Farms Batch System *Next Generation* (FBSNG)
Users Guide - Version 1.2

November 13, 2000

Contents

Chapter 1

Introduction

1.1 What is FBSNG?

FBSNG is a batch system designed specifically for a farm based computing architecture. Traditional batch system features such as job submission, job queueing, and load balancing are built into the system. FBSNG runs on all Unix systems supported by Fermilab.

1.2 FBSNG Features

The FBSNG system provides users working in a farm environment with many advantages over previous approaches(such as CPS and CPSBatch used in Run I). Some of FBSNG's major features are:

- Flexible Farm Configuration: FBSNG offers a great deal of flexibility in the configuration of a farm. Most of configuration tasks can be performed dynamically without re-starting FBSNG
- Load Balancing: FBSNG dynamically picks which farm node will run a particular task based on the *perceived* load of the farm. FBSNG does not track resource usage in real time, but assumes that tasks will use resources in a sufficiently well defined manner and that these tasks properly register their processing needs with FBSNG.
- Kerberos support: optionally, FBSNG can use Kerberos v5 to authenticate users or administrators. Also, it can be configured to create Kerberos credentials for batch processes.
- Portability: The majority of FBSNG is written in Python, making it very portable.
- Job Control: FBSNG provides commands to allow users to cancel jobs, or parts of jobs(called sections), as well as change priorities of a section.

- Job Monitoring: Users can get status information on any farm job running on the system. In addition, users can see the status of batch queues and the farm processing nodes.

- Job Queueing: Sections are submitted to batch queues, which may have their own priorities.

- Job History: Job history information is kept by FBSNG for a configurable amount of time. FBSNG provides the `history` command to access this information.

- Abstract Resources: User defined resources can be defined to individual nodes of the system.

- Global Resources: Resources can be defined that are global to the entire cluster.

- Application Programmers Interface (API): An API is provided that allows access to FBSNG internals.

1.3 Structure of FBSNG

FBSNG is composed of a set of daemon processes that monitor, control, and run jobs on the farm nodes. This section is provided to give a brief description of the structure of FBSNG. It is not necessary to know anything about the FBSNG internals in order to use it. This section is purely informational.

- `bmgr`: The `bmgr` (batch manager) runs on the FBSNG control node. It is responsible for accepting new jobs, scheduling, queueing, dispatching, managing resources, and maintaining connections with the `launcher` (see below).

- `launcher`: The `launcher` is a daemon process that is responsible for starting and monitoring user processes. The `launcher` runs on every node in the cluster, communicating status information to the `bmgr` and API..

- `logger`: This daemon is used to log FBSNG status and error messages.

- User Interface (UI): A set of commands that allow users to submit, monitor, and control jobs.

- Graphical User Interface: A GUI implementation of the UI.

- API: The Application Programmers Interface.

1.4 Scope of this document

FBSNG User's Guide describes FBSNG user's interface. Discussion of certain concepts of FBSNG are left beyond the scope of this document. The following is list of other FBSNG documents the user may consult for more information:

- `FBSNG Resources` - describes the FBSNG concept of "abstract resources" and how FBSNG users and administrators should use them to describe computational resources available on the farm and batch job resource requirements;
- `FBSNG Scheduler` - describes FBSNG Scheduler's algorithm and configuration guidelines;
- `FBSNG Installation and Administration Gude` - describes FBSNG installation, maintenance and support procedures including FBSNG Configuration Utility;
- `FBSNG Application Programmer's Interface Reference Manual` - contains detailed description of Python binding of FBSNG API.

Chapter 2

Using FBSNG

This chapter describes how to quickly setup and use the FBSNG system. For a more detailed description of FBSNG, see Chapter 3 - Using FBSNG.

2.1 Setting Up

FBSNG is a Fermilab product available within the UPS/UPD framework [1], and is setup using the UPS `setup` command:

```
setup fbsng
```

At this point, your environment has been appropriately set to run the FBSNG commands.

2.2 Quick Start

This section describes the submission of a simple FBSNG job, and methods used to monitor and control it.

2.2.1 User Interface

All UI commands are executed using the `fbs` script as follows:

```
fbs command arguments ...
```

where `command` is a valid fbs command described below.

`fbs help` will list out all of the available commands, and `fbs command help` will print the usage for a particular command.

[1] Refer to "UPS and UPD v4 Reference Manual" (document number GU0014) for details

2.2.2 Job Submission

The easiest way to submit an FBSNG job is to use `fbs exec` command. For example:

```
fbs exec -q TestQueue /bin/echo Hello world
```

This command submits simple single-section FBSNG job to the queue named "TestQueue". FBSNG will run single instance of a batch process which will produce "Hello world" message. `fbs exec` has a number of optional arguments which allow to submit more sophysticated single-section jobs. This command is described in more details later in this document.

An alternative, more powerful method of FBSNG job submission involves creation of *Job Description File* (JDF). The JDF file contains information that is needed by FBSNG to run your job. A JDF file is composed of one or more *sections*, a section being the basic unit of work in FBSNG.

In the following example, the JDF file contains two sections named `startup` and `main`. Section `startup` will execute two instances (specified by the line `NUMPROC=2`) of `startup.exe`, section `main` will create 10 instances of `main.exe` to run on the farm. The section `main` will not start until all processes from the section `startup` have completed. This is controlled with the `DEPEND=done(startup)` statement. Both sections will send mail to user `xyz@fnal.gov` when the section completes.

```
SECTION startup
        EXEC=startup.exe
        NUMPROC=2
        QUEUE=IO
        MAILTO=xyz@fnal.gov

SECTION main
        EXEC=main.exe arg1 arg2
        NUMPROC=4
        QUEUE=Worker
        MAILTO=xyz@fnal.gov
        DEPEND=done(startup)
        STDOUT=Output/
        STDERR=/dev/null
```

Submitting a farm job is done by running the `fbs submit jdf_file` command. If the name of the JDF file listed above is `my.jdf`, then the job would be submitted as follows:

```
>fbs submit my.jdf
Farm Job 1333 has been submitted...
```

A pathname of a valid JDF file can be (and usually is) supplied as an argument. It is possible to supply all of the information needed by FBSNG on the command line via the `-o` (override) argument.

