The LS-TaSC™ Tool

TOPOLOGY AND SHAPE COMPUTATIONS

SETTING UP A QUEUING SYSTEM

January 2021

Version 2021 R1

Corporate Address

Livermore Software Technology, an Ansys Company

P. O. Box 712

Livermore, California 94551-0712

Support Addresses

Livermore Software Technology, an Ansys Company

7374 Las Positas Road

Livermore, California 94551

Tel: 925-449-2500 ◆ Fax: 925-449-2507

Email: sales@lstc.com

Website: www.lstc.com

Livermore Software Technology, an Ansys Company

1740 West Big Beaver Road

Suite 100

Troy, Michigan 48084

Tel: 248-649-4728 ◆ Fax: 248-649-6328

Disclaimer

1-Mar-21

1. Setting UP a queuing system

1.1. Relationship with the LS-OPT queuing system

This queuing system is the same as used in LS-OPT. If your queue setup works for LS-OPT then it should work for LS-TaSC as well. This appendix mostly repeats the information for people not using LS-OPT.

In the LS-TaSC GUI the queuing is defined in the Scheduling tab of the Case definition, unlike in LS-OPT where the queuing system is defined in the run panel.

Also, you do not need to reinstall the *wrapper* program if it is already installed for LS-OPT.

The LS-TaSC and the LS-OPT queuing systems are the same. You only need to do the install process for one product. If it works for one, it should work for the other.

1.2. Experience may be required

Experience with the queuing system and help from the system administer may be required. The queuing systems are not provided by LSTC. Getting the queue system to work may therefore require work and insight from the customer.

1.3. Introduction

The LS-TaSC Queuing Interface interfaces with load sharing facilities (e.g. LSF1 or LoadLeveler2) to enable running simulation jobs across a network. LS-TaSC will automatically copy the simulation input files to each remote node, extract the results on the remote directory and transfer the extracted results to the local directory. The interface allows the progress of each simulation run to be monitored via the GUI. The

[1] Registered Trademark of Platform Computing Inc.
[2] Registered Trademark of International Business Machines Corporation

README.queue file should be consulted for the most up to date information about the queuing interface.

1.4. Installation

To run LS-TaSC with a queuing (load-sharing) facility the following binary files are provided in the `LSOPT_EXE` directory which un-tars (or unzips) from the distribution during installation of LS-OPT:

```
LSTASC_EXE/wrapper
LSTASC_EXE/runqueuer
```

The runqueuer executes the command line for the purpose of queuing and must remain in the LS-TaSC environment (the same directory as the lstasc executable).

The following instructions should then be followed:

Installation for all remote machines running LS-DYNA

1. Create a directory on the remote machine for keeping all the executables including `lsdyna`. Copy the appropriate executable `wrapper` program to the new directory. e.g. if you are running lsdyna on a Linux machine, place the `wrapper` appropriate for the architecture and operating system on this machine. You do not need to reinstall the *wrapper* program if it is already installed for LS-OPT.

Installation on the local machine

1. Select the queuer option in the GUI for the Case definition.

   To pass all the jobs to the queuing system at once, select zero concurrent jobs in the GUI or command file.

   In this example, the arguments to the rundyna.hp script are optional and can be hard-coded in the script.

2. Change the script you use to run the solver via the queuing facility by prepending "wrapper" to the solver execution command. Use full path names for both the wrapper and executable or make sure the path on the remote machine includes the directory where the executables are kept.

   The argument for the input deck specified in the script must always be the LS-OPT reserved name for the chosen solver, e.g. for LS-DYNA use `DynaOpt.inp`.

1.5. Example

Example: The LS-TaSC command relating to the queue is "/nec00a/mike/project/submit_pbs". The "submit_pbs" file is:

```
#!/bin/csh -f
#
# Run jobs on a remote processor, remote disk
set newdir=`pwd | sed -n 's/.*\/\(.*\)\/\(.*\)/\1\/\2/p'`
# Run jobs on a remote processor, local disk (no transmission)
# set newdir=`pwd`
echo $newdir
cat > dynscr << EOF
#!/bin/csh -f
#
#PBS -l nodes=1:ncpus=1
#
setenv LSOPT /nec00a/mike/codes/LSOPT_EXE
setenv LSOPT_HOST $LSOPT_HOST
setenv LSOPT_PORT $LSOPT_PORT
# Run jobs on a remote processor, remote disk
mkdir -p lsopt/$newdir
cd lsopt/$newdir
# The input file name is required for LS-TaSC
/nec00a/mike/codes/wrapper /nec00a/mike/codes/ls980.single
i=DynaOpt.inp
EOF
qsub dynscr
```

