

NAME:

sparchive – Archive merged and failed runs into tar files.

SYNOPSIS:

sparchive [options]

DESCRIPTION:

This command scans the run directories for runs that have been successfully merged or failed and then puts the output files into a tar file in the \$ALLRUNS directory. The run directories are then removed.

OPTIONS:**--background**

Run sparchive in the background.

--debug

Print out lots of debug messages.

-h, or, --help

Print out a brief help message with short descriptions of the options.

--nodelete

Archive the runs but do not delete run directories.

--norun

Do not archive and delete runs, just echo the batch commands to the screen.

-v <version>, or, --verbose

Print out status messages of progress making archive.

EXAMPLES:**Archive the merged and failed runs:**

sparchive -v

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), sparchive(1), sprite(1), spsub(1), spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

spbuild – build the run directories for simulation production

SYNOPSIS:

spbuild [options] [name-value pairs] [run numbers]

DESCRIPTION:

This command builds a set of run directories for simulation production. Mostly it manages all variables and files needed to run the set of simulation executables developed for Babar: bbsim, bogus, simapp, bear and moose. In SP10, there is just one stage (simu) and all the other stages (mixr and reco) can be ignored.

There are several methods of control for this program, and I hope to cover them all here. The first method is through the use of environment variables. The second method is through command line options. And a third method is through information in a database. The command line options are of two types, dashed options and name-value pairs. All dashed options must come before the name-value pairs, and the name-values pairs are separated by an '='.

The run numbers can be a single run or specified as a range of numbers, with a dash separating the numbers, such as 1250101-1250200, to build 100 runs.

Also more than one range of numbers can be placed on the line, such as, "1250101 1250105 1250151-1250200 1250205" to build a set of 53 runs. This option works, but use it with care, you might get run behaviors you do not expect.

If the "-n 100" option is used, spbuild will build 100 runs only.

The "--cycle" options should always be used to specify the run cycle (e.g. SP10). When used, the command will interrogate the database to find out which runs have been allocated to your site for this cycle.

When the "--local" option is used, the root files are written to a local directory on the batch node and copied over (and checksummed) after the job has finished. This can reduce the load. If the environment variable is \$\$PLOCALDIR defined in local-simu-setup, it will be used as the temporary area. If \$\$PLOCALDIR not defined, the default /tmp area is used. If \$\$PLOCALDIR is not defined by the user but is defined by the batch system at run time, then the area defined by the batch system can be used by copying the value to the \$\$PLOCALDIR in simu-local-setup e.g. at CCIN2P3: export SPLOCALDIR="\$TMPBATCH" and at RAL: export SPLOCALDIR="\$WORKDIR".

OPTIONS:**-A, or, --all**

Option for using Moose one step production, instead of the three stage Bogus, SimApp, Bear production. The option is short for all-in-one. A new procspec is created with this option starting with 'A', and there will be no mixr and reco stage for runs using this option.

-B x#, or, --background x#

Sets the background alias. This corresponds to an Objectivity collection defined in the file bkg-coll.db within ProdTools. Needed for the mixr step.

-b <collection name>, or, --boot <collection name>

Sets the objectivity collection used for input and output, over-rides the OO_FD_BOOT env. variable. Usually not used, preferred use is with the env. variable.

-c <cycle name>, or, --cycle <cycle name>

Only build runs for this production cycle (e.g. SP10).

- D** <tag version>, or, **--decayfiles** <tag version>
Use a different version of the ProdDecayFiles package for the decay file. If not defined (e.g. V00-08-89), spbuild will use the latest checked out version.
- fillonly** <site>
An option to put information about the run into the database without building the directories. spbuild will not allow you to put information into the database for run numbers which have already been built.
- g** #, or, **--granularity** #
The granularity used in the mixr step of production. If not used the default is 300.
- G** <3 | 4>, or, **--Geant** <3 | 4>
specify use of geant 3 - bbsim, or geant 4 - bogus. Must be 3 or 4 if used. If not set it defaults to 3.
- GRID**
Create the job.jdl file that is required when running on the Grid.
- h**, or, **--help**
Print out a brief help message with short descriptions of the options.
- j** <group>, or, **--jobgrp** <group>
Choose which group of production you are doing, QA or SP. This is only used in the validation stage (-j valid) to prevent export of the validation runs.
- local** If set, the batch job will write the root file to the directory defined by \$SPLOCALDIR in local-simu-setup on the batch node and copy the files over to the run directory after the job is finished; if \$SPLOCALDIR is not defined, the /tmp area is used. When --local not used (default), the root files are directly written to the run directory while the job is running. Setting the option can reduce the load on the disk holding the run directory.
- l** <site>, or, **--location** <site>
Set the site the production is being done. Usually not used, defaults to \$BFSITE.
- mixrrel** <release>
Specify the release used for the mixr step.
- M** <version>, or, **--mixrver** <version>
Specify the version of the mixr step to use in this production job. This is used so that version of the mixr can be performed again in the same run number, because of errors, or changing configurations.
- n** #, Build # number of runs only.
- norun**
Do not build the directories but just mirror information to the screen. This is no longer supported.
- nodb** Not used any more. The option is still in the system so it does not generate an error if used, but it doesn't do anything anymore. The old description is: Build the run directories without the use of the database. All options needed for running must be specified on the command line for the run to succeed. This option is required when using ProdTools in a personal mode, or for sites without a database of run information.
- q**, or, **--quiet**
Assume a default answer for any prompt, and do not print messages to the screen.
- R** <version>, or, **--recover** <version>
Specify version used for the reco step. This is used when the reco step is run more than once for a given run number, because of errors or changing configurations.
- recorel** <release>
Specify the release used for the reco step.

