Platform LSF™ Desktop Support™ User's Guide

Version 7.0 Update 2
Release date: November 2007
Last modified: December 4 2007
Support: support@platform.com
Comments to: doc@platform.com

We'd like to hear from you You can help us make this document better by telling us what you think of the content, organization, and usefulness of the information. If you find an error, or just want to make a suggestion for improving this document, please address your comments to doc@platform.com.

Your comments should pertain only to Platform documentation. For product support, contact support@platform.com.

Contents

CHAPTER

1

Using LSF Desktop Support to Run Jobs

You run jobs in LSF desktop support in much the same way as you run jobs in standard LSF, using the `bsub` command. However, you must make sure that the queue or host to which you submit your job is configured for LSF desktop support. In addition, unless your cluster is configured as an LSF desktop support-only cluster and the LSF desktop support queue is the default queue, you must specify this queue when you submit a job.

In this chapter

Submitting a Job

You can submit one or more jobs to LSF desktop support using the `bsub` command. You can submit a job that runs the following types of PC commands: `.cmd,` `.bat`, and `.exe`.

You must make sure that the queue or host to which you submit your job is configured for LSF desktop support. In addition, unless your cluster is configured as an LSF desktop support-only cluster and the LSF desktop support queue is the default queue, you must specify this queue when you submit a job.

Submit a job using the bsub command:

On the command line, type a `bsub` command to submit the job to LSF desktop support. Make sure that you include the applications and files required to run the job, and specify a queue name if you are not using the default LSF desktop support queue AC_QUEUE. For example:

```
bsub -q LongQueue \
-f "BinDir/JobScript.cmd > JobScript.cmd" \
-f "InputDir/InputFile > InputFile" \
-f "OutputDir/OutputFile < OutputFile" \
JobScript.cmd
```

By default, all files are cached, but the LSF administrator can globally disable or selectively enable file caching, as described in the *Platform LSF Desktop Support Administrator's Guide*. If selective file caching is enabled by the administrator, you can append + to a file name when submitting a job. For example:

`bsub -f "local_file+ > remote_file"` caches the file on the desktop client.

For the complete syntax of the `bsub` command for LSF desktop support, see "bsub" on page 17.

Submitting a Job with Many File Transfers

LSF desktop support allows a maximum of 32 file transfer requests with the `-f` option in the LSF desktop support version of the `bsub` command. To specify more than 32 file transfers, use the zip and unzip commands to reduce the number of file transfer requests.

In the following example, `myjob.exe` requires a total of 66 file transfers: 33 to copy files to the desktop client, and 33 to copy the results from the desktop client.

To transfer many files:

1. Zip the data files together into one file. For example:

   ```
   zip data.zip data1 data2 data3 ...data33
   ```

2. Create a job wrapper that unzips the data files, runs the executable and zips the results. For example, the wrapper `myjob.bat` might look like this:

   ```
   unzip data.zip
   myjob.exe
   zip result.zip result1 result2 result3 ... result33
   ```

3. Submit the job, transferring the data files, the wrapper and the executable to the desktop client, and transferring the zipped results file back from the desktop client. For example:

   ```
   bsub -f "data.zip > data.zip" -f "myjob.bat > myjob.bat" -f
   "myjob.exe > myjob.exe" -f "result.zip < result.zip"
   myjob.bat
   ```

4. When the job is completed, unzip the result file:

   ```
   unzip result.zip
   ```

If you do not have zip and unzip on your system, you can get them from the Internet, and install them on each desktop client as required.

Using Job Arrays

A job array is a sequence of jobs that share the same executable but have different input files. Creating a job array allows you to submit, control and monitor these jobs as a single unit. Using standard LSF commands, you can also control and monitor individual jobs that were submitted from an array.

After the jobs are submitted, LSF independently schedules and dispatches the individual jobs. Each job submitted from a job array shares the same job ID as the job array, and is uniquely referenced using an array index.

You create a job array at job submission time using the `bsub` command.

If a job needs to know its own ID, you can retrieve the ID of any job in an array using the LSB_JOBINDEX environment variable. See "To retrieve the ID of a job in an array:" on page 9 for instructions.