Under normal operation, the `submit` command will return immediately with a return code of 0. It is possible to have the `submit` command block until the job has completed, or until a certain amount of time has passed. The `-w` (wait) flag will cause `submit` to block until the job (or a section within the job) has completed (see section 3.6.1).

The FBSNG system supplies a unique integer for every job that is started within the system. This unique integer is referred to as the `jobid`, and can be used to reference the job as a whole. For example, one can kill the entire job by issuing the `fbs kill` command (see below) with the `jobid` as an argument. It is also possible to reference individual sections within a job by specifying `jobid.sectionname`. The `jobid.sectionname` is referred to as the *sectionid* throughout the remainder of this document.

2.2.3 Job Monitoring/Controlling

The following commands are supported for monitoring and controlling users jobs:

`fbs kill` - kill a job or a section
`fbs status` - show status of a job
`fbs nodes|hosts` - display status of farm nodes
`fbs wait` - wait for a job or a section to finish
`fbs queues` - display information about the FBSNG queues.
`fbs lj` - displays a short list of active jobs in the system.
`fbs ls` - displays a short list of active sections in the system.
`fbs history` - print historical job information
`fbs slog` - print section log information.

In the above example, status information for the job can be obtained by issuing the following command:

```
fbs status 1333
```

2.2.4 System Monitoring/Controlling

The following commands are supported for monitoring and controlling the FBSNG system:

`fbs hold|release` - hold/release a farm node, job, section, or queue.
`fbs lock|unlock` - lock/unlock a farm queue.
`fbs cprio` - change the priority of a job.
`fbs ptypes` - display the process types defined in the system.
`fbs resources` - display the resources defined in the system.
`fbs monitor` - monitor the system.

2.2.5 Additional Commands

FBSNG also provides other miscellaneous commands:

`fbs config` - used to dynamically modify FBSNG configuration.

`fbs chkcfg|showcfg|getcfg` - used to verify/interrogate configuration files.

`fbs xsysmon` - GUI for monitoring the system.

`fbs sysmon` - Non-graphical monitoring tool.

Chapter 3

FBSNG Details

This chapter describes the FBSNG system in more detail.

3.1 JDF File

The Job Description File (JDF) is the only argument supplied to the `fbs submit` command. It contains all information needed by FBSNG to execute user processes on the farm. The JDF is composed of one or more *sections*. Each section is defined with `Section` line that has a unique label, and contains *fields* that FBSNG uses to determine how and where to run your process. Each field is specified as follows:

```
fieldname=fieldvalue
```

The following table describes the various field names that are allowed:

Field Name	**Required**	**Default**
EXEC	Yes	None
QUEUE	Yes	None
NUMPROC	No	1
DEPEND	No	None
STDOUT	No	File in Home Directory
STDERR	No	File in Home Directory
NEED	No	0
MAILTO	No	None
LEADER_ONLY	No	0
PRIO_INC	No	0
PROC_TYPE	No	Default process type for the queue
PROC_RESOURCES	No	None
SECT_RESOURCES	No	None
SECT_STDOUT	No	None
HOLD_TIME	No	None
NICE	No	0

3.2 JDF Fields

Each field of the JDF file is now discussed.

3.2.1 EXEC

The `EXEC` is a required field that tells FBSNG what executable to run. The pathname given must be visible to the potential farm node that will be running it (via NFS for example). The pathname is relative to the users home area.

3.2.2 QUEUE

`QUEUE` is a required field that specifies which queue to use for job submission. The `fbs queues` command (see section 3.6.15) can be used to find out which queues are defined in the system. Each queue has a default *process type* associated with it. A process type is a name for a group of user processes with the same, or almost the same resource requirements. A job submitted to a particular queue will be assigned that process type, unless the user overrides it.

3.2.3 NUMPROC

`NUMPROC` tells FBSNG how many processes to start on farm nodes. The default value for `NUMPROC` is 1.

3.2.4 DEPEND

The `DEPEND` field contains a dependency string that specifies any dependencies that this section has on other sections. Using dependencies allow for conditional running of certain job sections. The valid conditions that may be supplied are:

- `started(sectionname)` - if the specified section has started running or has already finished, the condition is TRUE.
- `done(sectionname)` - if the specified section has finished successfully, the condition is TRUE, otherwise FALSE.
- `failed(sectionname)` - This will evaluate to TRUE if the section fails, otherwise - to FALSE. `failed` is synonym to `exit`.
- `ended(sectionname)` - if the specified section finished (failed or succeeded) then the condition is TRUE, otherwise FALSE.
- `zombie(sectionname)` - if it is known that the specified section will never start because its dependency condition will never be satisfied, the condition is TRUE, otherwise FALSE.

Simple logical expressions can be combined using parenthesis and the following logical operations:

```
OR  - ||
AND - &&
NOT - ~,!
```

An example follows.

```
DEPEND=done(stepA) && (done(stepB) || !started(stepC))
```

The dependencies in a JDF file determine the order in which sections are executed. At least one section in the JDF must not have a dependency. It is not important to order the sections in the JDF file, FBSNG will automatically determine a valid ordering that satisfies the dependencies. If two sections can run independently of each other, then the ordering of submission to the batch system will be determined by the order in which the sections are defined in the JDF file provided that the needed resources for that section are available.

3.2.5 STDOUT/STDERR

The STDOUT and STDERR fields cause a user process to redirect stdout and stderr of the batch processes to the directory or file specified. The default is to create a file in the users home area with the following form:
FBS_sectionid.logical_pid.[(out|err)]
logical_pid is a unique integer from 1 to NUMPROC assigned by FBSNG to each process in the section. sectionid is the unique section identifier.