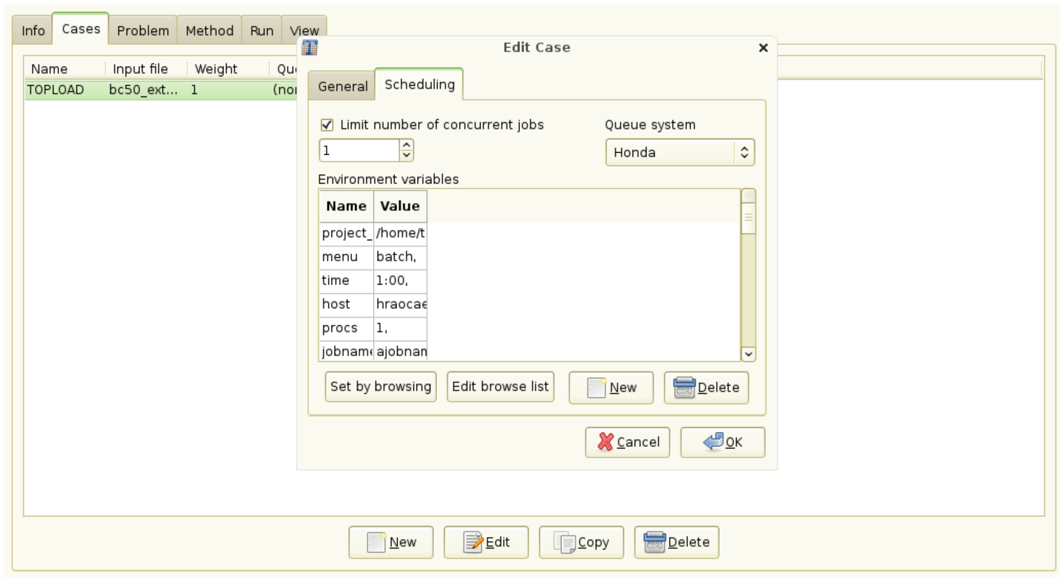
It is also possible to specify the queuer command directly on the command line. Environment variables can be specified on the solver command line (e.g. for the PBS queuing system) as well as LS-TaSC input data.

Example:

This example shows how the required environment variables LSOPT_PORT and LSOPT_HOST set by the runqueuer program are specified on the solver command line whereas the two user variables LSDYNA971_MPP and LSOPT_WRAPPER are defined and stored as special input entities (see Section 1.13). These can also be set on the command line using the Linux "`setenv`" command. `qsub` is a PBS queue submit command and the –v directive defined the names of environment variables to be exported to the job. The `qsub` manual pages should also be consulted for more details. Say we submit to qsub using the command "qsub -v LSOPT_PORT, LSOPT_HOST ../../dynscr2". The `dynscr2` file in this case is:

```
# This is the dynscr2 file
#==========================
#!/bin/csh -f
#
#$ -cwd -pe mpi 2
#
setenv NP 2
setenv ROUNDROBIN 0
#
# Define LSDYNA971_MPP environment variables in lsopt input
# or shell command ("setenv").
# $1 represents i=DynaOpt.inp and is automatically
# tagged on as the last argument of the lsopt "solver command".
#
setenv EXE "$LSDYNA971_MPP $1"
#
rm -f mpd.hostfile mpp.appfile
filter_hostfile < $PE_HOSTFILE > mpd.hostfile
#
# This python script builds an HPMPI specific "appfile" telling it
# exactly what to run on each node.
#
gen_appfile.hpmpi mpd.hostfile $SGE_O_WORKDIR $NP $ROUNDROBIN $EXE >
mpp.appfile
#
# This actually executes the job
#
$LSOPT_WRAPPER /opt/hpmpi/bin/mpirun -f mpp.appfile
#
```

The solver command data and environment variable input are displayed below:



Name	Input file	Weight	Qu
TOPLOAD	bc50_ext...	1	(no

[✓] Limit number of concurrent jobs: 1

Queue system: Honda

Environment variables

Name	Value
project_	/home/t
menu	batch,
time	1:00,
host	hraocae
procs	1,
jobnam	ajobnan

1.6. Mechanics of the queuing process

Understanding the mechanics of the queuing process should help to debug the installation:

1. LS-TaSC automatically prepends `runqueuer` to the solver command and executes runqueuer which runs the `submit_pbs` script.
2. The `runqueuer` sets the variables `LSOPT_HOST` and `LSOPT_PORT` locally.
3. In the first example, the `submit_pbs` script spawns the `dynscr` script.