-r <release>, or, **--release** <release>

Specify the release used for the production job. Must have the form of `##.##x`, `spbuild` will do consistency checks on the release directory using these numbers. Plus control scripts will handle changing run conditions between different releases, using sub-release numbers from this option. (The actual form of the release number for those who know regex's: `^d+\.d+\.d+[a-z]*i.`). This option can also be used to specify the entire path name to a release directory, which can be used to test executables not within the default release directory of `$BFDIST/releases`. The full path name must end in a directory with the form of `##.##x`, and the executables will be run from within the directory in `bin/$BFARCH`.

--skipdir

Do not make the directories, can not be used with the **--nodb** option.

--testdb

Connect to a test copy of the database, not the production database. All interaction with the database will not be logged into production. This option over-written with the **--nodb** option.

-T <F | R | P >, or, **--trigger** <F | R | P >

The trigger to use for production. It must be of type 'F' (full), 'R' (random), or 'P' (perfect). If not set it defaults to 'F'.

-t <simu | mixr | reco >, or, **--type** <simu | mixr | reco >

Specify the type to build, used if only building one step of the production job at a time. If not set all three steps will be built.

-u <user name>, or, **--user** <user name>

The user name of the person building the directories. If not set it defaults to the env. variable `$USER`. If a shared account is used for production this variable must be set.

-V <version>, or, **--version** <version>

Specifies the version number used for all three stages of production. Used if re-doing all steps of the production using the same run number.

ENVIRONMENT VARIABLES:

These variables are used at build time, and influence how ProdTools makes the run directories. These variable can be lost between build time and run time, so the variables which influence running are put into the `confi g.ksh` file. This file is then used by the Job script to set the variables during running.

\$ALLRUNS:

The directory which the run directories will be made, and the `confi g.` files and `log` files will be kept. If not set the default for this is `$BFROOT/prod/log/allruns` when using the `bbrprod` account in production, or `$BFROOT/work/<first letter>/<your name>/allruns` if used in personal account. This variable can be set to produce run directories in other places than the default.

\$BFARCH:

The platform the code is running on, this corresponds to the directory names within `$BFDIST/releases/<release number>/bin`. This has no default and must be set before building. The supported platforms are: Linux2, OSF1V4, SunOS5. This variable gets set at login with HEPiX scripts, or with 'srtpath'.

\$BFDIST:

The root directory for the babar release code. This defaults to `$BFROOT/dist`.

\$BFROOT:

This is the root directory for all babar work. If this is not set it defaults to `/afs/slac/g/babar`.

\$BFSITE:

The name of your site you are running the ProdTools from. This has no default and must be set before building. This variable corresponds to the subdirectories in the 'site' directory within ProdTools. It gets set at login with the HEPiX scripts. Over-written by the option **--location**.

\$DECAYFILES:

A variable for the user to set the directory with the decayfile. Not used in spbuild, but is used at runtime. More detailed note in the spsub man page.

\$PRODOPERATOR:

A variable used in shared production accounts to specify who is building the run directories. Over-written by the option --user.

\$PRODTOOLS:

The ProdTools directory for the scripts you wish to run, the default is \$BFROOT/prod/ProdTools. If you wish to use a different version of ProdTools, or test your own copy, set this variable before using.

\$OO_FD_BOOT:

The Objectivity collection you will be using during production for input and output. This must be set and has no default. It gets set by the commands setboot or a command specific to your production. See simulation manager for your site for how this should be set. This variable can be over-written with the --boot option.

Name-Value Pairs:**RELEASE**

Used in all three steps, specifies the release used for the step of the run. The same as using the --release option.

DECFILE

Used in the simu step, specifies the decay file. This corresponds to a file name in \$BFROOT/prod/ProdDecayFiles.

GENERATE

Used in the simu step, sets the generation variable in the control script.

NEVENTS

Used in the simu step, the number of events to generate.

DESC Used in the simu step, just a description of the events generated. Does not effect running, only get logged in the database.

QAREF

Used in simu step, the QA reference number for the job.

PROD_FD_BOOT

Used in the mixr and reco steps, the Objectivity collection which gets written to.

CONDALIAS

The conditions alias used for the run.

PROCSPEC:

The procspec was created to describe the production in a terse manner. It is used in a couple of places in simulation production. It is used in the directory structure of the production directories, and it is also used in the collection name of the Objectivity output collection. The form of the procspec is this:

X##.##xV#[x#]x, or, <mode><release><version>[background][trigger]

or if you know regex's:

[BGMRA]\d+\. \d+\. \d+[a-z]?V\d\d(x0?\d\d)?[FRP]?

And if you are confused, so am I, and if it looks hard to define, you are right. But this is what people want supported. So, what is this string of information anyway.

It starts with a single capital letter, B, G, M, R and now A to state the step of the production and which application is being run. B is bbsim, G is bogus, M is simapp R is bear, and now A is Moose. They are meant to specify the three steps of production, simu, mixr, reco, but the new bogus messed it up. The letter A is for the single stage production with Moose, and this is short for All-in-one production.

The next string of information (`(\d+.\d+.\d+[a-z]?)`) is the release, which must have a major number, sub-release number, and bug fix number. And this can also have the optional lettered release.

The next string of information (`(V\d\d)`) is the version. There can be more than one running of mixr and reco for one run in production, due to errors in running or changing conditions. The step of running could have died due to problems with database, computer, networking, bad hair day or other problems. It was decided never to over-write an Objectivity collection, but to create a new one. This meant that if you wanted to finish the job, you had to create a new collection after a job crash. This new collection has to have a different name, hence a different procspec, hence a new version number. The first time running the version number is V01, and increments up from there, for each step.

The next string of information (`(x0?\d\d)`) is used only for the mixr stage or Moose, and it is the alias to the background collection. The actual Objectivity collection is specified in the `bkgcoll.db` file within ProdTools. This string does not end there, there is also another form which is `x0#`, where a 0 is before the number. This is for running a mixr step without mixing in backgrounds. The `bkgcoll.db` file also has the conditions alias specified, so the number has to be specified to get the correct conditions alias, but to state that you want no background mixing the 0 is placed before the number. Confused, good, me to.

