

PL/SQL Utilities

Describes a simple Logging API for use in Oracle Databases 8i and above.

March 2005

Contents

Introduction	3
The package.....	3
Exceptions.....	4
How to Use.....	5
Logging.....	8
Definitions.....	11
Impact on processing.....	13
Futures.....	13

Introduction

Oracle databases need to manipulate external interface files and produce log files, telling the outside world what they've done (or failed to do).

Here is a guide to use the package FILE_OPS, now in use at various sites for over 6 years, which provides a **simple** way to read in *small* external files and write out essential log information.

Its ideal use is three fold:

- In producing confirmation logs
(I processed this amount of records in that elapsed time span)
- In providing detailed error logs
(An error occurred and it occurred here)
- In logging activity
(In debug mode, in monitoring large amounts of data)

It's been designed for and used in production jobs, where it has proved invaluable in diagnostics and in job control. It's also very useful when debugging and verifying PL/SQL routines.

The package

This package is a wrapper package for the UTL_FILE procedures and functions.

It provides simple functions for reading from and writing to nominated files.

PL/SQL provides collection and record objects that allow us to store data in temporary storage. It works very much in the way of arrays and structs of arrays found in such languages as C and Pascal or working storage in COBOL.

It uses the PL/SQL TABLE collection object as an array to store data destined for file operations before the read is read or written. This moves any file operations to the start or end of a process (Start to read in the file, end to write out the file), thus improving reliability and error handling. Once stored as TABLE, PL/SQL provides a series of objects that provide easy access to data items using collection methods such as COUNT.

There are two types of table provided:

- Log for log files
- Output for other stuff

Each is indexed by a numeric subscript, uses Oracle's index-by table data type and has a maximum length of 2000 bytes.

Exceptions

Each wrapper is a function; exceptions are handled by the return of FALSE and a user-defined error number. The majority of exceptions will arise because quoted directories are not in the instance's INIT.ORA file (unless UTL_FILE_DIR has been set to * against Oracle's recommendations) or the quoted file name doesn't exist.

There is a hidden function that looks for the UTL_FILE_DIR entry in the V\$PARAMETER table. If found it makes this the default directory. If there are several entries the first one is taken. File permissions usually raise INVALID_OPERATION.

Table of exception codes returned by FILE_OPS

Returned Code	Exception	Note
-20999	UTL_FILE.INVALID_PATH	Does path in INIT.ORA Exist? Is file_ops.l_location correct? Is it the right filename?
-20998	UTL_FILE.INVALID_MODE	Reading when you're trying to write or vice versa
-20997	UTL_FILE.INVALID_FILEHANDLE	
-20996	UTL_FILE.READ_ERROR	
-20995	UTL_FILE.INVALID_OPERATION	Are permissions on directory correctly set?
-20994	UTL_FILE.INTERNAL_ERROR	
-20993	UTL_FILE.WRITE_ERROR	
-20992	FILE_OPS.NOTHING_IN_TABLE	You've passed an empty table to write from

How to Use

You need to declare PL/SQL tables to store the data you want to read from or write to the file system.

As shown here:

```
declare
l_tab    file_ops.out_tab;
l_count  BINARY_INTEGER :=0;
```

`l_tab` is now an array, indexed by `l_count`. You will always want an index number and you will *always* want it initialized to 0. Then, in the body of the code, you say

`l_tab(l_count).out_text := 'Test Record';` to assign a value to the zero'th subscript of the array.

Incrementing `l_count` will move you to next slot in the array.

Reading values is the same process:

```
l_tab(l_count).out_text := 'Test Record';
l_buf := l_tab(l_count).out_text;
dbms_output.put_line(l_buf);
```

but reversed.

All Oracle's collection functions can be applied to these structures.
(See PL/SQL User's Guide and Reference Collections and Records)

If you want to write the contents of the PL/SQL table to a file then add these lines to our example:

```
-- set the filename
file_ops.l_filename := 'TestFileName.txt';
-- and make the call
IF NOT file_ops.write_file(file_ops.l_filename
                          , l_tab
                          , l_a) THEN
    dbms_output.put_line(l_a);
ELSE
    NULL;
END IF;
```

Examine the third parameter for any error conditions.