4. In Example 1 the queuing system then submits `dynscr` (see `qsub` command at the end of the submit_pbs script above) on the remote node which now has fixed values substituted for `LSOPT_HOST` and `LSOPT_PORT`.

5. In Example 2 the LS-TaSC schedules the `qsub` command directly with `LSOPT_HOST` and `LSOPT_PORT` as arguments and `i=DynaOpt.inp` appended at the end of the command. It therefore serves as an argument to `dynscr2.`

6. The wrapper executes on the same machine as LS-DYNA, opens a socket and connects back to the local host using the host/port information. The standard output is then relayed to the local machine. This output is also written to the `logxxxx` file (where `xxxx` is the process number) on the local host. To look at the log of any particular run, the user can select a button on the Run page under the *View Log* heading. The progress dialog is shown below, followed by the popup log.

   An example of an error message resulting from a mistype of “wrapper” in the submit script is given in another example log file as follows:

   ```
   STARTING command /home/jim/bin/runqueuer
   PORT=56984
   JOB=LoadLeveler
   llsubmit: The job "1/1.1" has been submitted.
   /home/jim/LSOPT_EXE/Xrapper: Command not found.
   Finished with directory
   ```

```
/home/jim/LSOPT/4.1/optQA/QUEUE/EX4a_remote/remote/1/1.1
```

7. The wrapper will also extract the data immediately upon completion on the remote node. A log of the database extraction is provided in the `logxxxx` file.

1.7. Environment variables

These variables are set on the local side by the `runqueuer` program and their values must be carried to the remote side by the queuing software. The examples above illustrate two methods by which this can be accomplished.

`LSOPT_HOST`: the machine where LS-TaSC (and therefore the `runqueuer`) is running.

`LSOPT_PORT`: TCP/IP port `runqueuer` listens on for remote connections

1.8. Troubleshooting

1. Diagnostics for a failed run usually appear in the logxxxx file in the run directory. If there is almost no information in this file, the wrapper path may be wrong or the submission script may have the wrong path or permission. For any job, this file can be viewed from the progress dialog on the **Run** page.

   Please attach the log file (lsopt_output) when emailing support@lstc.com.

2. Make sure that the permissions are set for the executables and submission script.
3. Check all paths to executables e.g. "wrapper", etc. No diagnostic can detect this problem.
4. Make sure that the result database is produced in the same directory as where the wrapper is started, otherwise the data cannot be extracted. (E.g. the front end program such as mpirun may have a specification to change the working directory (`-wd dir`)).
5. *Running on a remote disk.* Make sure that the file "`HostDirectory`" is not copied by a user script to the remote disk if the simulation run is done on a remote disk. The "`HostDirectory`" file is a marker file which is present only on the local disk. Its purpose is to inform the wrapper that it is running on the local disk and, if found on a remote disk, will prevent the wrapper from automatically transferring extracted results back to the local disk. In general the user is not required to do any file copying since input files (including LS-DYNA include files) are copied to the remote disk automatically. The response.* and history.* files are recovered from the remote disk automatically.
6. *Termination of user-defined programs:* LS-DYNA always displays a '`N o r m a l`' at the end of its output. When running a user-defined program which does not have this command displayed for a normal termination, the program has to be executed from a script followed by a command to write '`N o r m a l`' to standard output. The example file *runscript* shown below first runs the user-defined solver and then signals a normal termination.

```
mpiexec –n 2 /home/john/bin/myprogram –i UserOpt.inp
# print normal termination signal to screen
echo 'N o r m a l'
```

   which is submitted by the wrapper command in submit_pbs as:

```
 /home/john/bin/wrapper /home/john/bin/runscript
```

Note: Adding "`echo N o r m a l`" at the end of the wrapper command (after a semicolon) does not work which is why it should be part of the script run by the wrapper.

1.9. User-defined queuing systems

To ensure that the LS-TaSC job scheduler can terminate queued jobs, two requirements must be satisfied:

1. The queuer must echo a string

   `Job "`*`Stringa Stringb Stringc …`*`" has been submitted`

   or

```
Job Stringa has been submitted
```

e.g.

```
Job "Opteron Aqs4832" has been submitted
Job aqs4832 has been submitted
```

The string will be parsed as separate arguments in the former example or as a single argument in the latter example. The string length is limited to 1024 characters. The syntax of the phrases "`Job `" and "` has been submitted`" must be exactly as specified. If more than one argument is specified without the double quotes, the string will not be recognized and the termination feature will fail.

