**UTL\_FILE**

**FOPEN Function**

This function opens a file for input or output. The file location must be an accessible directory, as defined in the instance’s initialization parameter UTL\_FILE\_DIR. The complete directory path must already exist; it is not created by FOPEN. FOPEN returns a file handle, which must be used in all subsequent I/O operations on the file.

This version of FOPEN does not take a parameter for the maximum line size. Thus, the default (which is 1023 on most systems) is used. To specify a different maximum line size, use the other, overloaded version of "FOPEN Function". You can have a maximum of 50 files open simultaneously.

Accessible directories for the UTL\_FILE functions are specified in the initialization file using the UTL\_FILE\_DIR parameter.

Set the UTL\_FILE\_DIR to \* or any other valid path say(‘D:\photos’) which you want to access through UTL\_FILE. The parameter specification UTL\_FILE\_DIR = \* should be used with caution. It turns off directory access checking and makes all directories accessible to the UTL\_FILE functions.

To set the value for UTL\_FILE\_DIR, go to the Menu “Database Control-ORCL” to start Oracle Enterprise Manager, connect as user “sys as sysdba”, click on Administration, under Database Configuration click on ‘All Initialization Parameters’, click on SPFILE, select the parameter ‘utl\_file\_dir’ and set it to a value(\* or D:\photos), click on Apply. Logout of the Enterprise manager. Since this parameter is not dynamic, that is it is a static parameter, you need to restart the database to make it applicable.

OR

alter system set UTL\_FILE\_DIR='d:\hemant' scope=spfile;

and restart the database

show parameter UTL\_FILE\_DIR;

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','abcd','w');

UTL\_FILE.FCLOSE(a);

end;

/

A file abcd is created with size 0.

**Syntax**

UTL\_FILE.FOPEN (

location IN VARCHAR2,

filename IN VARCHAR2,

open\_mode IN VARCHAR2)

RETURN UTL\_FILE.FILE\_TYPE;

location Operating system-specific string that specifies the directory in which to open the file.

filename Name of the file, including extension (file type), without any directory path information. (Under the UNIX operating system, the filename cannot be terminated with a /).

open\_mode String that specifies how the file should be opened (either upper or

lower case letters can be used).

The supported values, and the UTL\_FILE procedures that can be

used with them are:

r read text (GET\_LINE)

w write text (PUT, PUT\_LINE, NEW\_LINE, PUTF, FFLUSH)

a append text (PUT, PUT\_LINE, NEW\_LINE, PUTF, FFLUSH)

If you open a file that does not exist using the **a** value for open\_mode, then the file is created in write (w) mode.

FOPEN returns a file handle, which must be passed to all subsequent procedures that operate on that file.

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',8000);

UTL\_FILE.FCLOSE(a);

end;

/

Here, we have used FOPEN with a fourth parameter, which specifies the max\_linesize, that is the number of characters per line including the newline character. (minimum value 1, maximum value 32767).

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',8000);

UTL\_FILE.put(a, 'hello');

UTL\_FILE.FCLOSE(a);

end;

/

Will create a file xyz and put the string ‘hello’ in it. If the file is not existing it will be created. If the file is already existing, then it is deleted and a new one is created

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','a',8000);

UTL\_FILE.put(a, 'hello man');

UTL\_FILE.FCLOSE(a);

end;

/

Here, if the file is not existing, then it is created. If the file is existing, it is opened in the ‘APPEND’ mode, and the contents ‘hello man’ are appended on a new line.

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',8000);

UTL\_FILE.put(a, 'hello');

UTL\_FILE.put(a, ' how are you');

UTL\_FILE.put(a, ' I am fine');

UTL\_FILE.FCLOSE(a);

end;

/

Here, the file is created and the contents are added to it(all on a single line), and a new-line character is added after the last PUT.

**PUT Procedure**

PUT writes the text string stored in the buffer parameter to the open file identified by the file handle. The file must be open for write operations. No line terminator is appended by PUT ; use NEW\_LINE to terminate the line or use PUT\_LINE to write a complete line with a line terminator.

The maximum size of an input record is 1023 bytes, unless you specify a larger size in the overloaded version of FOPEN.

UTL\_FILE.PUT (file IN FILE\_TYPE, buffer IN VARCHAR2);

file Active file handle returned by an FOPEN, FOPEN\_NCHAR call.

Buffer Buffer that contains the text to be written to the file.

