

NAME

process_sqlfile – process sqlfile (what else it could be)

SYNOPSIS**process_sqlfile.ksh**

```
myname:=caller_full_pathname
[login=database_login]
[sql="sql_file_args"]
[log_table=y|n]
[help=y]
```

process_sqlfile.ksh

P1: SQL file to be processed (using quotes with arguments)
P2: Login name to connect to the database
P3: Program directory
P4: Program name

DESCRIPTION

process_sqlfile.ksh processes SQL file for you, and does the error checking.

The new key=val argument style requires only one mandatory argument, myname, which is the program name that calls **process_sqlfile.ksh**. (the rest can be either derived or have default values). However, the caller name is available as "\$0" in the caller's program, this is not user replaceable part. If we can rely on the default, and derivable values, then the caller program is maintenance free. For different jobs, the content of the caller program is identical, only the names differ.

And if you do need to supply additional parameters, it can be in any order. "[:=" is used for case sensitive options, myname and sql, and "=" is used for other options, which are case insensitive.

The old argument style requires exactly 4 arguments, in the order shown above.

The new **process_sqlfile.ksh** is backward compatible, ie, it accepts both new and old argument style. I hope you find the new style is easier to use and maintain.

COMMAND LINE OPTIONS

SQL file is needed in old argument style. It is the SQL filename to be processed. If the SQL file has arguments, the file and arguments should be quoted, like this, "*SQL_file arg1 arg2 ...*".

It is optional in new argument style. If not specified, the SQL file is assumed to have the same name as the caller program, with .sql extension. For example, if caller program is my_job.ksh, the SQL file is assumed to be my_job.sql. It is specified with "sql:=" option. Again use quotes if there are arguments.

Login name is second argument to **process_sqlfile** in old argument style. It is optional in new argument style. If not specified, it is assumed to be the same as immediate directory containing the caller program, as the caller programs are organized by schemas.

"Program directory", and "Program name" are only needed in old argument style. The new argument style do not need these parameters.

"log_table=y|n" specifies whether to load the log table. The info is collected regardless. The use of log table allows to generate a nice looking report (which most managers love to see, and it helps you with getting good bonus). The precedence is the command line option, job configuration file, and the default which is Y (who don't like to get good bonus?).

If you do use log_table, the creation script is

```

create table jobs_log (
  PROGRAM_NAME      VARCHAR2(200) NOT NULL,
  FEED_NAME         VARCHAR2(200),
  HDR_LABEL         VARCHAR2(200),
  TRL_LABEL         VARCHAR2(200),
  TRLR_RECS_RCVD   NUMBER,
  FILE_RECS_COUNT  NUMBER,
  FILE_RCVD_DATE   DATE,
  MODULE_NAME      VARCHAR2(200),
  LOAD_STATUS      NUMBER,
  RECS_LOADED      NUMBER,
  RECS_REJ         NUMBER,
  RECS_DSC         NUMBER,
  LOGFILE          VARCHAR2(200),
  PROCESS_DATE     DATE           NOT NULL,
)

```

“help=y” prints out the man page you are reading.

CONFIGURATION FILE

process_sqlfile.ksh shares the same configuration file with **process_feed.ksh**.

A sample configuration file is shown below.

```

# BASIC ENVIRONMENT VARIABLES FOR ORACLE

typeset -x ORACLE_SID=BCCS1
typeset -x ORACLE_HOME=/bcc-bccs1-s/ora01/app/oracle/product/8.1.7
typeset -x PATH=$(/bin/getconf PATH):/usr/local/bin:$ORACLE_HOME/bin
typeset -x TNS_ADMIN=/var/opt/oracle
typeset -x LD_LIBRARY_PATH=

# USERS AND PASSWORDS

idlog=<login>           # user name and
pwdlog=<passwd>         # passwd for "INSERT INTO JOBS_LOG"
<SCHEMA>PASS=<passwd>  # <SCHEMA> passwd for process feed and sqlfile
pwd=<passwd>           # common passwd in the absense of <SCHEMA>PASS

# EXIT STATUS CODES