2. A termination script (or program) `LsoptJobDel` must be placed either in the main working directory (first default location) or in the directory containing the LS-TaSC binaries (second default). This script will be run with the arguments *stringA*, *stringB*, etc. and must contain the command for terminating the queue. An example of a Unix C shell termination script that uses two arguments is:

```
#!/bin/csh -f

aadmin –c $1 –j $2 stop
```

1.10. Blackbox queueing system

The Blackbox queueing system is another flavor of the User-defined queueing system. It can be used when the computers running the jobs are separated from the computer running LS-TaSC by means of a firewall. The key differences between User-defined and Blackbox are:

1. It is the responsibility of the queueing system or the user provided scripts to transfer input and output files for the solver between the queueing system and the workstation running LS-TaSC. LS-TaSC will not attempt to open any communications channel between the compute node and the LS-TaSC workstation.
2. Extraction of responses and histories takes place on the local workstation instead of on the computer running the job.
3. LS-TaSC will not run local placeholder processes (i.e. extractor/runqueuer) for every submitted job. This makes Blackbox use less system resources, especially when many jobs are run in each iteration.

When using the Blackbox queueing system, a `LsoptJobDel` script is required, just as in the User-defined case. Furthermore, another script named `LsoptJobCheck` must also be provided. This script takes one parameter, the job ID, as returned by the submission script. The script should return the status of the given job as a string to standard output.

The Blackbox queuer option requires the user to specify a command that will queue the job. The Blackbox option can also be specified in the Scheduling panel when defining a

Case. The command to queue the job must return a *job identifier* that has one of the following two forms:

```
Job "Any Quoted String" has been submitted
Job AnyUnquotedStringWithoutSpaces has been submitted
```

The Word "`Job`" must be the first non-white space on the line, and must appear exactly as shown. Any amount of white space may appear between "`Job`" and the job identifier, as well as after the job identifier and before "`has been submitted`".

The Blackbox queuer requires the presence of two executable scripts `LsoptJobCheck` and `LsoptJobDel`. These scripts must be located in either in the current LS-TaSC project directory or in the directory where the running LS-TaSC program is located. (For Windows, the scripts must have an added extension `.exe`, `.vbs`, `.cmd` or `.bat`). If the Blackbox queuer option is invoked for some solver, then LS-TaSC checks for the existence of executable scripts in one of these locations, and refuses to run if the `LsoptJobCheck` and/or `LsoptJobDel` scripts cannot be found or are not executable. The project directory is searched first.

LsoptJobCheck script

The user-supplied `LsoptJobCheck` script is run each time LS-TaSC tries to update the current status of a job. The `LsoptJobCheck` script is run with a single commandline argument:

```
LsoptJobCheck job_identifier
```

The working directory of the `LsoptJobCheck` script is set to the job directory associated with job_identifier.

The script is expected to print a status statement that LS-TaSC can use to update its status information. The only valid status statements are:

String	Description
`WAITING`	The job has been submitted and is waiting to start
`RUNNING`	The job is running.
`RUNNING` *`N/M`*	After `RUNNING`, the script may also report the progress as a fraction. `RUNNING 75/100` means that the job has ¼ to go. The progress information will be relayed to the user, but not used in any other way by LS-TaSC.
`FAILED`	The job failed. This is only to be used when the underlying queueing system reports some kind of problem. Hence, a solver that has terminated in error does not have to be deteceted by the `LsoptJobCheck` script.

FINISHED	The job has completed and any output files needed for extraction has been copied back to the run directory.

Any amount of white space may appear at the beginning of a status statement, and anything may appear after these statements. The optional *N/M* argument for `RUNNING` is interpreted as an estimate of the progress; in this case *N* and *M* are integers and *N/M* is the fractional progress. *N* must be not be larger than *M*.

If `LsoptJobCheck` terminates without printing a valid status statement, then it is assumed that `LsoptJobCheck` does not function properly, and LS-TaSC terminates the job using the `LsoptJobDel` script. All output from the `LsoptJobCheck` script is logged to the job log file (`logxxxx`) in the run directory for debugging purposes.