To submit a job array using the bsub command:

1. Specify the `bsub` command and include the `-J` option:

   bsub -J "*arrayName*[*indexList*, ...]"

   where *arrayName* is a string used to identify the job array. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (_).

   where *indexList* can be a range of unique positive integers, such as

   [1-5]

   or *indexList* can be in the format:

   [*start-end*[:*step*]]

   where *start* is used with *end* to specify the start of a range of indices, and *end* specifies the end of the range. *step* specifies the value to increment the indices in the range. For example:

   [1-10:2]

   specifies a range of 1-10 with a step value 2, creating indices 1,3,5,7 and 9.

2. Make sure that your input files are all stored in the current working directory (or specify the full path name for the directory where they are stored), and are all named consistently to correspond with the indices of the array. For example, if the array indices are from 1 to 1000:

   ```
   input.1, input.2, input.3, ..., input.1000
   ```

3. If there is more than one LSF desktop support queue, and you are not using the default queue, specify the queue name.

Example: Submitting a job array

The following example submits a job array with 1000 entries. The special character `%I` is replaced by the index of the job in the array.

```
bsub -J "array[1-1000]" \
-f "input.%I > input" \
-f "output.%I < output" \
-f "job.cmd > job.cmd" \
job.cmd
```

The above example submits 1000 jobs that correspond to 1000 input files named `input.1` to `input.1000`. The `-f` option is used to copy the input files to the desktop client and the resulting output files to the desktop server. It is also used to copy the executable, which is `job.cmd`. The job array is submitted to the default queue AC_QUEUE.

To retrieve the ID of a job in an array:

1. Write a Windows script to access the job environment when the job is run. For example, create a file called `job_index.bat`, as follows:
   cmd /c set LSB_JOBID>>test
   cmd /c set LSB_JOBINDEX>>test
   cmd /c set LSB_RUNID>>test
2. Submit the job array to the LSF desktop support queue. For example:
   ```
   bsub -q AC_QUEUE -f “test > test”
   -f “job_index.bat > job_index.bat”
   -f ”test_back%I < test” -J ”a[1-10]” job_index.bat
   ```
   ```
   Job <775> is submitted to default queue <AC_QUEUE>.
   ```
3. After the job is finished, `test_back` should look something like the following:
   ```
   cat test_back1
   LSB_JOBID=105
   LSB_JOBINDEX=1
   LSB_RUNID=1
   ```
   where LSF_JOBID is the run ID of the job, LSB_JOBINDEX is the job array index, and the value of LSB_RUNID is the number of times the job has been redispatched.

Displaying Active Jobs

You can display the jobs submitted by a specific user or all users using the `bjobs` command. By default, the `bjobs` command displays jobs submitted by the user who invoked the command.

To display active jobs using bjobs:

1. On the command line, type the following: **`bjobs -u`** *`user_name`*. For example, `bjobs -u all` displays the jobs submitted by all users.

   Jobs are displayed in the following order:

   - Running jobs
   - Pending jobs, listed in the order in which they are scheduled
   - Jobs, those in high priority queues are listed before those in lower priority queues

2. Use the information provided to determine the status of a job. For more information about job status, see "Job Status Codes and What They Mean" on page 12.

For additional information about the `bjobs` command, refer to the *Platform LSF Reference*.

Displaying Job History Information

You can use the `bhist` command to track what happened to your job after submitting it. The `bhist` command displays a summary of the pending, suspended, and running times of jobs for the user who invoked the command. Use `bhist -u all` to display a summary for all users in the cluster. For more details about the `bhist` command, refer to the `bhist` command in the *Platform LSF Reference*.

To display detailed job history information:

1. On the command line, type the `bhist` command as follows:

   `bhist -l` *job_ID*

   The -l option displays the time information and a complete history of scheduling events for each job. For example:

   `bhist -l “jobarray[5]”`

   displays the job history information for the fifth element in `jobarray`.

For additional information about the `bhist` command, refer to the *Platform LSF Reference*.