If STDOUT/STDERR is specified, FBSNG interprets the string according to the following rules:

- If the string does not start with a '/', the users home area is prepended to the string.
- If the string ends with a '/', the string is interpreted as a directory where the files sectionid.logical_pid.[(out|err)] will be created.
- If the string does not end with a '/', the string is interpreted as the filename that will be created.
- If the string contains '%j', it will be expanded with sectionid
- If the string contains '%n', it will be expanded with the logical_pid of the process.
- If the string contains '%J', it will be expanded with jobid.
- If the string contains '%S', it will be expanded with sectionname.

When a batch process starts on a farm node, it will have the environment variables FBS_PROC_STDOUT and FBS_PROC_STDERR will have the names of stdout and stderr files to respectively.

3.2.6 `NEED`

`NEED` is a flag that tells FBSNG how to handle a job if user processes terminate abnormally. If `NEED` = 1, then FBSNG will cancel the job (which implies killing all user processes). If `NEED` = 0, then FBSNG will let the remaining processes complete.

3.2.7 `MAILTO`

When a section completes, mail will be sent to the address given in the `MAILTO` line. The string provided in the `MAILTO` line will be passed directly to the mail program running on that system. Any valid string that the mail program accepts is valid on this line. For instance, most mail programs accept a comma separated list of email addresses and sends the mail to each of those addresses. This can be coded in the `MAILTO` line of the JDF file.

3.2.8 `AFTER` or `HOLD_TIME`

One can delay the starting of a section until a particular time by using the `HOLD_TIME` JDF line. The time string can be specified as either an absolute time, or one of two relative times:

- Absolute: `hour:[min:[sec]]` - Hold job until the specified time today. If this time has passed, then the job will be ready for immediate execution.
- Relative(1): `+day[-hour:[min:[sec]]]` - Hold job until the `day`-th day from now at the time specified.
- Relative(2): `+hour:min:sec` - The time specified refers to the amount of time from the current time. The job will be eligible for release at this time.

EXAMPLES:

- `10:00:00` - This is absolute time format 1, and translates to 10:00 today. If 10:00 has passed, then the job will be immediately eligible for execution.
- `+10:30:00` - This is relative format 2, and refers to 10 hours and 30 minutes from now.
- `+10` - This is relative time format 2 and translates to 10 hours from now.
- `+10-` - This is relative time format 1 and translates to 10 hours from now.
- `+1-10:00:00` - This is relative time format 1 and translates to one day from today (tomorrow) at 10:00.
- `+0-10:00:00` - This is a special case, and means the next time that it is 10:00. It does *not* mean 0 days from now(today) at 10:00. If the command is issued on Monday at noon, then the string means Tuesday at 10:00. If

the command is issued on Monday at 8:00, then the string is interpreted to be Monday at 10:00.

AFTER and HOLD_TIME are synonyms.

3.2.9 LEADER_ONLY

This feature is implemented to support MPICH-like parallel jobs. If this value is set to 1 in the JDF file, then FBSNG will start a user written wrapper program on the allocated nodes. The master process (FBS_PROC_NO == 1) will actually spawn the processes. FBSNG will provide the user with the list of nodes that have been allocated via the FBS_HOSTS environment variable.

3.2.10 PRIO_INC

The initial section priority relative to the normal initial priority.

3.2.11 PROC_TYPE

By default, when a job is submitted to a batch queue the user processes are assigned the process type associated with that queue. A process type is simply a definition of the resource requirements used by a particular class of jobs. For example, there may be a queue setup that is designated for processes that need 2GB of disk and use 75% CPU. A process type would be defined in FBSNG configuration that specifies these specific resource requirements. It may be necessary to submit a job to this queue that does not fit this process type. Overriding a queues default process type is done by the PROC_TYPE JDF field. FBSNG will use the process type specified in this field rather than the process type specified for the queue if this is specified.

3.2.12 PROC_RESOURCES

Process types define a set of base resources to use. There may be a process that needs additional resources in addition to those specified in the process type. FBSNG allows for user processes to specify additional processes using the PROC_RESOURCES JDF field. This tells FBSNG what resources to use for each process in addition to those defined for the process type being used (either the default process type for the queue the job was submitted to or the overridden process type given by the PROC_TYPE JDF field discussed above). If the resource specified in the PROC_RESOURCES field is already defined in the process type being used for this job, then this value will override the value of the process type provided that the resource usage specified *is greater* than that defined in the process type. In other words, it is not possible to specify 50 units of a resource called cpu if the process type being used is defined such that it uses 75 units of cpu.

The format of the specified resources is:

```
resource1:n resource2:m ...
```

For example, assume that a job is to be submitted that to specify 22 units of a resource named BWO and 18 units of a resource named PMD would be coded as follows:

```
PROC_RESOURCES=BWO:22 PMD:18
```

3.2.13 SECT_RESOURCES

Tells FBSNG what resources to use for a particular section in addition to those specified by FBSNG configuration for the section process type. The format of the line is the same as PROC_RESOURCES (see section 3.2.12)

3.2.14 SECT_STDOUT

Tells FBSNG where to put the output of the section to. This is not the output that may be written by a process to stdout, but is messages generated internally by FBSNG such as how many processes were started, where the processes were placed, etc. The directory where section output is to be placed must be protected in such a way that it can be written into by the account FBSNG runs under.

3.2.15 NICE

One can run FBSNG processes at higher UNIX nice level, which will make them run at lower UNIX scheduling priority. If NICE is specified, all user FBSNG processes will run at the nice value specified.

3.3 Section Names

FBSNG requires that each section within a JDF be defined by specifying the SECTION keyword:

```
SECTION section_name
```

Each section name must be unique, and can be made up of letters, numbers, and the special characters dash and underscore. Section names are used as labels for dependency clauses. A section is terminated by either another SECTION declaration, or the end of the JDF file.