You must have opened the file using mode w or mode a;

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',8000);

UTL\_FILE.put(a, 'hello');

UTL\_FILE.new\_line(a,1);

UTL\_FILE.put(a, ' how are you');

UTL\_FILE.new\_line(a,2);

UTL\_FILE.put(a, ' I am fine');

UTL\_FILE.FCLOSE(a);

end;

/

Here, the file is created and the contents are added to it. After ‘hello’, a line-terminator is added, and ‘how are you’ is then put to this new line, and then 2 line-terminators are added and then ‘I am fine is’ is put to the new line

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',8000);

UTL\_FILE.put(a, 'hello');

UTL\_FILE.new\_line(a);

UTL\_FILE.put(a, ' how are you');

UTL\_FILE.new\_line(a);

UTL\_FILE.put(a, ' I am fine');

UTL\_FILE.FCLOSE(a);

end;

/

If, the second parameter for new\_line is not specified, then it is taken as 1.

**NEW\_LINE Procedure**

This procedure writes one or more line terminators to the file identified by the input file handle. This procedure is separate from PUT because the line terminator is a platform-specific character or sequence of characters.

UTL\_FILE.NEW\_LINE ( file IN FILE\_TYPE, lines IN NATURAL := 1);

file Active file handle returned by an FOPEN call.

lines Number of line terminators to be written to the file.

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\hemant','xyz','w',8000);

UTL\_FILE.put(a, 'hello');

UTL\_FILE.put(a, 'how are you?');

end;

/

Here, the file is not closed, hence nothing gets written to it, and it’s size remain 0 bytes.

All data that you write to the file is buffered, and then written to the file. If huge amount of data is being written to a file, then the process of buffering and writing may take some time. If there is buffered data yet to be written when FCLOSE runs, then you may receive a WRITE\_ERROR exception when closing a file.

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',9000);

for i in 1..8998 loop

UTL\_FILE.put(a, 'h');

end loop;

UTL\_FILE.fclose(a);

end;

/

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',9000);

for i in 1..8999 loop

UTL\_FILE.put(a, 'h');

end loop;

UTL\_FILE.fclose(a);

Exception

When others then

Dbms\_output.put\_line(sqlerrm);

end;

/

**FCLOSE Procedure**

This procedure closes an open file identified by a file handle.

UTL\_FILE.FCLOSE ( file IN OUT FILE\_TYPE);

file Active file handle returned by an FOPEN or FOPEN\_NCHAR call.

declare

a utl\_file.file\_type;

b utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',8000);

b:=UTL\_FILE.FOPEN ('d:\photos','lmn','w',8000);

UTL\_FILE.put(a, 'hello');

UTL\_FILE.put(b, 'how are you?');

UTL\_FILE.FCLOSE\_ALL;

end;

/

**FCLOSE\_ALL Procedure**

This procedure closes all open file handles for the session. This should be used as an emergency cleanup procedure, for example, when a PL/SQL program exits on an exception. The buffered data is written to the respective files.

FCLOSE\_ALL does not alter the state of the open file handles held by the user. This means that an IS\_OPEN test on a file handle after an FCLOSE\_ALL call still returns TRUE, even though the file has been closed. No further read or write operations can be performed on a file that was open before an FCLOSE\_ALL.

Exec UTL\_FILE.FCLOSE\_ALL;

declare

a utl\_file.file\_type;

aa boolean;

b utl\_file.file\_type;

bb boolean;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',8000);

b:=UTL\_FILE.FOPEN ('d:\photos','lmn','w',8000);

UTL\_FILE.put(a, 'hello');

UTL\_FILE.put(b, 'how are you?');

UTL\_FILE.FCLOSE(a);

aa:=UTL\_FILE.is\_open(a);

if aa then

dbms\_output.put\_line('file a is open');

UTL\_FILE.put(a, 'I am fine?');

else

dbms\_output.put\_line('file a is closed');

end if;

UTL\_FILE.FCLOSE\_All;

bb:=UTL\_FILE.is\_open(b);

if bb then

dbms\_output.put\_line('file b is open');

UTL\_FILE.put(b, 'I am fine?'); --not added to file lmn

else

dbms\_output.put\_line('file b is closed');

end if;

end;

/

Output :

file a is closed

file b is open

The string ‘I am fine’ is not added to the file lmn

declare

a utl\_file.file\_type;

aa boolean;

begin

aa:=UTL\_FILE.IS\_OPEN(a);

if aa then

dbms\_output.put\_line('file a is open');

UTL\_FILE.put(a, 'I am fine?');

else

dbms\_output.put\_line('file a is closed');

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',8000);

UTL\_FILE.put(a, 'I am fine?');

end if;

UTL\_FILE.FCLOSE(a);

aa:=UTL\_FILE.IS\_OPEN(a);

if aa then

dbms\_output.put\_line('file a is open');

else

dbms\_output.put\_line('file a is closed');

end if;

end;

/

Output :

file a is closed

file a is closed

**IS\_OPEN Function**

This function tests a file handle to see if it identifies an open file. IS\_OPEN reports only whether a file handle represents a file that has been opened.