Note: The `LsoptJobCheck` script may print more than one status statement, but only the first one will be used to update the status.

LsoptJobDel script

The user-supplied `LsoptJobDel` script is run whenever the user chooses to terminate a job, or whenever LS-TaSC determines that a job should be killed (for example, if `LsoptJobCheck` fails). The `LsoptJobDel` script is run with a single commandline argument:

```
LsoptJobDel job_identifier .
```

The working directory of the `LsoptJobDel` script is set to the job directory associated with job_identifier.

1.11. Honda queuing system

The Honda queuing system interface is based on the Blackbox queuing system, but is dedicated to the particular needs of this system.

Mechanics of the Honda queuing process

The queuing system generates a status file for which an environment variable has been defined in LS-TaSC as:

```
$HONDA_STATUSFILE
```

The status file is the output of the PBS queue check command. During the initialization phase, LS-TaSC checks whether this variable setting points to a valid file. If it does not, LS-TaSC terminates before starting the scheduler, and prints a standard LSOPT-style error message.

The line which marks the fields in the status file is used to determine how to parse the file; this line has the form "----- ----------- - ----- ---- ....". Fields are extracted based on this line which consists solely of space and dash characters. The following fields are used:

4	name
6	status: 'R' for running or 'Q' for queued
10	total wall clock time allowed
11	total wall clock time consumed.

Fields 10 and 11 are used to set the progress indicator. If the indicator ever reaches 100%, then it will terminate due to total wall clock time restrictions.

If a job cannot be found in the status file, then it is assumed to be dead. The job status entry is not looked for until a minimum of 3 seconds after the job has been started. A status file is searched for a particular job status entry only if the status file has a modification time that is later than the start time of the job.

Since there is no way to determine the exit status of a job by looking only at this status file, the determination of the final exit status depends on whether or not the job is an LS-DYNA job. If the job is an LS-DYNA job, then the messag file is parsed for the status statements "N o r m a l" and "E r r o r" termination. If no messag file is found *10 seconds* after the job is no longer listed in the status file, then we assume an error termination.

If the job is a non-LS-DYNA job, then LsoptJobCheck (see Section 1.10) is executed just once after the job no longer appears in the status file. LsoptJobCheck should print either (a) FINISHED or (b) ERROR in order to communicate the final exit status. If LsoptJobCheck cannot be found or cannot be executed, then ERROR is assumed. The job log file will contain a message indicating any problem that may exist which prevents LsoptJobCheck from being run.

The HONDA queued jobs do not use LsoptJobDel as defined in the Blackbox queuing selection. Jobs are deleted using the standard PBSPro qdel command.

Various statements concerning how status information is gathered are logged to the job log files. These are:

1. Job status for LSDYNA jobs found in 'messag' file:

```
[HONDA] Termination status found in 'messag' file
[HONDA] exact termination statement
```

2. The job status line for the current job found in $HONDA_STATUSFILE is saved:

```
[HONDA] status line
```

3. The job is assumed finished if there is no status line found:

```
    [HONDA] Job 23551 not found in STATUS file - assuming job is
finished.
```

4. Indication that LsoptJobCheck is run at the end of a non-LS-DYNA job:

```
    [HONDA] Non LS-DYNA job. Running LsoptJobCheck to determine
exit status.
```

5. Status returned from LsoptJobCheck.

```
     [HONDA] Job finished - LsoptJobCheck reports normal
termination
     [HONDA] Job finished - LsoptJobCheck reports error
termination
```

Any errors while gathering status information are logged to the job log files such as log12345.

6. Missing messag file after LSDYNA terminates:

   ```
   [HONDA] Failed to find 'messag' file while FINISHING.
   [HONDA] Assuming ERROR termination for LSDYNA job.
   ```

7. Found no termination status statement in messag file

   ```
   [HONDA] Found no termination status in 'messag' file
   [HONDA] Assuming ERROR termination for LSDYNA job.
   ```

8. HONDA_STATUSFILE variable not set

   ```
   [HONDA] *** Error $HONDA_STATUSFILE not set.
   ```

9. Could not open $HONDA_STATUSFILE

```
     [HONDA] *** Error Failed to open
$HONDA_STATUSFILE=pbsq_status
```

10. LsoptJobCheck script not found for non-LSDYNA job

    ```
    [HONDA] *** Error LsoptJobCheck cannot be found.
    [HONDA]     Assuming error termination for non-LSDYNA job.
    ```