3.4 Process Exit Codes

Each process reports it's exit code upon completion. There are special FBSNG process exit codes that FBSNG generates. The following list describes these special exit codes:

- 100 - setuid/setgid failed
- 101 - failed to open stdout file
- 102 - failed to open stderr file
- 103 - user home directory was not found
- 104 or 105 - failed to start specified executable

3.5 User Environment

User processes running on the farm lose the environment in which they were submitted. When a user process begins running on the farm, the only environment that the process has is:

- FBS_JOB_SIZE: Number of user processes in the section.
- FBS_PROC_NO: logical_pid of the process.
- FBS_JOB_ID: The FBSNG job id.
- FBS_SECTION_NAME: The FBSNG section name.
- FBS_HOSTS: A list of nodes assigned to this job.
- FBS_SCRATCH: Temporary directory created for the process on one of local scratch disks. This directory with its contents will be automatically deleted after the process exits.
- FBS_PROC_STDOUT: Specifies the pathname where the process' `stdout` will be written to.
- FBS_PROC_STDERR: Specifies the pathname where the process' `stderr` will be written to.
- FBS_PROC_POOLS: Mapping between resource pool names which were requested for the process and names of actual resources allocated.
- FBS_SECT_POOLS: Mapping between resource pool names which were requested for the section and names of actual resources allocated.
- FBS_SCRATCH_requested_pool_name: if a scratch disk space was allocated through a resource pool, this environment variable will point to corresponding disk area. For example, if the user requested certain amount of a resource from pool "sratch", and actual disk space was allocated on the disk "scratch3", then the environment variable FBS_SCRATCH_scratch will point to the temporary directory created on the "scratch3" disk and the value of FBS_PROC_POOLS will contain "scratch:scratch3".
- HOME: The users home area.

3.6 FBSNG Command Line Interface

All FBSNG commands are executed with the `fbs` script:
`fbs [fbs_command][args]`
The following commands are currently implemented:

```
submit
exec
wait
status
history
kill
msg
monitor
nodes
xsysmon
sysmon
hold|release
lock|unlock
queues
ptypes
jobs
resources
cprio
slog
```

3.6.1 `submit`

The format of the submit command is:

```
fbs submit [-w [-t <timeout>]][-o <override_string>]...
           [jdf_file>]
```

If the job is successfully submitted, the command will return a status of 0 and write the farm job id to `stdout`. This id can be used by other FBSNG commands to control and monitor the job. The jobid refers to all user processes associated with this JDF. The following example submits the JDF file `sleep_well`:

```
fbs submit sleep_well
Farm Job < 6505 > has been submitted ...
```

The `-w` argument cause `submit` to block until the job finishes. If the `-t` option is supplied, then the command will wait at most `timeout` minutes. `submit` returns with the following codes if the wait option is specified:

- 0 if all sections ended with a return status of 0.
- 1 if one or more sections ended with a return status not equal to 0.
- 2 for all other errors.

It is possible to override fields in the JDF file. This allows one to create a template JDF, and then override particular fields within the JDF. Using the override flag, one can also specify all information needed by FBSNG, eliminating the need to specify a JDF file on the `submit` command. The override string is composed of at least one `keyword=value` pairs. Valid keywords are:

```
SECTION
EXEC
QUEUE
NUMPROC
DEPEND
STDOUT
STDERR
NEED
MAILTO
NICE
LEADER_ONLY
SECT_STDOUT
PROC_RESOURCES
SECT_RESOURCES
PROC_TYPE
HOLD_TIME
```

If `value` has whitespace characters in it, then it must be enclosed in single quotes. If `SECTION` is omitted, then the override will be applied to all sections of the job. For example, if a user submitted `test.jdf`, which was composed of two sections `one` and `two`, the following would cause both sections to execute 10 copies of the executable:

```
-o ''NUMPROC=10''
```

If the JDF file is not supplied as an argument to the `submit` command, then the values supplied with the `-o` flag will be used to construct the users job. When a JDF file is not supplied, a `SECTION` must be given in the override string.

The following are some examples of using the override string.

Example 1 - Override the `EXEC` field for step `main`

```
fbs submit -o " SECTION=main EXEC='my.exe arg1 arg2'" my.jdf
```

Double quotes surround the entire string, single quotes surround the EXEC string.

Example 2 - Override the `NUMPROC` field of section `one` and override the `EXEC` field of section `two`.

```
fbs submit -o " SECTION=one NUMPROC=5" -o
   "SECTION=two EXEC='my.exe arg1 arg2'"
```

Again, double quotes are used to surround the entire override strings, single quotes are used to surround substrings within the override string (in this example, `my.exe arg1 arg2`). Notice on the second example that two override flags were given to the `submit` command. Both overrides will be applied, provided that the section names are unique. If the section names are the same, then the last override string will be processed and the preceding override strings ignored.

3.6.2 wait

The `wait` command will block until a specified job or section has completed. The format of the command is:

```
fbs wait [-t <time_out>] ([-s <section>] <jobid>|<sectionid>)
```

If `-s` is supplied, the command will wait for `section` to complete. If `-t` is supplied, the command will return with `137` if the section or job being waited on did not complete in `time` minutes. If all of the sections of the job being waited on end with a status of `0`, then the command returns `0`, otherwise a `1` is returned. If an error occurred, then `2` is returned. The section name can be supplied either with the `-s` flag or in the `jobid` string. If it is specified in the `jobid` string, the format must be `jobid.section`. If a section is supplied in both places, the `-s` flag is ignored.

Examples

```
fbs wait 123
   Wait indefinetly for all sections of job 123 to complete.
fbs wait -t 5 123
   Wait for a maximum of 5 minutes for all sections of job 123  to
   complete.
fbs wait -s one 123
   Wait indefinetly for section one of job 123 to complete.
fbs wait one.123
   Wait indefinetly for section one of job 123 to complete.
```

3.6.3 exec

The `exec` allows to execute shell commands as batch processes without creating a JDF file. This command submits simple one section batch job. Also, this command can be used to run batch processes in interactive mode. The format of the command is:

```
fbs exec -q <queuename> [<other options>] <command> [<arguments> ...]
```

Command options are:

- -q ¡queue name¿ - submit the section into specified queue, this is required parameter, similar to QUEUE JDF field;
- -I - run batch processes in interactive mode;
- -v - verbose output including equivalent JDF file;
- -s ¡section name¿ - assign section name, default is Exec, similar to section name in JDF;