The end of the procspec is also only used in the mixr step, this is a capital letter F, R, or P. This specifies the type of trigger used, see the trigger option for details.

How the Procspec is built:

The first letter is decided by the type being built: simu defaults to will be 'M'; and reco will be 'R'. Moose will be 'A'.

The release is chosen from the release specified in the `--release` option, or the `RELEASE` name-value pair. If these are not set then it looks for information from the database, and uses that. This information can then be over-written in the mixr and reco steps by using the options `'--mixrrel'`, and `'--recorel'`.

The version of each stage of running defaults to 'V01' if nothing is set. This is over-written by the `--version` option, which will set the version for all stages of production. The mixr and reco stage versions can be over-written with the `--mixrver` or `--recover` options. These options supersede the `--version` option.

The next option is the background index used in the mixr step. This comes from the `--background` option, and must match a listing in the `bkgcoll.db` file. This defaults to 'x0' or no background mixing.

The last option is the for the trigger in the mixr step. This comes from the `--trigger` option. It defaults to 'F'.

DIRECTORY STRUCTURE:

`spbuild` builds directories, hence its name. The directories contain files necessary for the running of the simulation production jobs. The directories will be created in the `$ALLRUNS` directory. See `$ALLRUNS` notes above for details.

Each production run has a run number, and each run will have its own directory of files, where the top level directory is just the run number. Within this directory is the job directories, and the `status.txt` file. The `status.txt` file will state all the actions ProdTools has performed on the job, and the time.

There are sub directories within the run directory, for each job done in the run. For Moose, there is 1 subdirectory. For production pre-Moose, there will be at least 3 sub-directories for a good run: one for simu files, one for mixr files, and one for reco files. The name of the mixr and simu directories will be the procspec for the mixr and simu jobs, and the name of the reco directory will be the procspec for the mixr job, and then a dash and the procspec for the reco job.

Inside each of these directories, spbuild places three files - config.sh, config.tcl, and the decay file. The config.sh contains all shell variables needed for the running of the job, things such as full path names to the work, release, ProdTools, allruns directories. Mostly everything the running shell script will need to know to check on file structure and start the process running. The config.tcl is sourced by the control scripts for the applications, and contains all information needed to run the application, such as number of events, run number, release number, conditions alias and collections. The decay file will depend on the process being generated.

So putting it together for Moose there will be::

```
<run number> ->
  <simu procspec> ->
    config.sh, config.tcl, decayfile.dec
```

For pre-Moose there will be:

```
<run number> ->
  <simu procspec> ->
    config.sh, config.tcl
  <mixr procspec> ->
    config.sh, config.tcl
  <mixr procspec>-<reco procspec> ->
    config.sh, config.tcl
```

DATABASE INTERACTION

When there is a database to interact with, spbuild will look in a table called prod_run for run definitions. This table has defined run numbers and the simulation requests and decay modes for the run, along with background index and conditions alias and the location for the run. The program will query the database for information about the run number and the request and decay mode, to produce the correct config files for that run. All information for the decay mode will be archived in the data, and spbuild will get that information to define the decay file, filtering, and generation. Also when the run number is defined in the web interface, prod-run records the information on release, number of events, condition alias, and background index to be used for the run. And spbuild will query the database for this information also.

If the database has been filled, then spbuild will only need the run number to correctly build the config files for the run. If you are using a shared production account to do the production, spbuild will also require to know who you are. So the only option on the command line used is '--user <your name>' and this information will go into the config files to record who built the run.

You should always sue the "--cycle" option to select runs from a particular production cycle e.g. SP10.

Also if there is no information about the run in the database, you can specify all information on the command line, and spbuild will insert this information into the database. If the run number already exists in the database, then spbuild will not allow the run to be built, since the information about the run has already been archived in the database, and other options on the command line would allow a run to be built which does not agree with the information in the database. This can cause problem for people who have made

mistakes in building the run, where the database will not allow you to re-build the run with different parameters. In these cases you need to define a new run number, with the new parameters and build the new run.

EXAMPLES:

Building the moose job for 100 runs from SP10

```
spbuild --cycle SP10 --user douglas -n 100
```

Building the moose job for all runs from SP10

```
spbuild --cycle SP10 --user douglas
```

Making all three steps, without database, specifying needed information on the command line:

```
spbuild --nodb -r 9.10.2b -G 4 -B x13 CONDALIAS=Aug2000 NEVENTS=1000 DEC-  
FILE=B0B0bar_e+e-.dec 100250-100259
```

Making each step separately, without database, using name value pairs:

```
spbuild --nodb -t simu RELEASE=9.10.2b DECFILE='ccbar.dec' GENERATE='continuum'  
NEVENTS=1000 DESC='e+e- -> ccbar' QAREF=01040 071240-071289
```

```
spbuild --nodb -t mixr -B x1 RELEASE=9.10.2b PROD_FD_BOOT=$OO_FD_BOOT 071240-071289
```

```
spbuild --nodb -t reco -B x1 RELEASE=9.10.2b PROD_FD_BOOT=$OO_FD_BOOT 071240-071289
```

Making all three steps with a properly filled database:

```
spbuild --user <your name> 100200-100250
```

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1), spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

spcheck – get information on runs in the ALLRUNS directory from files

SYNOPSIS:

spcheck [options] [run numbers]

DESCRIPTION:

spcheck parses information from the files for each run, and returns a nicely formatted table on the status of each run. The default behavior is create a one line status of each run asked, for - where each procspec built is listed as built, submit, run, done, good, fail, hang, killed. The meaning of the status output is:

built - the run directory and procspec directories are built; **submit** - the procspec has been submitted; **run** - the procspec is running; **done** - the procspec is done, but it is unknown if the run was good; **good** - the run was good, the number of events written equals the event requested; **fail** - the run failed, the number of events written is less than requested; **hang** - the run was running, but the log file has not been written to in more than 30 mins.; **killed** - the run has been killed with spjobs --kill, or the run was running and the log files has been written to in more than 8 hours, or the run was submitted but hasn't run yet it the past 2 days. **merging** - the run is being merged ; **merged** - the run has been merged but not yet exported ; **finished** - the run has completed but not yet checked(?) ;

If the run directory has not been built, or there is not status file for this run, spcheck will note this.

spcheck can also check on the amount of events written to a collection. The output collection for each procspec is determined and the number of events for this collection is found. spcheck can then check that this number of events is the same as the number of events requested, and mark a run as good, or failed. This function is handled in the --coll option.