11. LsoptJobCheck script did not print either (a) FINISHED or (b) FAILED.

```
     [HONDA] *** Error LsoptJobCheck did not return a valid
status.
     [HONDA]          Assuming error termination for non-LSDYNA
job.
```

If $HONDA_STATUSFILE is not updated in a timely fashion, then the scheduler can hang forever, never moving forward. A message is passed to lsopt through the communication socket if this happens:

```
 *** Warning HONDA_STATUSFILE out of date by more than 5 minutes
 *** Job progress monitoring suspended until next update
```

Even though the status file is checked before starting the scheduler, it is still possible for file errors to occur. These are also sent directly to LS-TaSC.

```
 *** Error $HONDA_STATUSFILE not set
 *** Error Failed to open $HONDA_STATUSFILE=pbsq_status
```

1.12. Microsoft Windows Compute Cluster server

LS-TaSC supports submission of jobs to the Microsoft Compute Cluster Pack Scheduler. Two scripts called `submit.cmd` and `submit.vbs,` that work together, are available to interface LS-TaSC with CCP. The script can be downloaded from `ftp://ftp.lstc.com/ls-opt/QUEUING/MSCCS`. Before using the scripts the variables in the beginning of the file `submit.cmd` needs to be changed to fit your local environment. Most users do not need to change the `submit.vbs` file.

The case panel of LS-TaSC when using the CCP scripts requires that the executable is `submit.cmd` (or the local name) and the queuing system is set to Microsoft CCP/CCS.

1.13. Passing environment variables

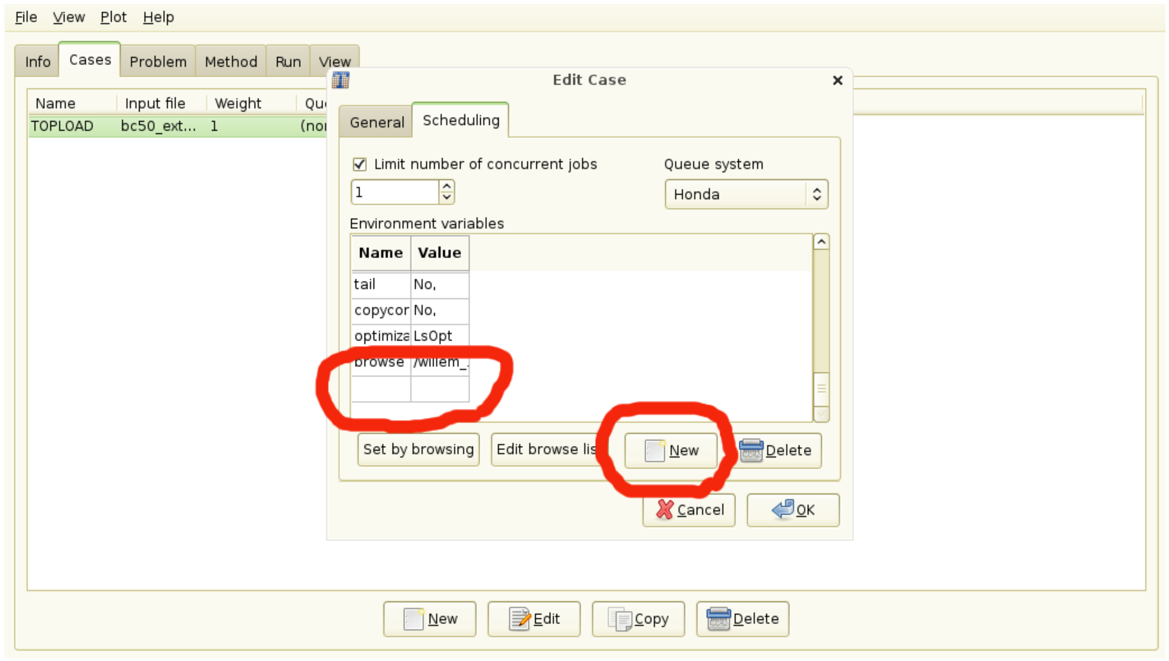
LS-TaSC provides a way to define environment variables that will be set before executing a solver command. The desired environment variable settings can be specified directly in the com file with solver commands:

They can be specified within the Scheduling tab when defining a Case.

1.13.1. Adding a new environment variable definition

Select the New button. After selecting this option, an empty, editable environment variable definition will appear.

We do not allow the names of variables to contain anything other than upper- or lower-case letters, numbers, and underscore (_) characters. Variable values are not so limited.