- -n ¡numproc¿ - run ¡numproc¿ parallel batch processes, default is 1, similar to NUMPROC JDF field;
- -p ¡proctype¿ - use the specified process type, default is determined by the queue the section is submitted to, similar to PROC_TYPE JDF field;
- -w ¡wait time¿ - wait for completion of this job for specified amount of time. Wait time can be specied as integer number of seconds, or -1 or ”forever” to make it wait indefinitely. This option is ignored in interactive mode;
- -h ¡hold time¿ - hold the job until specified time, similar to HOLD_TIME or AFTER JDF field;
- -r ”¡resource¿:¡amount¿ ... ¡attribute¿” - request additional resources for each process, similar to PROC_RESOURCES JDF field;
- -i ¡increment¿ - submit the job at higher or priotity (minimal queue priority plus the specified increment), similar to PRIO_INC JDF field;
- ¡cmmand¿ [¡arguments¿] - the command to be executed by batch processes, similar to EXEC JDF field.

Example

This is an example of using `exec` command to submit a simple job:

```
fnpcb> fbs exec -v -q LongQ -n 5 -r disk:3 /bin/sh -c "ls -l /tmp"
SECTION Exec
NUMPROC = 5
QUEUE = LongQ
EXEC = /bin/sh -c 'ls -l /tmp'
PROC_RESOURCES =  disk:3

Job 2536
fnpcb>
```

In this example, the user submitted 5-process section into LongQ queue with additional requirements of 3 units of ”disk” resource per process. Each process was to execute the specified command. Because -v option was specified, JDF equivalent of submitted job was printed out.

Interactive Mode

If `exec` command is used in interactive mode (with -I option), it establishes interactive connection between the terminal where the job is submitted from and the batch processes as soon as they start. If the submitted section has more than one process, (as specified with -n option), user can switch the connection from one running process to another. In order to switch to another process, user has to press control-] and then process number. Here is an example of interactive batch session:

```
fnpcb> fbs exec -I -n 3 -q IOQ /bin/tcsh
Job 2534

[Connected to process #1]
Exit from .cshrc
fnpcb> ps
  PID TTY          TIME CMD
21447 ttyp0    00:00:00 tcsh
21933 ttyp0    00:00:00 ps
fnpcb>
# to switch to proc #, ? to list, q to quit, k to kill> 2

[Connected to process #2]
Exit from .cshrc
fnpcb> hostname
fnpcb
fnpcb>
# to switch to proc #, ? to list, q to quit, k to kill> k

Section state is:  failed
fnpcb>
```

In this example, user submitted 3-process batch job to run 3 instances of interactive UNIX shell (/bin/tcsh). When the job started, user connected to process 1, then switched to process 2, then killed the job.

3.6.4 status

The `status` command allows one to monitor status of farm jobs. The output of the command will display information on all user processes associated with this job. One can view a specific job or jobs associated with a particular user. If no arguments are given, then `status` will report on all jobs associated with the user id of the person issuing the command. The format of the command is:

```
fbs status [-u (<user>|all)] [-r] [-q queue]
    [(<jobid>|<sectionid>)]
```

Examples

```
fbs status 123
   Get the status on job 123.
fbs status -u maryanne
   Get status on all of user maryanne's jobs
```

```
  fbs status -u all
     Get status on all user jobs.
```

Here is an example of the output produced by the command:

```
> fbs status 2528
Job ID:  2528
  Section ID: 2528.Exec   Name: Exec

  Process resources: cpu:100
  Section resources:
  Start Time: Wed Nov  1 11:00:37 2000    End Time: Not Finished
  Hold Time:                              Prio: 0
  Exec:       fbsng/test/sleep.sh 10 30
  State:      running                     Depend:
  ProcType:   Worker                      Queue: LongQ
  NumProc:    3                           Nice: 0

  Process #1  (2528.Exec.1)                  on fnpc105    Status: exited:0

  Process #2  (2528.Exec.2)                  on fnpc106    Status: running
  PID           CPU   ACPU Command
  ---------- ------ ------ ----------------------------------------
  28238           0      0 csh fbsng/test/sleep.sh 10 30
  28309           0      0  python /home/ivm/fbsng/test/sleep.py 10 30

  Process #3  (2528.Exec.3)                  on fnpc107    Status: running
  PID           CPU   ACPU Command
  ---------- ------ ------ ----------------------------------------
  25738           0      0 csh fbsng/test/sleep.sh 10 30
  25809           0      0  python /home/ivm/fbsng/test/sleep.py 10 30
  -------------------------------------------------------------------------
```

In this example, the users jobs created several subprocesses. Process 20304 is the parent, processes 20305 and 20306 are direct descendents of 20304 (shown by indentation of one space.). Process 20307 is a descendent of 20305.

In the above example, the users job requested a resource pool `disk` on a per process basis. The `Pool Info:` line for per process info indicates that on this host the underlying resource given to this process was `disk1`.

One can get a raw listing instead of a formatted listing by using the `-r` option. This output will more suitable for parsing.

3.6.5 `history`

The `history` command provides one with a history report of jobs that have completed. The format of the command is:

```
fbs history [-b <begin_time>][-e <end_time>]
            [-u ('<user> ...'|all)] [-s '<section>...']
            [-h '<node>...'] [-q '<queue> ...']
            [-j ('<jobid>|<sectionid>) ...']
            [-t '<process_type>...']
```

`-b <begin_time>`: Report on only jobs starting after `begtime`. The default is to look as far back in the history files as possible.
`-e <end_time>`: Report on jobs ending before `endtime`. The default is to show jobs up to the current time.
`-u <user> ...|all`: Report on jobs for a particular user(s). The default is to only show jobs for the user issuing the command.
`-s <section> ...`: Report on `section`. If the section has been specified with the `-j` option, then the `-s` is ignored. `-h <node> ...`: Report on processes running on specified nodes. The default is to show processes on all nodes.
`-q <queue> ...`: Report on processes running on specified queues. The default is to show processes on all queues.
`-j <jobid>|<sectionid> ...`: Report on specified jobs. Each `job_id` can take the form `section.jobid`, which will cause only `section` information for that job to be printed.
`-t <proc_type> ...`: Report on sections of specified process types.