The default behavior is to check for all built procspecs in the status file. But you can also specify the information for a certain procspec on the command line, much like spbuild and spsub. To look for the status of a certain procspec over many runs use the procspec, Geant, version, mixrver, recover, release, mixrrel, and recorel options.

Very Useful (tm) options are --status and --need. --status will give you a one line statement of where the run is in production, from being built to being done, and will add how far along a run is if it is running. --need will tell what is needed in the system, skipping over the good runs, it will point out the state of runs which are not done yet, and where they are.

Run numbers are in the same format as for spbuild and spsub, see those man pages for more information. If no run numbers are needed, spcheck will search through all runs built in the allruns directory.

By default spcheck will format the output for color, to turn this off use the --nocolor option.

OPTIONS:**--abandoned**

Only display information about procspecs that have been abandoned.

--analysis

Print out a set of stats analyzing the computers used in the run range

--built Only display information about procspecs which have only been built.

- c, or, --coll**
Look at the number events written to the output collection and compare this to the number events requested for the run and procspec.
- debug**
Print out debugging information to the screen
- decfile**
Include the decfile name in the status line if the run is finished.
- done** Only display information about procspecs which are done.
- fail** Only display information about procspecs which have failed. Output includes number of events produced for logfile, and computer the job failed on.
- fast** Do checks on status of runs with the minimum amount of file parsing. This option will skip the percentage done, decfile used, time taken in running, and the events per second calculation. This will speed up checking on large numbers of runs.
- finished**
If combined with '--fast' this option will create a quick list of finished runs.
- G (3|4), or, --Geant (3|4)**
Specify the version of geant used in the simu stage, either 3 or 4. If this option is used it must be either 3 or 4.
- good** Only display information about procspecs which are good.
- hang** Only display information about procspecs which are hanging, where the status says it is running, but the files haven't changes between 30mins and 4 hours, after 4 hours it is marked killed.
- h, or, --help**
Print out a sort help message about the options.
- html** Format the output as a web page. Useful for monitoring, setup a process which will do a 'spcheck --html --status >> spcheck.html' every once in a while. sprite will do this automatically for you.
- kill** Only display information on procspec which have been marked killed, either from killing the job with spjobs, or if the status has not changed on a run in 4 hours.
- l, or, --long**
Print out a longer output than the one line statement. This option is used to see more information from the status file. Not suggested for a list of many run numbers, but useful to get more info about the status of one or two runs.
- merged**
Only display information on procspec which have been merged.
- merging**
Only display information on procspec which are in the merging state.
- M #, or, --mixrver #**
Specify the version of the mixr stage in the run.
- mixrrel <release>**
Specify the release used in the mixr stage, see spbuild for more on format.
- need** Display the procspecs which need to be done next. spcheck will skip over the good procspec and display one line about what procspec needs to be done next in production for each run number. Will also display the needed command which should be used to keep production running. (Not fully operational yet, but close...)
- nocolor**
Do not use the color formatting when printing to the screen.

- nopercent**
Check on the status of the run, but do not parse the log file to check on the percent done for the run. Used in speeding up checking on status of runs.
- o <filename>**, or, **--output <filename>**
Write the needed ProdTools commands into this file. Combined with the --need option.
- prospec <prospec>**
Specify the full prospec to check, see spbuild for format.
- R #**, or, **--recover #**
Specify the version of the reco stage in the run.
- recorel <release>**
Specify the release used in the reco stage in the run.
- r <release>**, or, **--release <release>**
Specify the release used in each stage of the run.
- run** Only display information about prospecs which are running.
- set <option>**
Set the run control for a specific run, used to fine tune sprite control over runs. You can set a run to be ignored by sprite by using 'spcheck --set abandon' so sprite will ignore the run in its next cycle. To turn sprite control back on for an abandoned run you can use 'spcheck --set unset'. You can specify a run to be tried double the failure before abandon by using 'spcheck --set recover'. This is used for runs which are managed by sprite, abandoned, but now the system is fixed, and you would like sprite to try again.
- submit**
Only display information about prospecs which are submitted.
- summary**
After checking the runs, spcheck will include a one line summary of the number of jobs running and submitted in the list.
- status**
Display a one line status of the run showing where it is in the production cycle. The option will also add extra information to the submit, run, and hang outputs. Where the last modification time for the log file is displayed and for running the percent done for the run is shown.
- t (simu|mixr|reco)**, or, **--type (simu|mixr|reco)**
Specify the stage of the run you would like to check
- T #**, or, **--tail #**
Tail the log file for the prospec. The number of lines printed is given in the option.
- timeout #**
Set the time in minutes after which a hanging job is marked as killed. The default is 480 minutes (8 hours).
- q**, or, **--quiet**
Don't print output to the screen. Useful with the --summary option, to produce a one line summary output only.
- queue**
Use a different queue when generating the submit command.
- u**, or, **--update**
Update the collection information into the status file. Prints a line into the end of the status file about the number of events for that prospec.
- V #**, or, **--version #**
The version of each stage used in the run, also the version of the simu stage.