Here's the example in full:

```
declare
l_tab    file_ops.out_tab;
l_count  BINARY_INTEGER :=0;
l_buf    VARCHAR2(255)   := NULL;
l_a      NUMBER          := 0;
BEGIN
l_tab(l_count).out_text := 'Test Record2';
l_buf := l_tab(l_count).out_text;
dbms_output.put_line(l_buf);
-- set the filename
file_ops.l_filename := 'TestFileName.txt';
-- and make the call
IF NOT file_ops.write_file(file_ops.l_filename
                           ,l_tab
                           ,l_a) THEN
    dbms_output.put_line(l_a);
ELSE
    dbms_output.put_line(l_a);
    NULL;
END IF;
END;
```

Here's the output:

```
Test Record2
0
```

PL/SQL procedure successfully completed.

And here's the contents of the file

```
bash-2.03$ cat TestFileName.txt
Test Record2
```

Note that `l_a` is the record count and that *it begins at zero*

The structure is the same for reading files.

```
declare
l_tab    file_ops.out_tab;
l_count  BINARY_INTEGER :=0;
l_buf    VARCHAR2(128)   := NULL;
l_a      NUMBER(5)       := 0;
begin
file_ops.l_filename := 'TestFileName.txt';
IF NOT file_ops.read_file(file_ops.l_filename
                        ,l_tab
                        ,l_a) THEN
    dbms_output.put_line(l_a);
ELSE
    dbms_output.put_line(l_a);
    dbms_output.put_line(l_tab((l_a-1)).out_text);
END IF;
end;
/
```

And here's the content of the file read back to us:

```
1
Test Record2
```

PL/SQL procedure successfully completed.

Note that the first line of the array is zero (consistent with the write) though l_a represents the number of records in the file; hence (l_a-1).

Logging

In batch processes it is essential that a record of a job's activity is stored somewhere. Storing progress data in Oracle tables is superficially attractive but poses issues for correct operation of commit and rollback, as the logging of progress data may not be safely extricated from the overall logical unit of work.

The intricacies of the PL/SQL package DBMS_OUPUT are well known, the limitation on buffer sizes and return of output only on completion of transaction being two factors that militate against serious usage.

Writing activity to a log array and then writing the contents of that array to a file removes the logging procedure from any work unit considerations and considerably improves logging in error conditions.

It also allows operation engineers to configure existing scheduling and monitoring tools to include the nominated log directory in their configurations. This monitoring can give significant leverage to enforcing process interdependency.

The FILE_OPS package includes some wrapper code to facilitate easy logging.

The default layout uses XML that imposes a simple structure on the file as follows:

```
<LOGFILE>Filename
  <LOG>
    <DATETIME>YYYYMMDDHH24MISS</DATETIME>
    <TEXT>Your 1st log text goes here</TEXT>
  </LOG>
  <LOG>
    <DATETIME>YYYYMMDDHH24MISS</DATETIME>
    <TEXT>Your 2nd log text goes here</TEXT>
  </LOG>
  <LOG>
    <DATETIME>YYYYMMDDHH24MISS</DATETIME>
    <TEXT>Your 3rd log text goes here</TEXT>
  </LOG>
</LOGFILE>
```

You can use other XML tools such as XSLT and XPATH for formatting and searching.

You might wish to place these files where a server can see them.

You can also store these logs in Oracle using the XMLType datatype, which means you can build up a history of logs that can be stored as part of the database.

As with all PL/SQL table usage, you'll need to declare it at the beginning of the code you want to log. Like this:-

```
l_log file_ops.log_tab;  
l_count BINARY_INTEGER := 0;
```

Then to log text to the log table insert the following code :-

```
file_ops.do_log('Your log text here',l_log,l_count);
```

wherever you want to record activity.

The actual log record looks like this:-

```
<LOG><DATETIME>20040622163247</DATETIME><TEXT>Text</TEXT></LOG>
```

You can use text, number or date, although the date field is converted to whatever is set as the default within the target instance. (NLS settings etc...)

It's important to realise that you must write the contents of the log table to the log file when you've finished. There's no check prompting you to do so.

Writing a log table out to file

This fragment shows you how:

```
IF NOT file_ops.write_log(file_ops.get_log_file_name('TestLog')  
                        ,l_log  
                        ,l_a) THEN  
    dbms_output.put_line(l_a);  
ELSE  
    NULL;  
END IF;
```

This is close to the previous call, yet note the addition of the function `get_log_file_name`. This function returns the text given in the parameter with a date string in the form `YYYYMMDDHH24MISS` and the file prefix of `.log` appended. Thus `TestLog` would then become `TestLog20040622170534.log`. The purpose of this function is to guarantee a standard and unique log file name.

