in the example uses a `BFILE` to read the image data. Using a `BFILE` requires creating a logical directory name in the database by using the `CREATE DIRECTORY` statement. The code for `get_image` is shown on the following page.

The get_image function uses the DBMS_LOB package to read a binary file from the operating system:

```
CREATE OR REPLACE FUNCTION get_image(
     filename VARCHAR2, dir VARCHAR2 := 'TEMP')
RETURN RAW IS
  image RAW(32767);
  file  BFILE := BFILENAME(dir, filename);
BEGIN
  DBMS_LOB.FILEOPEN(file, DBMS_LOB.FILE_READONLY);
  image := DBMS_LOB.SUBSTR(file);
  DBMS_LOB.CLOSE(file);
  RETURN image;
END;
/
```

To create the directory called TEMP, execute the following statement in SQL Developer or SQL*Plus:

```
CREATE DIRECTORY temp AS 'd:\temp';
```

Note

- You need the CREATE ANY DIRECTORY system privilege to execute this statement.
- Due to firewall restrictions at the Oracle Education Center, the examples on this page and the previous page are not available for demonstration.

The SEND_ATTACH_VARCHAR2 Procedure

This procedure is the SEND procedure overloaded for VARCHAR2 attachments.

```
UTL_MAIL.SEND_ATTACH_VARCHAR2 (
   sender             IN    VARCHAR2 CHARACTER SET ANY_CS,
   recipients         IN    VARCHAR2 CHARACTER SET ANY_CS,
   cc                 IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   bcc                IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   subject            IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   message            IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
   mime_type          IN    VARCHAR2 CHARACTER SET ANY_CS
                              DEFAULT 'text/plain; charset=us-ascii',
   priority           IN    PLS_INTEGER DEFAULT 3,
   attachment         IN    VARCHAR2 CHARACTER SET ANY_CS,
   att_inline         IN    BOOLEAN DEFAULT TRUE,
   att_mime_type      IN    VARCHAR2 CHARACTER SET ANY_CS
                              DEFAULT 'text/plain; charset=us-ascii',
   att_filename       IN    VARCHAR2CHARACTER SET ANY_CS DEFAULT NULL);
```

ORACLE

The SEND_ATTACH_VARCHAR2 Procedure Parameters

- **sender:** The email address of the sender
- **recipients:** The email addresses of the recipient(s), separated by commas
- **cc:** The email addresses of the CC recipient(s), separated by commas. The default is NULL.
- **bcc:** The email addresses of the BCC recipient(s), separated by commas. The default is NULL.
- **subject:** A string to be included as email subject string. The default is NULL.
- **Message:** A text message body
- **mime_type:** The mime type of the message, default is 'text/plain; charset=us-ascii'
- **priority:** The message priority. The default is NULL.
- **attachment:** A text attachment
- **att_inline:** Specifies whether the attachment is inline. The default is TRUE.
- **att_mime_type:** The mime type of the attachment, default is 'text/plain; charset=us-ascii'
- **att_filename:** The string specifying a file name containing the attachment. The default is NULL.

Sending Email with a Text Attachment: Example

```
CREATE OR REPLACE PROCEDURE send_mail_file IS
BEGIN
  UTL_MAIL.SEND_ATTACH_VARCHAR2(
    sender => 'me@oracle.com',
    recipients => 'you@somewhere.net',
    message =>
      '<HTML><BODY>See attachment</BODY></HTML>',
    subject => 'Oracle Notes',
    mime_type => 'text/html'
    attachment => get_file('notes.txt'),
    att_inline => false,
    att_mime_type => 'text/plain',
    att_filename => 'notes.txt');
END;
/
```


Sending Email with a Text Attachment

The slide shows a procedure that calls the `UTL_MAIL.SEND_ATTACH_VARCHAR2` procedure to send a textual or an HTML message with a text attachment. In addition to the `sender`, `recipients`, `message`, `subject`, and `mime_type` parameters that provide values for the main part of the e-mail message, the `SEND_ATTACH_VARCHAR2` procedure has the following parameters highlighted:

- The `attachment` parameter (required) accepts a `VARCHAR2` data type with a maximum size of 32,767 binary characters.
- The `att_inline` parameter (optional) is a Boolean (default `TRUE`) to indicate that the attachment is viewable with the message body.
- The `att_mime_type` parameter (optional) specifies the format of the attachment. If not provided, it is set to `application/octet`.
- The `att_filename` parameter (optional) assigns any file name to the attachment. It is `NULL` by default, in which case, the name is assigned a default name.

The `get_file` function in the example uses a `BFILE` to read a text file from the operating system directories for the value of the `attachment` parameter, which could simply be populated from a `VARCHAR2` variable. The code for `get_file` is shown on the following page.

The get_file function uses the DBMS_LOB package to read a binary file from the operating system, and uses the UTL_RAW package to convert the RAW binary data into readable text data in the form of a VARCHAR2 data type:

```
CREATE OR REPLACE FUNCTION get_file(
  filename VARCHAR2, dir VARCHAR2 := 'TEMP')
RETURN VARCHAR2 IS
  contents VARCHAR2(32767);
  file  BFILE := BFILENAME(dir, filename);
BEGIN
  DBMS_LOB.FILEOPEN(file, DBMS_LOB.FILE_READONLY);
  contents := UTL_RAW.CAST_TO_VARCHAR2(
                  DBMS_LOB.SUBSTR(file));
  DBMS_LOB.CLOSE(file);
  RETURN contents;
END;
/
```

Note: Alternatively, you could read the contents of the text file into a VARCHAR2 variable by using the UTL_FILE package functionality.

The preceding example requires the TEMP directory to be created similar to the following statement in SQL*Plus:

```
CREATE DIRECTORY temp AS '/temp';
```

Note

- The CREATE ANY DIRECTORY system privilege is required to execute this statement.
- Due to firewall restrictions at the Oracle Education Center, the examples on this page and the previous page are not available for demonstration.

Quiz

The Oracle-supplied `UTL_FILE` package is used to access text files in the operating system of the database server. The database provides functionality through directory objects to allow access to specific operating system directories.

a. True
b. False

Answer: a

The Oracle-supplied `UTL_FILE` package is used to access text files in the operating system of the database server. The database provides read and write access to specific operating system directories by using:

- A `CREATE DIRECTORY` statement that associates an alias with an operating system directory. The database directory alias can be granted the `READ` and `WRITE` privileges to control the type of access to files in the operating system.
- The paths specified in the `utl_file_dir` database initialization parameter

Summary

In this lesson, you should have learned:

- How the DBMS_OUTPUT package works
- How to use UTL_FILE to direct output to operating system files
- About the main features of UTL_MAIL

This lesson covers a small subset of packages provided with the Oracle database. You have extensively used DBMS_OUTPUT for debugging purposes and displaying procedurally generated information on the screen in SQL*Plus.

In this lesson, you should have learned how to use the power features provided by the database to create text files in the operating system by using UTL_FILE. You also learned how to send
email with or without binary or text attachments by using the UTL_MAIL package.