Assume a file xyz.txt containing the following :

Hello

How are you

I am Fine

Thank you.

declare

a utl\_file.file\_type;

buf varchar2(1000);

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz.txt','r');

UTL\_FILE.GET\_LINE(a,buf);

dbms\_output.put\_line(buf);

end;

/

This will print the first line of the file xyz.txt

Output :

Hello

**GET\_LINE Procedure**

This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. Text is read up to, but not including, the line terminator, or up to the end of the file.

If the line does not fit in the buffer, then a VALUE\_ERROR exception is raised. If no text was read due to end of file, then the NO\_DATA\_FOUND exception is raised.

declare

a utl\_file.file\_type;

buf varchar2(3);

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz.txt','r');

UTL\_FILE.GET\_LINE(a,buf);

dbms\_output.put\_line(buf);

exception

when others then

dbms\_output.put\_line(sqlerrm);

end;

/

declare

a utl\_file.file\_type;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','w',8000);

End;

/

This creates a file xyz with 0 bytes

Now we try to read from the above created xyz file which contains no text. Hence no\_data\_found exception is raised

declare

a utl\_file.file\_type;

buf varchar2(1000);

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz','r');

UTL\_FILE.GET\_LINE(a,buf);

exception

when no\_data\_found then

dbms\_output.put\_line('no text to be read');

End;

/

Assume a file xyz.txt containing the following :

Hello

How are you

I am Fine

Thank you.

declare

a utl\_file.file\_type;

buf varchar2(1000);

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz.txt','r');

UTL\_FILE.GET\_LINE(a,buf);

dbms\_output.put\_line(buf);

while length(buf)>0 loop

UTL\_FILE.GET\_LINE(a,buf);

dbms\_output.put\_line(buf);

end loop;

exception

when no\_data\_found then

dbms\_output.put\_line('no text to be read');

UTL\_FILE.FCLOSE(a);

End;

/

**PUTF Procedure**

This procedure is a formatted PUT procedure. It works like a limited printf(). The format string can contain any text, but the character sequences %s and \n have special meaning.

UTL\_FILE.PUTF ( file IN FILE\_TYPE, format IN VARCHAR2, [arg1 IN VARCHAR2 DEFAULT NULL, . . . arg5 IN VARCHAR2 DEFAULT NULL]);

file Active file handle returned by an FOPEN call.

format string Format string that can contain text as well as the formatting characters \n and %s.

arg1..arg5 From one to five operational argument strings.

Argument strings are substituted, in order, for the %s formatters in the format string. If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each %s for which there is no argument.

declare

a utl\_file.file\_type;

buf varchar2(10):='RAM';

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz.txt','w');

utl\_file.putf(a,'Hello %s, My name is %s\n','friends', buf);

buf:='SHAM';

utl\_file.putf(a,'Hello %s, My name is %s\n','enemies', buf);

utl\_file.fclose(a);

End;

/

declare

a utl\_file.file\_type;

buf varchar2(1000);

cursor c1 is select empno, ename from emp;

begin

a:=UTL\_FILE.FOPEN ('d:\hemant','xyz.txt','w');

for erec in c1 loop

utl\_file.putf(a,'Employee No. :%s is named as %s\n',erec.empno, erec.ename);

end loop;

utl\_file.fclose(a);

End;

/

declare

a utl\_file.file\_type;

buf varchar2(1000);

cursor c1 is select \* from emp;

begin

a:=UTL\_FILE.FOPEN ('d:\hemant','xyz.txt','w');

for erec in c1 loop

utl\_file.putf(a,'Employee No. :%s'||' Name : %s'||' Sal %s'||' Comm : %s'||' Deptno : %s'||' Manager : %s\n',erec.empno, erec.ename, erec.sal, erec.comm,erec.deptno, erec.mgr);

end loop;

utl\_file.fclose(a);

End;

/

This will give an error, since there cannot be more than 5 operational argument strings.