EXAMPLES:

List the status of all runs and all procspecs built for those runs

```
spcheck
```

List the status of running jobs and how far along they are

```
spcheck --status --run
```

Show what is needed for production and write ProdTools commands to a file

```
spcheck --need -o commands.sh
```

Tail the log file for a certain job

```
spcheck -t simu -T 50 550650
```

List the status of each procspec built for a set of runs

```
spcheck 410100-410199
```

List out the full status of the mixr stage for one run

```
spcheck -t mixr -l 410150
```

Check on the number of events in a collection for the reco stage of a run

```
spcheck -t reco -c 410150
```

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1), spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

spclean – clean out the run directories after running or failures.

SYNOPSIS:

spclean [options] [run numbers]

DESCRIPTION:

This utility cleans out un-needed files from the run directories after running or after a failure. In the case of running, the directory has only files needed for archive purposes. After a failure, the directory will now be ready to start the run again. The utility also interacts with the database to reset the values for the run, and there is an option to delete a record from the database. This option will probably go away soon.

There are three levels of cleaning:

Clear: Just remove XDR files.

Kill: clean up everything for a restart, deletes everything except for the config files.

Wipe: delete all files in directory and the database entries

The format for the run numbers is the same as for spbuild and spsub, see those man pages for details.

OPTIONS:

-B <alias>, or, **--background** <alias>

Set the background alias used in the mixr stage.

-c, or, **--clear**

Set the clear level of cleaning, just removes XDR files

-G <3|4>, or, **--Geant** <3|4>

Choose the version of geant, 3 for bbsim usage, 4 for bogus usage.

-h, or, **--help**

Print out a short help message.

--keepdb

Does not delete the record from the database

-k, or, **--kill**

Set the kill level of cleaning. Clean up everything for a restart, deletes everything except for the config files.

--mixrrel <release>

The release used for the mixr stage, used if different than the default release.

-M #, or, **--mixrver** #

The version of the mixr stage.

--nodb Do not connect to the database, this option will go away soon.

-n, or, **--norun**

I don't know what this does.

--recorel <release>

The release used in the reco stage, used if different than the default release.

-R #, or, **--recover** #

The version of the reco stage.

- r** <release>, or, **--release** <release>
The default release used for this run.
- T** <F|P|R>, or, **--trigger** <F|P|R>
The trigger used for the mixr stage.
- t** <simu|mixr|reco>, or, **--type** <simu|mixr|reco>
The stage of the run that you want to clean.
- u** <username>, or, **--user** <username>
The user name of the person who is doing the cleaning.
- V** #, or, **--version** #
The default version of the stages of the run.
- w**, or, **--wipe**
Set the wipe level of cleaning, delete all files in the directory, and the database entries. This level needs to be re-thought

EXAMPLES:

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1), spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

spcontrol – start and stop the run control daemon for simu production

SYNOPSIS:

spcontrol [options] (command)

DESCRIPTION:

This will start and stop the sprite run control daemon for you. Mostly it goes and changes the parameter 'runcontrol' in the sprite_rc file. There is one command line option and this is followed by the command. The command line entries are not options, but inputs, so there are no dashes ('-') used.

OPTIONS:**--debug**

Print out debug messages to the screen.

-h, or, --help

Print out a help message.

-c <rcfile>, or, --conf <rcfile>

Name of the sprite_rc file. If this is not used, the default file will get used from \$PRODTOOLS/site/\$BFSITE/sprite_rc. This option has a few features. If you specify a directory for the option, sprite will use <DIR>/sprite_rc as the rc file and this can be used to put rc files in another default directory. Or if you specify a different file name, without the full path, sprite will use \$PRODTOOLS/site/\$BFARCH/<FILE> as the rc file, this can be used to have different names for rc files for different runs used, or to have two sprites running with different control files. Or the full path name to the rc file can be used here, to have the rc file called from anywhere. Careful about using this option, since it can result in confusing use of sprite. Useful if your site subdirectory is in afs.

COMMANDS:**addruns**

Add an additional range of runs for sprite to manage. You will be prompted for a start and end run number.

changeruns

Change range run for sprite to manage. You will be prompted for a start and end run number.

run Change runcontrol to running.

savestate

Save the state of sprite. On the next call to 'spcontrol start' it will start with this state. This way you can restart sprite with 'suspend' or 'running', instead of the default to start 'running'.

status Returns the status of sprite.

start This will start sprite. If the runcontrol state was stop, this will change runcontrol to 'running'. The output of sprite is re-directed to a file \$ALLRUNS/sprite_out.log. Usually nothing goes in the file, but some errors might appear here in rare problems.

stop Change runcontrol to stop. sprite will exit on next cycle, which might take upto 6 mins.

suspend

Change runcontrol to suspend. sprite will continue to cycle and monitor but it will no longer build or submit runs.

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1),
spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

spexport – Export merged collections to SLAC.

SYNOPSIS:

spexport [options]

DESCRIPTION:

This command scans the merge directory \$RUNDIR/merge for successfully merged runs and exports them to SLAC. When finished the database at SLAC is updated.

OPTIONS:

--debug x#

Print out lots of debug messages.

-h, or, --help

Print out a brief help message with short descriptions of the options.

--nodelete

Do not delete collections after successful transfer.

--noexport

Do not export collections. Useful for testing but you will get fake "transfer error" messages.

-n, or, --number

The number of merge collections to export. Default is 20.

-i, or, --size

Maximum size to export (default=13 GB).

-u <user name>, or, --user <user name>

The user name of the person running this command. If not set it, defaults to the env. variable \$USER. If a shared account is used for production this variable must be set.

-v <version>, or, --verbose

Print out status messages of progress making merged collections

EXAMPLES:**Export some merged collections to SLAC**

spexport --verbose --user douglas

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1), spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

spjobs – get information on simulation production runs in the batch system

SYNOPSIS:

spjobs [options] [run numbers]

DESCRIPTION:

spjobs gets information from the batch system on currently running jobs. It then parses this information into a nicely formatted table to the screen. The table has information on run number, batch job id, stage running, current status (pending or running), where it was submitted from, and computer it is running on, and other info.