Example

Job submission

```
bsub -q AC_QUEUE -f "myjob2.exe+ > myjob.exe-" -f "myjoboutput.log+ <"
      myjob.exe
Job <751> is submitted to queue <AC_QUEUE>.
12:53pm Fri, Aug-20-2004 my_host:~/tmp
[417]- bsub -q AC_QUEUE -f "myjob2.exe- > myjob.exe-" -f "myjoboutput.log+
      <" myjob.exe
Job <752> is submitted to queue <AC_QUEUE>.
```

Job history

```
12:53pm Fri, Aug-20-2004 my_host:~/tmp
[418]- bhist -l 751 752
Job <751>, User <LSF_user>, Project <default>, Command <myjob.exe>
Fri Aug 20 12:53:47: Submitted from host <my_host.lsf.platform.com>, to Queue
      <AC_QUEUE>, CWD <$HOME/tmp>, Copy Files "myjob2.exe+ > myjob.
      exe-" "myjoboutput.log+ < myjoboutput.log+";
Fri Aug 20 12:53:57: Dispatched to <my_host.lsf.platform.com>;
Fri Aug 20 12:53:57: Starting (Pid 9327);
Fri Aug 20 12:54:16: Running with execution home
</home/LSF_user/milkyway/fyactest/top/ach_top/work/.jobs>, Execution CWD
</home/LSF_user/tmp>, Execution Pid <-1>;
Fri Aug 20 12:54:16: Waiting;
Fri Aug 20 12:58:39: External Message "Running on <LSF_user>" was posted from
"root"to message box 0;
Fri Aug 20 12:58:40: Running;
Fri Aug 20 13:03:44: Done successfully. The CPU time used is 0.0 seconds;
Fri Aug 20 13:04:00: Post job process done successfully;

Summary of time in seconds spent in various states by Fri Aug 20 13:04:00
PEND PSUSP RUN USUSP SSUSP UNKWN TOTAL
274 0 323 0 0 0 597
```

Job Status Codes and What They Mean

Some job status codes in LSF desktop support have a slightly different meaning than in standard LSF. Refer to the following, which gives specific LSF desktop support meanings for those codes.

Note: You cannot suspend a running LSF desktop support job.

Codes and LSF desktop support-specific descriptions

Code	Description
PEND	The job is still queued in the LSF Scheduler
WAIT	The job is still queued in the desktop server, waiting for an desktop client to run it.
RUN	The job is currently running on an desktop client.
UNKNOWN	The desktop client service has not reported for a long time (by default, for more than 600 seconds); for example, while the Tomcat application server is restarting.

All other job status codes retain their standard LSF meanings.

Killing a Job

You can stop a job from processing and remove it from the system using the `bkill` command.

To kill a job:

On the command line, type the `bkill` command as follows:

bkill *job_ID*

For example:

```
bkill 1234
```

kills job 1234.

To kill an entire job array:

On the command line, type the `bkill` command as follows:

bkill *jobarray_ID*

where *jobarray_ID* is the job ID of the array. For example:

```
bkill 12345
```

kills the entire array.

To kill an element in a job array:

On the command line, type the `bkill` command as follows:

bkill "*jobarray_ID*[*index*]"

where *jobarray_ID* is the job ID of the array, and *index* is the element number you want to kill. For example:

```
bkill "12345[5]"
```

kills the fifth element in the array.

CHAPTER

2

LSF Desktop Support Commands

LSF desktop support uses many standard LSF commands. However, some key commands are different from the standard LSF commands— they are included here to avoid confusion.

In this chapter

LSF Commands

The following LSF commands are supported within LSF desktop support:

- `badmin`
- `bhist`
- `bhosts`
- `bjobs`
- `bkill` (most options)

Note: The `bkill` command can be used to kill an LSF desktop support job. However, the options that pertain to signals are not supported in LSF desktop support.

- `bqueues`
- `bsub` (used to submit jobs)

Note: In earlier versions, the LSF desktop support-only `asub` command was also used to submit jobs directly to the LSF desktop support queue.

- `busers`
- `lsadmin`

bsub

submits a batch job to LSF desktop support

SYNOPSIS

bsub [*options*] *command* [*arguments*]

bsub [**-h** | **-V**]

OPTION LIST

-H
-K
-b *begin_time*
-e *err_file*
-eo *err_file*
-extsched "AC_RES_REQ=*resources***"**
-f "*local_file op* [*remote_file*]**"** ...
-J *job_name* | **-J "***job_name***[***index_list***]%***job_limit***"**
-o *out_file*
-oo *out_file*
-P *project_name*
-q "*queue_name ...***"**
-sp *priority*
-u *mail_user*
-w '*dependency_expression***'**
-W *run_limit*

DESCRIPTION

Submits a job for batch execution to a queue and assigns it a unique numerical job ID.