Name	Input file	Weight	Qu
TOPLOAD	bc50_ext...	1	(nor

☑ Limit number of concurrent jobs: 1

Queue system: Honda

Environment variables

Name	Value
tail	No,
copycor	No,
optimize	LsOpt
browse	/willem_

Figure 1-1: Adding a new environment variable

1.13.2. Editing an existing environment variable definition

To edit an environment variable, double-click on the environment variable in the Environment variables list. The display mode of the variables will change to make it editable.

1.13.3. Set by browsing

Select the **Select by Browsing** button. In order for this option to work, user-supplied executables must be present in the directory

$HOME/LSOPT_SCRIPTS

The directory LSOPT_SCRIPTS must exist as a subdirectory of the user's home directory, and it must contain executables. If the directory LSOPT_SCRIPTS does not exist, or if there are no executables in this directory, an error box will appear. Setting the LSOPT_SCRIPT Unix/Linux/Windows system environment variable may specify an alternative script directory.

After selecting the **Set by browsing** option, a dialog of buttons will appear, one for each executable in this directory. For example, suppose this is the directory listing for `$HOME/LSOPT_SCRIPTS`:

```
-rwxr-xr-x 1 joe staff 13597 2009-12-01 18:09 lsdyna_submit.autounion*
-rw-r--r-- 1 joe staff 13597 2009-12-01 17:46 stdin.save
-rwxr-xr-x 1 joe staff     9 2009-08-10 14:23 test*
-rwxr-xr-x 1 nielen staff  9 2009-08-10 14:26 testb*
```

Then, when you select the Set by browsing option, the following dialog appears:

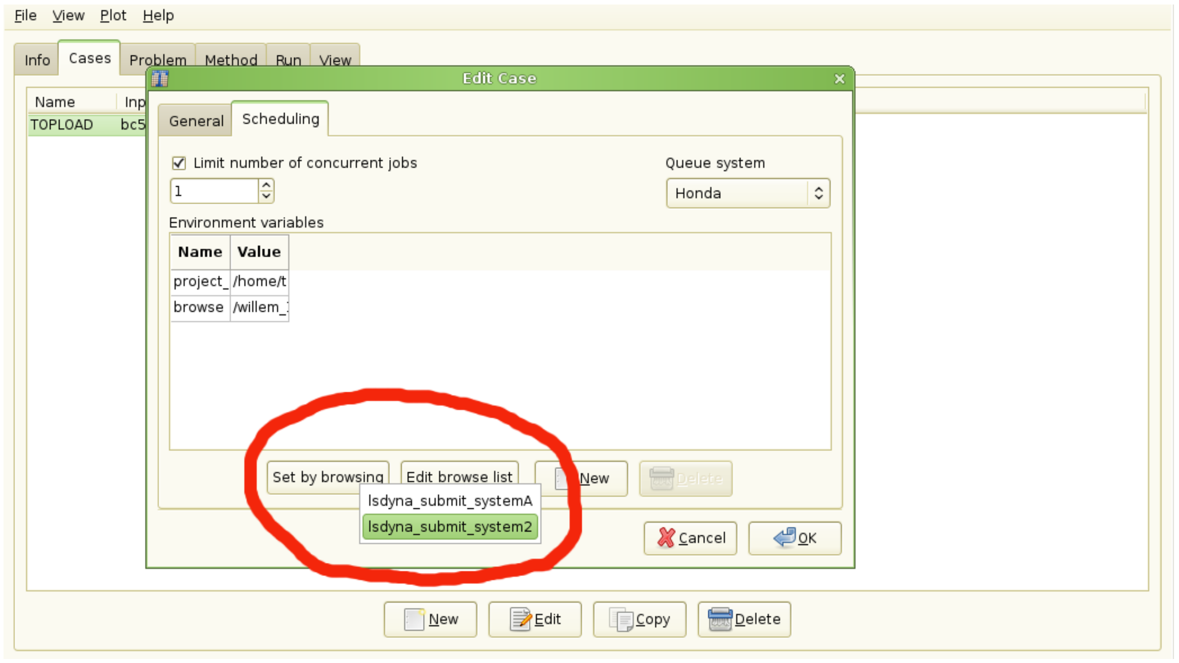


Figure 1-2: Set new environment variables by browsing

A valid browse command must print environment variable definitions to standard output in the form name='value'; the single quotes are optional if value does not contain spaces. A valid sample output is shown below (the line is wrapped because of its length).