This utility should be where all batch information for simulation production is parsed and developed. Also interactions with the batch system for management of simulation jobs should be handled by this utility. Such as: You can also kill jobs for the batch system using spjobs. This is the preferred method of killing jobs, since a record of the kill will be placed into the status.txt file for the run, and the spcheck tool can pickup on this event, marking a job as killed.

The format of the run numbers is the same as for spbuild, or spsub. If there are no run numbers given to the spjobs, it will list all runs in the batch system, with the same computer operating system you are running spjobs from. This is useful to manage jobs on linux or solaris separately if multiple computer systems are defined for a queue, like it is here at SLAC.

The LastMod field in the listing printed out by spjobs shows the modification time of logfile for running jobs. If the logfile is older than watchsec seconds (the default is 30 minutes) than the current time the job is considered to be hanging. This is denoted by changing the status of the job from RUN to HANG. Of course, this information should be taken with care -- sometimes a job might be just very silent.

The time is listed as number of minutes since last update of the logfile is the jobs is running, or as the date of last modification if the run is done or exited.

A watch mode allows for spjobs to monitor the state of jobs in the batch system by printing out a listing every watchsec seconds.

OPTIONS:**-a, or, --all**

List batch information from all computer systems defined on the batch queue. Useful for installations which have more than one computer system defined for a queue. Like at SLAC where the queue for the jobs runs on solaris and linux.

-c, or, --computer

List information about the job running on a certain computer. Partial strings are accepted. To list runs on say bronco's in the 600's you would use '-c bronco6'.

--debug

Print out all batch information, and other debug info to the screen

-D, or, --done

List information about jobs recently done.

-E, or, --exit

List information about jobs exiting the batch system.

-h, or, --help

Print out a short help message.

-H, or, --history

List information about all recently done jobs, along with jobs in the system. Useful for finding info on jobs which are done.

--hanging

Only list hanging jobs. Does not work if --nolastmod option is in effect.

--kill

Kill all jobs that spjobs lists. Useful for killing jobs in the batch system by run number, or by computer type, or all jobs.

--nolastmod

Disable the lookup and print out of the modification time of logfiles.

-P, or, --pending

Only list pending jobs.

-Q, or, --quiet

Don't write out info about the run.

-R, or, --running

Only list running jobs.

-s, or, --summary

Print out a summary of how many jobs of what type are in the queue.

-t <simu|mixr|reco>, or, --type <simu|mixr|reco>

List information about the type of job, simu, mixr, or reco.

-T #, or, --tail #

Print out the last # lines of the logfile for the run in the queue. The number of lines to print must be specified.

--watch [#]

Enter watch mode, an optional parameter watchsec can be specified. In this mode spjobs prints out the listing of the jobs in the system every watchsec seconds. The parameter watchsec is also used in determination if the job is hanging. Default is 30 minutes.

BATCH SYSTEM SUPPORT:

spjobs is able to support multiple batch systems through modular utilities. At this time there is batch system support for LSF, PBS and Codine batch systems, which have been developed at SLAC, U. Dallas, and Caspur. There has been a little confusion about how to use these batch system modules, so I will try to briefly explain it here.

To use one of these modules for your site you need to put a few lines in your batchUtils.pl file in the site directory. The first line you need is to choose which batch modules you will use, you can do this with a require statement. at slac this line is:

```
require "LSFUtils.pl";
```

Since we use the LSF system, you should replace the file name with "PBSUtils.pl" or "CodineUtils.pl" depending on your site.

The next line you need to add is a filter for the machines names your jobs will run on. This is useful for complicated batch systems where spjobs needs to filter on the batch information to only list certain files. Also spjobs will only by default display the machines with the same system as the computer running spjobs. This is needed here at slac, where jobs on solaris machines are managed separately from jobs on linux machines. This machine name filtering is done with the %SYSTEMS hash, for slac it is:

```
%SYSTEMS=("linux" => "(noric|barb)", "solaris" => "(tersk|shire|bronco)");
```

The names in the parenthesis are partial matches to machines names. You will need to replace these with the names of machines used in your system. The matches are logical or of the strings within the '|' characters, so any number of partial machine names can be specified here. If you are only using one system then will not need the other part of this line, although it doesn't hurt anything to keep it in. spjobs will only list runs which are on machines whose names match strings in this hash. So, if you are getting problems where spjobs states no runs are in the system, and you know they are there, then check that all machine names in your system will match something on this line.

EXAMPLES:

List all of your jobs in the system

spjobs

Print a summary of all jobs from all computers without listing each job info

spjobs -s -Q -a

Kill all running jobs

spjobs -R --kill

Kill jobs in the system by run number

spjobs --kill 450000-450099

Report about hanging jobs each 30 minutes

spjobs --hanging --watch

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1), spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

spmerge – Create and manage jobs for merging run collections.

SYNOPSIS:

spmerge [options]

DESCRIPTION:

This command scans the run directories for runs that have completed and merges the output root files together when there are enough files of the right type (determined by the decay file.) The merged file will be up to 1.5 Gbytes. The command assumes something has gone wrong with the merge if the job takes more than 4 hours to run or has not started running for 24 hours. The successfully merged jobs are put in \$MERGEDIR (default = \$RUNDIR/merge).

OPTIONS:

-C, or, --checknum

The number of merge jobs to check, usually a limit for testing. Default is 20.

--debug x#

Print out lots of debug messages.

-h, or, --help

Print out a brief help message with short descriptions of the options.

--ignore

Forces a merge even if merge has failed 3 times previously.

--norun

Do not submit merge jobs, just echo the batch commands to the screen.

-n, or, --number

The number of merge jobs to setup, usually a limit for testing. Default is 10.

-o, or, --options

Batch queue options to be passed to submit command.

-q, or, --queue

Choose the queue to submit the merge job to, the default is the same as spsub.

-u <user name>, or, --user <user name>

The user name of the person running this command. If not set it, defaults to the env. variable \$USER. If a shared account is used for production this variable must be set.