Unless your cluster is configured as an LSF desktop support-only cluster and the LSF desktop support queue is the default queue, you must specify the LSF desktop support queue when you submit a job using the `-q` option.

OPTIONS

-H

Holds the job in the PSUSP state when it is submitted. The job is not scheduled until you tell the system to resume the job.

-K

Submits a batch job and waits for the job to complete. You are not able to submit another job until the job is completed.

-b *begin_time*

Dispatches the job for execution on or after the specified date and time. The date and time are in the form of [[*month*:]*day*:]*hour*:*minute* where the number ranges are as follows: month 1-12, day 1-31, hour 0-23, minute 0-59.

At least two fields must be specified. These fields are assumed to be *hour*:*minute*. If three fields are given, they are assumed to be *day*:*hour*:*minute*, and four fields are assumed to be *month*:*day*:*hour*:*minute*.

-e *err_file*

Specify a file path. Appends the standard error output of the job to the specified file.

If the current working directory is not accessible on the execution host after the job starts, LSF writes the standard error output file to `/tmp/`.

-eo *err_file*

Specify a file path. Overwrites the standard error output of the job to the specified file.

If the current working directory is not accessible on the execution host after the job starts, LSF writes the standard error output file to `/tmp/`.

-extsched "AC_RES_REQ=*resources***"**

Specifies that the job must run on a desktop client that meets the specified resource requirements. You use the information that follows to create a logical expression that defines the resources required. Check with your LSF desktop support administrator for a list of any custom resource requirements you can specify in addition to the following list:

Resource Name	Value Type	Description	Example
cput	string	CPU type	PII or PIII
ncpus	numeric	How many CPUs there are	1
cpusp	numeric	Main frequency of CPU (MHZ)	500
mem	numeric	Free physical memory (MBytes)	256
maxmem	numeric	Max physical memory (Mbytes)	64
disk	numeric	Max free disk space LSF desktop support jobs can use (MBytes)	1000

The following CPU types are supported:

CPU Types	Description
Am486	AMD-Am486
ATHLON	AMD Athlon™
ATHLON64	AMD Athlon™ 64
Celeron	Intel Celeron™ processor, models 5 or 6
K5	AMD-K5™
K6	AMD-K6™
K6-2	AMD-K6-2™
OPTERON64	AMD Opteron™ 64
Pen	Intel Pentium® processor
PPro	Intel Pentium® Pro processor
PII	Intel Pentium® II processor, model 3 or 5
PIIX	Intel Pentium® II Xeon™ processor
PIII	Intel Pentium® III processor, Intel Pentium® III Coppermine processor

CPU Types	Description
PIIIX	Intel Pentium® III Xeon™ processor
PIV	Genuine Intel Pentium® 4 processor

Specify the resource specification in the following format:

```
"AC_RES_REQ=resource_name operator value[operator
resource_name operator value...]"
```

Note: The resource specification cannot exceed 512 bytes in length.

`resource_name`

The name of the resource as defined in `ac.restype.xml`.

`operator`

The operator that defines the relationship between the resource name and the value, or between nested expressions. The operators are listed below in the order of precedence (the order in which they are evaluated, from top to bottom). The operator can be one of the following:

Operator	Description
(	Beginning of expression. Expressions within parentheses are evaluated first.
)	End of expression.
>	Greater than. Use only with value type numeric.
<	Less than. Use only with value type numeric.
>=	Greater than or equal to. Use only with value type numeric.
<=	Less than or equal to. Use only with value type numeric.
==	Equal to. Use with any value type.
!=	Not equal to. Use with any value type.
&&	Logical AND. Both expressions must be true.
||	Logical OR. One of the expressions must be true.

`value`

The value of the resource to compare.

To specify multiple resources, combine specifications together using && or || operators. You can join specifications using parentheses. For example:

```
"AC_RES_REQ=(cput==PIV||mem>256)&&disk>500"
```

The above example specifies either a Genuine Intel Pentium 4 CPU or 256 MB RAM, and 500 MB disk space.