The full example looks like this:

```
declare
l_log      file_ops.log_tab;
l_count   BINARY_INTEGER :=0;
l_buf     VARCHAR2(128)  := NULL;
l_a       NUMBER(5)      := 0;
begin
-- text sample
file_ops.do_log('TestFile.txt'
               ,l_log
               ,l_count);
-- number sample
file_ops.do_log(1234567
               ,l_log
               ,l_count);
-- date sample
file_ops.do_log(sysdate
               ,l_log
               ,l_count);
IF NOT file_ops.write_log(file_ops.get_log_file_name('TestLog')
                          ,l_log
                          ,l_a) THEN
    dbms_output.put_line(l_a);
ELSE
    dbms_output.put_line(l_a);
END IF;
end;
```

Note how we're checking for the three basic datatypes of Text, Number and Date.
Here's the PL/SQL output

0

PL/SQL procedure successfully completed.

And here's the contents of the log file:

```
-rw-r--r-- 1 oracle dba          254 Jun 23 09:49 TestLog20040623094938.log
bash-2.03$ cat *938.log
<LOGFILE>TestLog20040623094938.log
<LOG><DATETIME>20040623094938</DATETIME><TEXT>TestFile.txt</TEXT></LOG>
<LOG><DATETIME>20040623094938</DATETIME><TEXT>1234567</TEXT></LOG>
<LOG><DATETIME>20040623094938</DATETIME><TEXT>23-Jun-04</TEXT></LOG>
</LOGFILE>
```

Definitions

```
--
FUNCTION read_file(p_filename  IN VARCHAR2
                  ,p_tab_result OUT out_tab
                  ,p_sqlcode   IN OUT NUMBER) RETURN BOOLEAN;
--
Reads in the contents of the file specified in p_filename into the
PL/SQL table defined by the variable p_tab_result.
If the function returns FALSE then the function has failed and the
value of p_sqlcode will be negative, otherwise the function has been
successful.
--
FUNCTION write_file(p_filename  IN VARCHAR2
                  ,p_tab_result IN out_tab
                  ,p_sqlcode   IN OUT NUMBER) RETURN BOOLEAN;
--
Writes the contents of the PL/SQL table to the filename specified by
the value in p_filename.
The directory in which p_filename resides must be one defined in the
UTL_FILE_DIR directive of the database initialisation file. If no
directory is defined in p_filename, then the V$PARAMETER table is read
with a key of UTL_FILE_DIR and the first value returned is used. If
none is found then an error is returned.
If the function returns FALSE then the function has failed and the
value of p_sqlcode will be negative, otherwise the function has been
successful.
--
PROCEDURE do_log(p_text  IN      VARCHAR2
                ,p_tab   IN OUT log_tab
                ,p_count IN OUT BINARY_INTEGER);
--
Adds the text p_text to the PL/SQL table p_tab and returns the value
p_count incremented by 1
--
PROCEDURE do_log(p_text  IN      NUMBER
                ,p_tab   IN OUT log_tab
                ,p_count IN OUT BINARY_INTEGER);
--
Adds the number p_text to the PL/SQL table p_tab and returns the value
p_count incremented by 1
--
PROCEDURE do_log(p_text  IN      DATE
                ,p_tab   IN OUT log_tab
                ,p_count IN OUT BINARY_INTEGER);
--
Adds the date p_text, in default date format, to the PL/SQL table p_tab
and returns the value p_count incremented by 1
```

```
--
FUNCTION write_log(p_filename IN VARCHAR2
                  ,p_tab_result IN log_tab
                  ,p_sqlcode IN OUT NUMBER) RETURN BOOLEAN;
--
Writes the contents of the PL/SQL table IN TAGGED XML format to the
filename specified by the value in p_filename.
The directory in which p_filename resides must be one defined in the
UTL_FILE_DIR directive of the database initialisation file. If no
directory is defined in p_filename, then the V$PARAMETER table is read
with a key of UTL_FILE_DIR and the first value returned is used. If
none is found then an error is returned.
If the function returns FALSE then the function has failed and the
value of p_sqlcode will be negative, otherwise the function has been
successful.
--
FUNCTION get_log_file_name(p_filename IN VARCHAR2) RETURN VARCHAR2;
--
Returns the value p_filename in the format
<p_filename>YYYYMMDDHH24MISS.log
--
FUNCTION get_err_file_name(p_filename IN VARCHAR2) RETURN VARCHAR2;
--
Returns the value p_filename in the format
<p_filename>YYYYMMDDHH24MISS.err
--
```

Impact on processing

Logging should be used relatively sparingly. Our tests suggest that, in volume situations, excessive use of logging can slow things down by 10-15 per cent or so.

Futures

This package can be extended to handle the DIRECTORY features of 9i and 10g. It should also be possible to stream output to specific ports, which would allow on-line monitoring of PL/SQL processes.