3.6.6 kill

The `kill` command can be used to kill an individual farm job section, or the entire farm job to be killed. If a section name is not provided, the `kill` command will kill all user processes associated with the farm job. A list of farm jobid's can be provided. The format of the command is as follows:

```
fbs kill [-s <section>] [-i] (<jobid>|<sectionid>) ...
```

The following command kills the job submitted in the previous section:

```
fbs kill 6505
Farm job < 6505 > killed ...
```

If a sectionid is given to the command as an argument, then `kill` will cause an individual section of the specified job to be killed. In this form, any section names provided with the `-s` flag will be ignored for this job.

By default, `kill` causes a `SIGINT` to be sent to all user processes. Users can catch this signal and perform some sort of cleanup. After some grace period (defined in an FBSNG configuration file), `SIGKILL` is sent to the users processes. Users can cause a job to be killed immediately with the `-i` flag. In this case, `SIGKILL` is sent to the users processes immediately.

Examples

`fbs kill 123`
Kills all sections of job 123.

`fbs kill -s main 123`
Kills just section `main` of job 123.
`fbs kill 123.main`
Kills just section `main` of job 123.
`fbs kill -i 123`
Immediately kills all sections of job 123.
`fbs kill 123 234 345`
Kills all sections of jobs 123, 234, and 345.
`fbs kill -s main 123 234 345`
Kills just section main of jobs 123, 234, and 345.
`fbs kill -s ''main output'' 123.one 234 345`
Kills just section one of job 123, and sections main and output of jobs 234 and 345.

3.6.7 msg

The `msg` command is used to pass a message to the `status` command. When `msg` is called, it will cause a message supplied by the user to be displayed when using the `status` command. Users can issue this command at various points in their executable to track the progress of their job. For instance, a message could be sent describing what record is being processed ("Record 1000 is now being processed"). The format of the command is:

```
fbs msg <user-message>
```

3.6.8 monitor

The `monitor` command invokes the GUI interface to FBSNG. There are no options or arguments to this command.

3.6.9 nodes

The `nodes` command provides users with information about the farm cluster. The short listing displays the status of each node, the process types associated with the node, and ids of job sections that are running on the node. The format of the command is:

```
fbs nodes [-d][-l][-t '<class> ...'][<node> ...]
```

The following is an example of the output:

```
> fbs nodes
  HOST               STATUS CLASS       PROCESSES
 fnpc112.fnal.gov    up     IO          ['step1.1547.1']
 fnpc5.fnal.gov      down   Worker      []
```

The number in parenthesis after the job number indicates the number of user processes associated with that job that are running on that node.

The CLASS column shows the node class.

The PROCESSES column shows what processes are currently running on that node. The processes are shown in the form:

```
sectionid.logical_pid
```

The nodes also has a long format that displays load averages and the reason that a node is down if applicable. The following two commands illustrate this output:

```
[farms@fnpc112 tests]> fbs nodes -l
---------------------------------------------------
HOST: yogi              CLASS: Worker          STATUS: up
RESOURCES:  {'disk': (0, 18), 'cpu': (90, 300), 'blue': (0, 0)}
PROCESSES:  ['39.Start.1']
```

The resource values are represented by a pair of values (used,capacity). In the example above, the resource cpu has a total capacity of 300, 90 of which are allocated. The -d option displays only nodes which are "down".

A node may be in one of four states:

up - The node is up and running, jobs are allowed to be dispatched to this node;

down - FBSNG detected a node failure and disabled the node until the failure condition is cleared. Status of batch processes running on the node is unknown.

held - This indicates that jobs are not allowed to be dispatched to this node.

released - This indicates that jobs are allowed to be dispatched to this node. The -c <class>... flag causes the command to only print information about nodes that belong to a particular class.

3.6.10 hosts

The hosts is simply an alias for nodes. Refer to section 3.6.9 for details.

3.6.11 xsysmon

This simply starts the GUI systems monitoring tool. The format of the command is:

```
fbs xsysmon [(<node>...|<node_prefix><start><end>
         |-j (<jobid>|<sectionid>))]
```

3.6.12 sysmon

The sysmon command provides an ASCII interface to the system monitoring tools. The format of the command is:

```
fbs sysmon [(-a|-cpsdxlTiIOeDCg)] [-r <interval>]
           [-n <times>] <node> ...
```

The flags are now described:
`-c`: Print the CPU usage column
`-p`: Print the paging rate.
`-s`: Print swap space information
`-i`: Print interrupt information.
`-d`: Print disk usage
`-x`: Print context switch column
`-l`: Print load information
`-T`: Print the TPKTS column
`-I`: Print the IPKTS column
`-O`: Print the OPKTS column
`-C`: Print the COLLS column
`-e`: Print the ERRS column.
`-a`: Print all columns
`-g`: Print a glossary. This prints the titles of the columns that would have been displayed, delimited by `|`.
`-r secs`: Causes the output to be repeated every `secs` seconds.
`-n repeatnum`: Causes the output to be repeated `repeatnum` times. The default is to repeat forever.

3.6.13 `hold,release`

The `hold` and `release` commands change the status of a farm node, a farm queue, or a farm job/section. If not specified, the command will operate on `node`. The format of the `hold` command is:

```
fbs hold node [-r '<reason>'] <node> ...
fbs hold queue <queue> ...
fbs hold (job|section) [-t <time>] (<jobid>|<sectionid>) ...
```

The format of the `release` command is:

```
fbs release node <node> ...
fbs release queue <queue> ...
fbs release (job|section) (<jobid>|<sectionid>) ...
```

The `-r` flag is only valid when holding a node. `-t` is ignored when releasing a queue or node. When releasing a job, the `-t` is not valid. If holding/releasing a job or section, a time can be specified when to perform the action. Refer to section 3.2.8 for a detailed explanation of `time_str`

- `mm/dd/yyyy-hh:mm:ss` for specifying an exact time.
- `+dd-hh:mm:ss` for specifying relative times.