`-f "`*`local_file operator [remote_file]`*`" ...`

Required within LSF desktop support, because it does not use a shared file system—all files and executables must be copied to the remote host.

Copies a file between the local (submission) host and the remote (execution) host. Specify absolute or relative paths for the file on the local host, but specify a relative path for the remote file.

If the remote file is not specified, it defaults to the local file, which is required.

Use multiple `-f` options to specify multiple files. You can specify up to 32 file transfers in one `bsub` command. If you need to make more than 32 file transfer requests, you can use zip and unzip to transfer all data files or results files in a single transfer.

You can use the following operators:

- **>** Copies the local file to the remote file before the job starts.

  You can enable or disable reading from the cache on the desktop client for a specific file when submitting a job. If the administrator allows writing to the cache on desktop clients (when the `SEDDisableCache` parameter in `SEDConfig.xml` is set to `no`), you can also enable or disable writing to the cache for a specific file when submitting a job, as described below. For additional information on file caching settings, refer to the *Platform LSF Desktop Support Administrator's Guide*.

  - The **+** operator indicates:
    - LSF desktop support downloads the file from the cache, if it exists there. Otherwise, it downloads the file from the Web server.
    - If the administrator allows desktop clients to write to the cache, then LSF desktop support also writes the file to the desktop client cache.

    For example: `bsub -f "local_file+ > remote_file"` caches the file on the desktop client.

  - The **-** operator indicates:
    - LSF desktop support downloads the file from the Web server.
    - LSF desktop support does not write the file to the desktop client cache.

    For example: `bsub -f "local_file- > remote_file"` does not write the file to the desktop client cache.

  - If there is no additional operator, for example `bsub -f "local_file > remote_file"`, then file caching on the desktop client depends on the file caching setting for each desktop server (specified by the `LSB_MED_CACHEFILE_DEFAULT` parameter in `med.conf`). For additional information on file caching, refer to the *Platform LSF Desktop Support Administrator's Guide*.

  Note: If a **+** or **-** operator is the last character of a file name used with the **>** operator, it is removed from the file name. Therefore, to use a file name ending in **+** or **-** with the **>** operator, repeat the operator as the last character. For example, if the copied file is `my_data+`, specify it as `my_data++`. For example, the command would be: `bsub -f "myfile++ > myfile"`

- **<** Copies the remote file to the local file after the job completes.
- **<<** Appends the remote file to the local file after the job completes. The local file must exist.
- **><** Copies the local file to the remote file before the job starts. Then copies the remote file to the local file after the job completes. Overwrites the local file.
- **<>** Copies the local file to the remote file before the job starts. Then copies the remote file to the local file after the job completes. Overwrites the local file.

Note: If an operator specifies copying a file from the server to a client when caching on the desktop client is enabled, then if the file has been transferred to the desktop client before, it is copied from the cache instead of from the desktop server.

-J *job_name*
-J "*job_name***[***index_list***]%***job_slot_limit***"**

Assigns the specified name to the job, and, for job arrays, specifies the indices of the job array and optionally the maximum number of jobs that can run at any given time. If you do not specify a job name, the command name is used.

To specify a job array, enclose the index list in square brackets, as shown, and enclose the entire job array specification in quotation marks, as shown. The index list is a comma-separated list whose elements have the syntax *start*[-*end*[**:***step*]] where *start*, *end* and *step* are positive integers. If the step is omitted, a step of one is assumed. The job array index starts at one. By default, the maximum job array index is 1000.

All jobs in the array share the same job ID and parameters. Each element of the array is distinguished by its array index.

-o *out_file*

Specify a file path. Appends the standard output of the job to the specified file. Sends the output by mail if the file does not exist, or the system has trouble writing to it.

If only a file name is specified, LSF writes the output file to the current working directory. If the current working directory is not accessible on the execution host after the job starts, LSF writes the standard output file to `/tmp/`.

-oo *out_file*

Specify a file path. Overwrites the standard output of the job to the specified file if it exists, or sends the output to a new file if it does not exist. Sends the output by mail if the system has trouble writing to the file.

If only a file name is specified, LSF writes the output file to the current working directory. If the current working directory is not accessible on the execution host after the job starts, LSF writes the standard output file to `/tmp/`.