-v <version>, or, --verbose

Print out status messages of progress making merged collections

EXAMPLES:

Create merge jobs on all completed runs:

spmerge --user <your name>

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1), spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

sprite – run control daemon for BaBar simulation production

SYNOPSIS:

sprite

DESCRIPTION:

sprite will control job submission for simulation production. This is meant to be run in the background on a computer, and control job submission to get runs done in the most stable manner. The hope is for less human interaction in simulation production. There are no command line options for sprite, but its control is defined in a file called `sprite_rc`. This file lives in your site directory in your ProdTools installation.

sprite is designed to be controlled by the `spcontrol` command rather than directly submitted. There is a man page for this util, and its use. It will start, stop, and suspend sprite for you.

At first time start up sprite will create a run control file for you, with default settings, if one does not exist. You will need to edit this file to get started, look in your site directory for the newly created `sprite_rc`. The lines are commented and easy to understand (hopefully...) change the settings as needed for your environment, mostly this will have to be the `maxJobRunning` parameter. You must at least change the `runcontrol` from 'stop' to 'running'. You can then start sprite.

If your site directory is in afs, this can cause problems as the token can expire and the sprite will no longer be able to write to `sprite_rc`. sprite will check for the expiration of the afs token if `AfsTokenWatch` is set and will stop 2 hours before expiration. Alternatively, you can specify a different `sprite_rc` when you start `spcontrol` (see below.)

To start sprite, you just have to type '`spcontrol start`' at the command line. It will launch sprite as a background process on that machine. You will have to make sure that ProdTools utils will work on that machine. Mostly if `spbuild` and `spsub` works on the machine, sprite will work on the machine.

sprite will check on runs, as defined by the '`runrange`' parameter, and do what is needed. It is a stateless process, which will cycle checking on runs and doing what is needed, which means no information is carried over from one cycle to the next. This means the sprite can be stopped or started at any point and it will continue on where it left off.

sprite will not interfere with on going runs or production. You can continue to submit runs if you wish, sprite will see that they are running, and ignore them. sprite will just check on runs and do what is needed, and use the available resources. If you have a hard limit of number of jobs at your site which is less than available resources, be careful if you have sprite submit jobs along with submitting jobs by hand. sprite will only limit the number of jobs it controls, if you submit more by hand you may go above the `maxJobRunning` limit set in sprite. Just a warning.

If there are problems and failures in a production run, sprite will attempt to build a new version of the job and submit. Because of this, the shell environment must be setup for building runs. Proper setting of `srtpath`, `setboot`, and `$OO_FD_BOOT/$CDB_ROO_BOOT` will be needed by sprite.

Because of the need on the environment to be setup before running, sprite can only handle one type of runs at a time, one setting of `srtpath` and `$OO_FD_BOOT/$CDB_ROO_BOOT`. If you have a set of runs with a mixture of releases, this will not work. This hopefully will change in future versions of ProdTools.

Also at slac there is a script setup which will restart sprite every 24hours, and setup the env. (`srtpath` and `setboot` etc...) need for sprite to run. This script is `restart_sprite`, and can be run from the command line as soon as you log in as `bbrprod` at slac. If you need such a function at your site, you should take a look at this script.

QUICK HOW-TO

To get started first time:

0. If your site sub-directory is on afs, make sure you have a token.
1. If spsub, spbuild, and spcheck work for you, type 'spcontrol start'.
2. A default sprite_rc file will be made in your site sub-directory in your ProdTools installation, open this file in an editor.
3. Change 'runcontrol' to 'running', change 'maxJobRunning' to the number of cpus you have for simu. production, change the 'runrange' to limit the range of runs sprite will control (if needed), set 'cycle' to the production cycle (e.g. SP10). Save the file, and exit the editor.
4. type 'spcontrol start' again, it should now be controlling your simu. production jobs.

Command Line OPTIONS

-c <rc file>, or, **--conf** <rcfile>

Name of the sprite_rc file. If this is not used, the default file will get used from \$PRODTTOOLS/site/\$BFSITE/sprite_rc. This option has a few features. If you specify a directory for the option, sprite will use <DIR>/sprite_rc as the rc file and this can be used to put rc files in another default directory. Or if you specify a different file name, without the full path, sprite will use \$PRODTTOOLS/site/\$BFARCH/<FILE> as the rc file, this can be used to have different names for rc files for different used, or to have two sprites running with different control files. Or the full path name to the rc file can be used here, to have the rc file called from anywhere. Careful about using this option, since it can result confusing use of sprite. Useful if your site subdirectory is in afs.

--debug

Print out lots of debug messages on each cycle of sprite.

RUN CONTROL FILE

The control of sprite is through the run control file. If you do not have one, sprite will create a default run control file for you. The file name is sprite_rc and is installed in your site sub-directory in the ProdTools install. The file contains comment lines, which start with '#' and can contain any string after this character and are ignored. The file also contains parameter lines in the form of "<name>=<value>" pairs. Where <name> is the name of the parameter to be set, and <value> is the value of the parameter in the form of an arb. string. The '=' character is necessary, and there are no spaces between the name and value, and the '=' character. Also the name must come at the start of a line, there can be no space before the <name> string. The name is case sensitive.

NEEDED PARAMETERS

runcontrol

This controls the running state of sprite, there are three values for this parameter: 'stop', 'running', 'suspend'. 'stop' will stop sprite and the background process will stop and exit after a few minutes. runs, check on status, and monitor the system. 'suspend' sprite will do everything except submit or build runs. The default is stop when sprite_rc is first created.

runrange

The range of runs which sprite will control. To control all runs on disk this can be set to '100000-39999999'. This is useful if you have a set of runs for sprite to get done for production, and a set of runs which you need to do by hand for testing. Or if there is a set of runs on disk for some reason you are keeping, but will never get done. This run range can be much larger than the set of runs actually built on disk, where if the run is not built yet, sprite will ignore that it is not

there. The intended use of sprite is to just run in the background, and as you build runs it will start to submit them and get them done. This run range is not intended to be like the input to spsub where you submit a set of runs already built on disk.