```

```

(( SUCCESS = 0 ))
(( FEED_NOT_EXISTS = 1 ))
(( FEED_HDR_NOT_EXISTS = 2 ))
(( FEED_TRL_NOT_EXISTS = 3 ))
(( FEED_COUNT_MISMATCH = 4 ))
(( LOAD_COUNT_ZERO = 5 ))
(( LOAD_COUNT_MISMATCH = 6 ))
#
(( SQL_PROCESSING_ERROR = 9 ))
(( ARG_MISMATCH = 10 ))
(( DIRECTORY_NOT_EXISTS = 11 ))
(( SQLLOAD_EXECUTION_ERROR = 12 ))
(( EXCEED_MAX_ERRORS = 13 ))
(( EXCEED_MAX_DISCARDS = 14 ))

# FEED

INCOMING_DIR=~ftpstars
BACKUP_DIR=~ftpstars/backup
(( DATA_KEEP = -1 )) # days to keep data file (negative number = forever)
(( LOG_KEEP = -1 )) # days to keep log file (negative number = forever)

# NOTIFICATION

JOB_EMAIL="foo@my.com,bar@my.com"
JOB_PAGER="1234567@pager.com,7654321@pager.com"

```

DIRECTORY STRUCTURE

The setup is similar to **process_feed.ksh**.

I described below a preferred directory structure. No absolute path is hard coded in this setup, which makes the program very portable.

First, you setup the top level job directory, *whatever_dir*/jobs, which we will refer as *TOP_DIR*.

Secondly, you setup bin directory under *TOP_DIR*, that is where this program **process_sqlfile.ksh** should reside. And its configuration file should be in *TOP_DIR*/etc and should be named ".jobs_env". It should be read/write by owner only.

Thirdly, the caller programs should be organized by schema. For example,

```
bc_load_del_pending_trades.ksh
bc_load_del_pending_trades.sql
```

are placed under *TOP_DIR*/trade. **process_sqlfile.ksh** will put the log files under:

```
I<TOP_DIR>/trade/bc_load_del_pending_trades:

log/bc_load_del_pending_trades/bc_load_del_pending_trades.02-03_22:41.out
log/bc_load_del_pending_trades/bc_load_del_pending_trades.02-03_22:46.out
log/bc_load_del_pending_trades/bc_load_del_pending_trades.02-03_22:53.out
```

The *.out is the unix stdout and stderr, which includes SQL code and output.

The autosys command line would be:

```
command: I<TOP_DIR>/trade/bc_load_del_pending_trades.ksh
```

SECURITY

The user program does not handle passwd, *process_sqlfile.ksh* retrieve the passwd for the login. The passwd file should be read only by the program, and the program disables trace, and is only modifiable by the owner.

The passwd is not part of sqlplus command arguments and therefore does not show up in ps output.

CVS REPOSITORY

By design, all programs, control files, sql files are under the same top level directory, it is easier to put them under version control system CVS. This avoids the clutter we had on the production system.

Here is an example to commit a change to CVS:

```
$ cvs ci -m "ver 3.0 tested" process_feed.ksh
Checking in process_feed.ksh;
/bcc-bccs1-s/ora01/home/oracle/jobs-repo/jobs/bin/process_feed.ksh,v
+ <-- process_feed.ksh
new revision: 1.6; previous revision: 1.5
done
```

EXAMPLES

New style:

```
$ cat bc_load_del_pending_trades_a.ksh

#!/bin/ksh
[[ $0 == /* ]] || exec $(pwd -P)/$0 "$@"
${0%*/jobs*}/jobs/bin/process_sqlfile.ksh myname:=$0

$ cat bc_load_del_pending_trades_a.sql

delete from pending_trades where trade_date=20011208;

$ cat bc_load_del_pending_trades_b.ksh

#!/bin/ksh
[[ $0 == /* ]] || exec $(pwd -P)/$0 "$@"
${0%*/jobs*}/jobs/bin/process_sqlfile.ksh myname:=$0 sql:="{0%.ksh} 20011208"

$ cat bc_load_del_pending_trades_b.sql

delete from pending_trades where trade_date=&1;
```

Old style:

```
$ cat bc_load_del_pending_trades_b.ksh
```

```
#!/bin/ksh
[[ $0 == /* ]] || exec $(pwd -P)/$0 "$@"
${0%/jobs*}/jobs/bin/process_sqlfile.ksh \
  "${0%.ksh} 20011208" \
  trade/ignored \
  ${0%/*} \
  ${0##*/}
```

Please note that “/ignored” part was supposed to be “/passwd”. As this is not secure, the passwd handling is moved from the user program to the driver program, and moved from command line to here document. However, we keep the syntax for backward compatibility.

As you can see, the parameter in 3rd and 4th position can be derived from “\$0”, myname. That is why these parameters are not needed in the new argument style.

Please also note that no path is hard coded in the new argument style.

SEE ALSO

process_feed.ksh help=y.

AUTHORS

Michael Wang, 2001–2002, <xw73@columbia.edu>.

Author Unknow, prior to 2001.