3.6.14 lock,unlock

The `hold` and `release` commands lock/unlock a FBSNG queue. The format of the `lock` and `unlock` command is:

```
fbs queue lock <queue> ...
fbs queue unlock <queue> ...
```

Locking a queue will forbid any job from being submitted to the queue.

3.6.15 queues

The `queues` command shows information about the FBSNG queues. The format of the command is:

```
queues [-l] [<queue_name> ...]
```

Example of output:

```
fnpcb> fbs queues
Name            State           Def Proc Type    NPend   NRun NTotal
----            -----           -------------   ------ ------ ------
FastQ           Active          Worker               0      0      0
IOQ             Active          IO                   0      0      0
KrbQ            Active          KWorker              0      0      0
LightQ_1        Active          Light                0      2      3
LightQ_2        Active          Light                0      3      3
LightQ_4        Active          Light                1      2      3
LightQ_8        Active          Light                0      3      3
LongQ           Active          Worker               0      0      2
Q2              Active          Worker               0      0      0
Q5              Active          Worker               0      0      0
```

For each queue, the output includes name of the queue, queue state (active, held, locked), default process type, number of pending and running sections and total number of sections including done, canceled, etc.

The `-l` flag causes a long listing to be displayed. For example:

```
fnpcb> fbs queues -l LongQ
Queue: LongQ                Status: Active     Def Proc Type: Worker
MaxSPrio: 100    SPGap: 10     MaxQPrio: 100
MinQPrio: 0      QPDec: 1      QPInc: 1      QPGap: 10
Sections: Pending: 0  Running: 1  Total: 1
   2556.P3:*
-------------------------------------------------------------------------
```

The following information is displayed in the long version:

- Queue - The name of the queue.

- Status - The status of the queue.
- Def Proc Type - The default process type.
- MaxSPrio - Maximum section priority for this queue.
- SPGap - Section priority gap.
- MinQPrio - Minimum priority of this queue.
- MaxQPrio - Maximum priority of this queue.
- QPDec - Queue priority decrement.
- QPInc - Queue priority increment.
- QPGap - Queue priority gap.
- Number of pending and running sections
- Total number of sections including those canceled, done, etc.
- List of sections and status for each section.

3.6.16 ptypes

The `ptypes` command lists the process types defined in the system. The format of the command is:

```
fbs ptypes [-lruq] [<proc_type> ...]
```

The meaning of the flags are as follows:

- -l - Produce a long listing.
- -r - Print out default resources requirements information.
- -q - Print resource allocation quota.
- -u - Print current usage of resources.

The following shows the output of a long listing:

```
>fbs proc_types -l
-----------------------------------------------------------------
PROCESS TYPE                 : IO
DEFAULT PROCESS RESOURCES    : disk:10 IOcpu:50
DEFAULT SECTION RESOURCES    : scratch:5
RESOURCE USAGE QUOTA         :
RESOURCES IN USE             :
MAXIMUM PRIORITY INCREMENT   : None

-----------------------------------------------------------------
```

```
PROCESS TYPE                  : Worker
DEFAULT PROCESS RESOURCES     : cpu:90
DEFAULT SECTION RESOURCES     :
RESOURCE USAGE QUOTA          : cpu:1000 nfs_disk:18
RESOURCES IN USE              : cpu:0 nfs_disk:0
MAXIMUM PRIORITY INCREMENT    : None
```

3.6.17 lj

The `lj` command produces a short listing of jobs running in the FBSNG cluster. The format of the command is:

```
fbs lj [-u (<user>|all)] [(<jobid>|<sectionid>] ...
```

By default, all jobs are displayed. -u option can be used to display only jobs of specified user. The following is an example of

```
fnpcb> fbs lj
JobID  State      User     Sections
------ ---------- -------- ----------------------
2540   running    ivm      Light_1:* Light_2:d Light_4:* Light_8:*
2541   ended      ivm      P3:d
2542   running    ivm      Light_1:* Light_2:* Light_4:* Light_8:*
2543   ended      ivm      P3:d
2544   ended      ivm      P3:d
2545   running    ivm      Light_1:* Light_2:* Light_4:* Light_8:*
2546   running    ivm      Light_1:* Light_2:* Light_4:r Light_8:r
2547   pending    ivm      Light_1:r Light_2:r Light_4:r Light_8:r
fnpcb>
```

For each job, the command prints its id, state, username and list of sections with status of each section represented with single character:

- w - waiting
- r - ready
- h - held
- * - running
- d - done
- f - failed
- c - canceled
- z - zombie

3.6.18 ls

The `ls` command produces a short listing of FBSNG sections. The syntax of the command is:

```
fbs ls [-u (<user>|all) [-q queue] [<sectionid> ...]
```

Example of output:

```
fnpcb> fbs ls
SectID          User     St Queue      Pr  Proc Type  Nproc   Time
-----           ----     -- -----      --  ---------  -----   ----
2548.Light_2    ivm      *  LightQ_2   0   Light       8/10  11/01 12:21:55-Str
2548.Light_1    ivm      d  LightQ_1   0   Light       0/10  11/01 12:24:02-End
2548.Light_4    ivm      d  LightQ_4   0   Light       0/1   11/01 12:23:59-End
2548.Light_8    ivm      *  LightQ_8   0   Light      10/10  11/01 12:22:01-Str
2549.Light_2    ivm      *  LightQ_2   0   Light      10/10  11/01 12:22:08-Str
2549.Light_1    ivm      *  LightQ_1   0   Light       8/10  11/01 12:21:58-Str
2549.Light_4    ivm      *  LightQ_4   0   Light       5/5   11/01 12:22:12-Str
2549.Light_8    ivm      *  LightQ_8   0   Light       3/10  11/01 12:23:21-Str
2550.Light_2    ivm      *  LightQ_2   0   Light      10/10  11/01 12:22:15-Str
2550.Light_1    ivm      *  LightQ_1   0   Light      10/10  11/01 12:22:05-Str
2550.Light_4    ivm      *  LightQ_4   0   Light      10/10  11/01 12:22:18-Str
2550.Light_8    ivm      *  LightQ_8   0   Light      10/10  11/01 12:23:32-Str
```

For each section, the output includes section id, username, section status, section priority, process type, number of running processes, total number of processes and date/time of last section state change.