Hence, you may use two PUTFs as follows :

declare

a utl\_file.file\_type;

buf varchar2(1000);

cursor c1 is select \* from emp;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz.txt','w');

for erec in c1 loop

utl\_file.putf(a,'Employee No. :%s'||' Name : %s'||' Sal %s'||' Comm : %s'||' Deptno : %s',erec.empno, erec.ename, erec.sal, erec.comm, erec.deptno);

utl\_file.putf(a, ' Manager : %s\n', erec.mgr);

end loop;

utl\_file.fclose(a);

End;

/

declare

a utl\_file.file\_type;

buf varchar2(1000);

cursor c1 is select empno, ename from emp;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz.txt','w');

for erec in c1 loop

utl\_file.putf(a,'Employee No.%s is named as %s\n',erec.empno, erec.ename);

--utl\_file.fflush(a);

end loop;

-- utl\_file.fclose(a);

End;

/

In the above, no data is written to the file

declare

a utl\_file.file\_type;

buf varchar2(1000);

cursor c1 is select empno, ename from emp;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz.txt','w');

for erec in c1 loop

utl\_file.putf(a,'Employee No.%s is named as %s\n',erec.empno, erec.ename);

--utl\_file.fflush(a);

end loop;

utl\_file.fclose(a);

End;

/

In the above, data is written at the time the file is closed

declare

a utl\_file.file\_type;

buf varchar2(1000);

cursor c1 is select empno, ename from emp;

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz.txt','w');

for erec in c1 loop

utl\_file.putf(a,'Hii Employee No.%s is named as %s\n',erec.empno, erec.ename);

utl\_file.fflush(a);

end loop;

-- utl\_file.fclose(a);

End;

/

In the above, data is written at every flush

create or replace procedure myread is

a utl\_file.file\_type;

buf varchar2(1000);

begin

a:=UTL\_FILE.FOPEN ('d:\photos','xyz.txt','r');

utl\_file.get\_line(a,buf);

dbms\_output.put\_line(buf);

UTL\_FILE.FCLOSE(a);

end;

/

declare

a utl\_file.file\_type;

buf varchar2(1000);

cursor c1 is select empno, ename from emp;

begin

a:=UTL\_FILE.FOPEN ('d:\hemant','xyz.txt','w');

for erec in c1 loop

utl\_file.putf(a,'Employee No.%s is named as %s\n',erec.empno, erec.ename);

utl\_file.fflush(a);

myread;

end loop;

utl\_file.fclose(a);

End;

/

create or replace procedure myread is

type mytype is table of varchar2(100) index by binary\_integer;

a utl\_file.file\_type;

buf mytype;

var number:=1;

begin

a:=UTL\_FILE.FOPEN ('d:\hemant','xyz.txt','r');

LOOP

UTL\_FILE.GET\_LINE (a, buf(var));

dbms\_output.put\_line(buf(var));

var := var + 1;

END LOOP;

UTL\_FILE.FCLOSE(a);

exception

when no\_data\_found then

dbms\_output.put\_line(' over');

end;

/

Create or replace package mypack is

Gctr number:=1;

End;

/

create or replace procedure myread is

type mytype is table of varchar2(100) index by binary\_integer;

a utl\_file.file\_type;

buf mytype;

var number:=1;

begin

a:=UTL\_FILE.FOPEN ('d:\hemant','xyz.txt','r');

LOOP

UTL\_FILE.GET\_LINE (a, buf(var));

If var=mypack.gctr then

dbms\_output.put\_line(buf(var));

end if;

var := var + 1;

END LOOP;

UTL\_FILE.FCLOSE(a);

exception

when no\_data\_found then

mypack.gctr:=var;

dbms\_output.put\_line('over');

end;

/

In the above block, if you comment the line ‘utl\_file.fflush(a);’, then you get an exception ‘no\_data\_found’, since you are writing to the file and also wanting to read from it through the stored procedure ‘myread’. Hence, you use the utl\_file.fflush(a);

**FFLUSH Procedure**

FFLUSH physically writes pending data to the file identified by the file handle. Normally, data being written to a file is buffered. The FFLUSH procedure forces the buffered data to be written to the file. The data must be terminated with a newline character. Hence, in the above code, even if you remove the ‘\n’ character from the PUTF, then also you get the exception ‘no\_data\_found’. Thus, flushing is useful when the file must be read while still open.