If you use `-oo` without `-e` or `-eo`, the standard error of the job is stored in the output file.

-P *project_name*

Assigns the job to the specified project. If you do not specify a project, the project defined in LSB.DEFAULTPROJECT is used.

-q "*queue_name ...***"**

Submits the job to one of the specified queues. Quotes are optional for a single queue. The specified queues must be defined for LSF desktop support.

If you do not specify a queue name, the default LSF desktop support queue AC_QUEUE is used.

`-sp` `priority`

Specifies job priority, which allow users to order their jobs in a queue. Valid values for priority are any integers between 1 and MAX_USER_PRIORITY. LSF and queue administrators can specify priorities beyond MAX_USER_PRIORITY.

`-u` `mail_user`

Sends mail to the specified email destination.

`-w` `'dependency_expression'`

LSF places your job only when the dependency expression evaluates to TRUE. If you specify a dependency on a job that LSF cannot find (such as a job that has not yet been submitted), your job submission fails.

The dependency expression is composed of one or more dependency conditions. To combine conditions, use the following logical operators:

&& (AND)

|| (OR)

! (NOT)

Use parentheses to indicate the order of operations, if necessary.

Enclose the dependency expression in single quotes ('), and use double quotes for quoted items within it, such as job names.

In dependency conditions, the variable *op* represents one of the following relational operators:

>, >=, <, <=, ==, or !=

Use the following conditions to form the dependency expression:

`done(``job_ID |"job_name" ...)`

The job state is DONE. This is the default condition.

`ended(``job_ID | "job_name")`

The job state is EXIT or DONE.

`exit(``job_ID | "job_name" [,[op] exit_code])`

The job state is EXIT, and the job's exit code satisfies the comparison test. If you specify an exit code with no operator, the test is for equality (== is assumed).

`job_ID |"job_name"`

`numdone(``job_ID, op number | *)`

For a job array, the number of jobs in the DONE state satisfies the test. Use * (with no operator) to specify all the jobs in the array.

`numended(``job_ID, op number | *)`

For a job array, the number of jobs in the DONE or EXIT states satisfies the test. Use * (with no operator) to specify all the jobs in the array.

numexit(*job_ID, op number* | *)

For a job array, the number of jobs in the EXIT state satisfies the test. Use * (with no operator) to specify all the jobs in the array.

numpend(*job_ID, op number* | *)

For a job array, the number of jobs in the PEND state satisfies the test. Use * (with no operator) to specify all the jobs in the array.

numrun(*job_ID, op number* | *)

For a job array, the number of jobs in the RUN state satisfies the test. Use * (with no operator) to specify all the jobs in the array.

numstart(*job_ID, op number* | *)

For a job array, the number of jobs in the RUN, USUSP, or SSUSP states satisfies the test. Use * (with no operator) to specify all the jobs in the array.

post_done(*job_ID* | "*job_name*")

The job state is POST_DONE (the post-processing of specified job has completed without errors).

post_err(*job_ID* | "*job_name*")

The job state is POST_ERR (the post-processing of the specified job has completed with errors).

started(*job_ID* | "*job_name*")

The job state is:

- RUN, DONE, or EXIT

- PEND or PSUSP, and the job has a pre-execution command (bsub -E) that is running.

-W *[hour:]minute*

Sets the run time limit of the batch job. If a job runs longer than the specified run limit, it is killed immediately.

The run limit is in the form of [*hour*:]*minute*. The minutes can be specified as a number greater than 59. For example, three and a half hours can either be specified as 3:30, or 210.

-h

Prints command usage to `stderr` and exits.

-V

Prints LSF release version to `stderr` and exits.

command [argument]

The job can be specified by a command line argument *command*. *command* is assumed to begin with the first word that is not part of a `bsub` option. All arguments that follow *command* are provided as the arguments to the *command*.

Example: Transfer single application file

The following example transfers a single Windows application TestCPU.exe to an desktop client to run:

```
bsub -f "TestCPU.exe > TestCPU.exe" TestCPU.exe
```

Example: Transfer multiple files

The following example transfers `TestCPU.bat`, which launches `TestCPU.exe`:

```
bsub -f "TestCPU.exe > TestCPU.exe" \
-f "TestCPU.bat > TestCPU.bat" \
TestCPU.bat
```

Index