exe=/home/trent/LSTC/PERL/lsdyna-caec01_pbs_sub.pl menu=batch time=1:00 host=abcdefgh07 procs=1 jobname='My Job' project=isd email=No delay=No preemptable=No version='LS-DYNA 970 MPP SP 6763' ioloc=/home/trent inpfile=DynaOpt.inp mem1=auto mem2=auto pfile=Generic dumpbdb=No dynamore=No clean=No tail=No copycont=No optimization=LsOpt

All of the name='value' strings are directly imported into the Env Vars tab in bulk. In addition to these **Browse List** variables, a special browse variable is created that should not be edited. This variable records the program name used to create the Browse List.

NOTE: All variables must be printed on one line, which must be the last line of output from the program. Lines before the last line are ignored.

WARNING: The user-supplied browse program should never define the browse variable in its output. The name browse should be treated as a reserved name.

A simple Linux browse command could be a shell script:

```
#!/bin/bash
echo This line is ignored. Only the last line survives
echo A=B C=D
```

Running the browse command shown above will import two variables, A and C, into the browse list.

NOTE: Strings in the Env Vars List appearing above the browse= line are all part of the Browse List. Strings in the Env Vars tab that appear below browse= are never part of the Browse List. User-defined environment variables will always follow after the browse variable definition (e.g., last=first in the figure above was not defined by the browse command.)

1.13.4. Edit browse list

Select the Edit Browse list button. Choosing this option does nothing unless a Browse List has been previously created. If a valid Browse List is present in the Env Vars tab, then selecting this option will run the original program that created the Browse List, together with all of the current Browse List options passed as command line arguments, one per existing environment variable. Each command-line argument has the form name=value. However 'value' is not single-quoted because each name=value argument is a separate command-line argument. The customer-supplied browse command should offer the user an opportunity to edit the existing variables, and the browse command should return the newly edited list on one line, in the same format as described above. This would normally be done through some sort of graphical user interface. The returned list will be used to replace all of the previous Browse List.

The next example script returns an initial Browse List consisting of two variables, A and C. Invoking the editing feature appends a new variable (tN=N) to the list.

```
#!/bin/bash
echo This line will be ignored. Only the last line survives.
if [ "$1" == "" ]; then
  echo A=B C=D;
else
  echo $* "t"$$"="$$;
```

fi

When this script is invoked using the "Create by Browse" feature, there are no command-line arguments, and the script prints "A=B C=D" to standard output. However, when the script is invoked using the edit feature for the first time, two command-line arguments "A=B" and "C=D" are passed to the script. This time the return line consists of the original command-line arguments (printed using $*) and tN=N, where N is the PID of the shell process. If the editing feature is invoked a second time, then three command-line arguments are passed to the script ("A=B", "C=D", and "tN=N"). Another new variable tN is appended, where N is the newest PID of the script process. This sample script has little practical value, except to illustrate how existing variable settings are passed by command-line to the previous browse command, and to illustrate how one can use the editing feature to modify or add new variables.

Note: The browse command can ABORT the replacement operation by printing a blank line to the standard output and immediately terminating. Otherwise the current Browse List may be deleted. If the browse command abnormally terminates, then an error box will appear with a title bar indicating that the command failed.

1.13.5. How the browse list is used by LS-TaSC

The **Browse List** (indeed, the complete **Env Vars List**) is used to set environment variables before running the solver command specified by LS-TaSC. However, if the first variable returned by the browse command is **exe**, then a pre-processing command is run before running the actual solver command. The pre-processing command is the value of the **exe** variable. The pre-processing command has a command line

```
$exe var1=$var1, var2=$var2, ... varN=$varN
```

That is, the command executed is the value of the **exe** variable; additional command line arguments consist of all **Browse List** strings with a comma delimiter appended to each intermediate one. (The final argument is not followed by a comma.)

Note: Such a pre-processing command is always run from within the current **LS-TaSC Job Directory**. Therefore, any file that the pre-processing command references must be specified by a fully-qualified path or must be interpreted relative to the current **LS-TaSC Job Directory**. So, the **LS-TaSC Case Directory** will be **".."** and the **LS-TaSC Project Directory** will be **"../..”.**

1.14. Enabling LSTCVM job proxy support

The LSTCVM proxy server is distributed separately from LS-TaSC. The installation is usually handled by a systems administrator, who is advised to contact his local LSTC support site beforehand to see whether support is available before attempting installation.