3.6.19 resources

The `resources` command lists the resources defined in the system. The format of the command is:

```
fbs resources [-g][-l][-n<node>][<resource>...]
              [-p <proc_type>][<resource>...]
```

The following example shows resources available on the farm:

```
fnpcb> fbs resources
Resource          Type Usage Capacity
-----------       ---- ----- --------
nfs_disk             G     0       18
Blue                 L     0        0
IO                   L     0     attr
Krb                  L    13     attr
Worker               L     0     attr
cpu                  L   130     1800
disk1                L     5       98
```

```
disk2              L     8       20
scratch            L     0        0
disks              L    13      118 (pool: disk1 disk2)
fnpcb>
```

Word "attr" in capacity column indicates node attribute. In case of node attribute, usage column indicates how many processes use the attribute. For resource pools, the output includes composition of the pool.

3.6.20 `cprio`

The `cprio` command changes the priority of a section. The format of the command is:

```
fbs cprio [+]<n> <sectionid> ...
fbs cprio -<n> <sectionid> ...
```

`+` and `-` specify whether to increment or decrement the priority of a section by `n`. If neither is specified, the default is to increase the priority.

3.6.21 `slog`

The `slog` command prints out detailed section log to `stdout`. The format of the command is:

```
fbs slog <sectionid>
```

Chapter 4

Sample JDF Files

This section is provided to give users a flavor for how to code JDF files. Several examples will be given to illustrate various features.

4.1 Example 1

The following JDF contains three sections, `input_phase,main,output_phase`.

```
SECTION input_phase
        EXEC=input.exe arg1 arg2
        QUEUE=input_queue
        NUMPROC=1
        SECT_RESOURCES=red nfs_disk:1
        PROC_TYPE=Test

SECTION main
        EXEC=bin/analysis.exe arg1 arg2 arg3
        QUEUE=main_queue
        NUMPROC=5
        DEPEND=started(input_phase)
        PROC_RESOURCES=Linux
        STDERR=%J.startup.err
        STDOUT=%J.startup.out
        NEED=1

SECTION output_phase
        EXEC=/home/user/bin/output.exe arg1 arg2 arg3 arg4
        QUEUE=output_queue
        DEPEND=done(input_phase) && done(main)
        NUMPROC=1
```

This job consists of three sections. `input_phase` will run one instance of `input.exe` found in user's home directory, and, in addition to resources defined for the process type "Test", will use the attribute `red` and 1 unit of `nfs_disk` resource. This section will be submitted to the `input_queue` queue.

The `main` section will run when the `input_phase` section has been started. It will run 5 instances of executable `analysis.exe` found in subdirectory "bin" user's home directory, Processes of this section will run only on nodes with attribute `Linux`. The section will be submitted to `main_queue`. `NEED=1` means that the section will be considered as failed and all its processes will be killed if any of them fails.

The final section of this job is the `output_phase` sections, and will only be dispatched to run if and when the `input_phase` and `main` sections succeed. Otherwise, if any of them fails, this section will become "zombie" and will be treated as canceled. This section will run one instance of the executable `output.exe`

4.2 Example 2

The following example shows how the environment variable `$FBS_PROC_NO` can be used. In this example, the processes running on the farm will each process 1 file. Using `$FBS_PROC_NO`, we can write a script which makes sure that each user process processes a different file than the others. The script can take n arguments, and inspect `$FBS_PROC_NO` to determine which file to process. The relevant code could look as simple as this:

```
#!/bin/sh -f
# Usage: process.sh <file1> <file2> <file3> ...
# This script will use FBS_PROC_NO to choose which of the files
# to process.

# ...

my_number=$FBS_PROC_NO
my_file=${$my_number}
fbs msg "Processing $my_file"
process.exe $my_file

# ...
```

The `EXEC` field in the JDF file would have as it's arguments the names of the files to process and might look like this:

```
SECTION ProcessData
    NUMPROC = 3
    EXEC = bin/process.sh data_file_A data_file_B data_file_C
    ...
```

4.3 Example 3

The following example illustrates the flexibility in the FBSNG dependency clauses. Complex dependency clauses can be created. Since FBSNG automatically orders the executions stream correctly based on the dependency clauses, it doesn't matter which order the sections are entered into the JDF file.

```
SECTION A
  EXEC=AAA.exe
  QUEUE=main_queue
  NUMPROC=5
  DEPEND=done(in1)
  STDERR=startup.err
  STDOUT=startup.out
  MAILTO=testuser@fnal.gov
  NEED=1
SECTION B
  EXEC=BBB.exe
  QUEUE=main_queue
  NUMPROC=5
  DEPEND=done(in2)
  STDERR=startup.err
  STDOUT=startup.out
  MAILTO=testuser@fnal.gov
  NEED=1
SECTION C
  EXEC=BBB.exe
  QUEUE=main_queue
  NUMPROC=5
  DEPEND=done(in1) && done(in2)
  STDERR=startup.err
  STDOUT=startup.out
  NEED=1
SECTION ABC
  EXEC=ABC.exe
  QUEUE=main_queue
  NUMPROC=5
  DEPEND=done(A) && (done(B) || done(C))
  STDERR=startup.err
  STDOUT=startup.out
  NEED=1
SECTION out
  EXEC=output.exe
  QUEUE=output_queue
  DEPEND=done(ABC)
  NUMPROC=1
```

```
  NEED=1
SECTION in1
  EXEC=input.exe
  QUEUE=input_queue
  NUMPROC=2
  NEED=1
SECTION in2
  EXEC=input.exe
  QUEUE=input_queue
  NUMPROC=1
  NEED=1
```

FBSNG enforces the dependency chain specified. The order that the sections appear in the JDF file have *no* relevance in determining when a section is to be executed. In the above example, `in1` and `in2` would be immediately eligible to run.