The run range can contain multiple non-contiguous block of runs. For example: '345600-345700 656500-656700 678000 678005' would work, where you put a space between the blocks of runs. sprite will do the runs in numerical order only. This might produce a surprise if you use a range of: '675000-675200 652000-652300' which will work, but the second block will be done first. The default is all runs.

cycle The production cycle to be managed by sprite e.g. SP10. This allows different production cycles to be controlled from sprite.

maxSubmitPerCycle

The max number of jobs that sprite will submit at once. The actual number of jobs that sprite will submit is an adjusting parameter depending on system resources, number of jobs running, maxJobRunning parameter, and jobs submitted. This number should not be too big, there are stability issues in using the databases and too many jobs starting at once. Somewhere between 6 and 10 is a good number for this parameter. Default is 6.

maxJobRunning

The maximum number of jobs which sprite should run. Set this parameter to the number of jobs which should be running at your site, if this is less than system resources. Like if you are sharing a batch queue with other jobs. If you want to be limited to the system resources set this parameter to the number of cpu's that you have. This number can be more than the number of possible jobs if needed, and sprite will limit production by resources. But because of scaling number of jobs submitted at one time, for stability this number should be with 10-15% of the actual number of cpu's.

submitWait

The amount of time in seconds to wait before submitting the next set of jobs. Job start up in simulation production takes some time, and for stability issues it is good to wait some time before starting more jobs, to allow for objectivity collection creation. A good setting for this is between 75 and 120 secs. Default is 120 seconds.

failuresBeforeAbandon

The number of times sprite will re-try a run before abandoning it. If a job fails too many times, sprite will mark the run as abandoned and ignore it in future cycles. Good setting for this number is between 3 and 5. Setting this to 1 will mean that sprite will not re-build a run to try and fix it. This is not recommended since this will require more work from you to see if the runs can be fixed by hand, plus it kills a lot of what sprite was built to do. Default is 5.

OPTIONAL PARAMETERS

AfsTokenWatch

If set to yes, sprite will exit 2 hours before the afs token is due to expire. This is useful if your site directory where sprite_rc is stored is under AFS. Alternatively move sprite_rc to a non-AFS area and use the

AllRunsDir

This is full path name of the directory containing all the working directories for the jobs. This sets the ALLRUNS environment variable at sprite startup.

archive

This option can be set to 'local' or 'remote'. If this option is set sprite will try to archive the runs it manages once they have finished. This option is just getting started and 'remote' doesn't do anything yet.

AutoSweepDir

The dir. sprite will look in for semaphore files for external control of sprite and jobs. Used by Tofogh's AutoSweep scripts to control production at SLAC. This is mostly useful for controlling the jobs during sweeping of databases, so production can be brought down, and come back in an automatic way at all hours of the day and night at SLAC. Might be useful for remote sites in automatic control of jobs during a run cycle for export to SLAC, I don't know. Might be just as useful to create scripts which control sprite through sicontrol, it is up to others if they might need these features or not.

diskwatch

This is the directory that sprite will monitor to ensure there is enough room before submitting jobs to prevent the disk becoming full. sprite uses 'df' on this directory so it can be any directory on the disk that you want to monitor. The minimum amount of free space is set by minAvailSpace.

exportWait

The amount of time in hours to wait before exporting another set of merged collections to SLAC. Default is 3 hours.

htmdir

A directory which will be served by a web server, so that everyone in the world can monitor your site. These are a copy of the files produced into the monitor directory, but there is no archive. If this parameter is not set, there will be no copy of the monitor pages made. This is not needed for production, and the choice to use this option for your site is up to you.

queue The batch queue to be used for simulation jobs.

mergequeue

The batch queue to be used for merged jobs. useful if you need a different queue with higher priority than the simulation jobs.

mergeWait

The amount of time in hours to wait before submitting the next set of merge jobs. Default is 6 hours.

minAvailSpace

The minimum amount of free space that must be available for sprite to continue to submit jobs. See diskwatch.

monitor

This can be set to local or all, the default is local if not used. This controls the ability to monitor all simulation production sites along with your own. Other site dir. will be created and all simulation sites can be monitored at each local site. This feature is only getting started, and remote monitoring of all sites only works at slac for now. Soon can work for everyone, and this feature can be moved up in priority if desired by others.

monitordir

The directory to put the monitoring pages and archive of monitor information. The default is \$ALLRUNS/monitor, and this will fine for most people. The archive will use about 15MB of space, just so you know. One use for this, although questionable, is if you have two batch queues and two version of sprite running, there can be two monitoring directories.

monplot

Tell sprite to use the sponplot util to create plots of the last weeks production. These plots can then be viewed in the monitoring web pages. This requires the install of the pgplot libraries and the PGPLOT perl modules at your site. To use this set to 'yes' or '1', to get rid of the option, comment it out, or set to 'no' or '0'.

OO_FD_BOOT

The OO_FD_BOOT env. variable can be set here in the sprite config, and sprite will set this env. variable at startup. Obsolete in release 24 (SP10 production). A similar facility for CDB_ROO_BOOT does not exist.

producer

The name of the person to be associated with this production. Useful if you are using a shared or generic account. Default is sprite.

spjobs This can be set to yes or no, the default is no. If this is yes then sprite will use spjobs to determine if a run is still in the batch system if the run has been pending in the submit state for too long. If the run has left the batch system, the sprite will mark it as killed, and build and submit a new version. Warning: your site must have spjobs working, where if a job is in the system, spjobs will have a listing (run number and state), if not long pending jobs will get marked killed on you.

spritename

An option name to give sprite, if you have feelings for the little code. This is used to be able to run more than one sprite at a site, if needed for separate production farms or separate run directories, or if you think you need to. There should only be one sprite running under any name, and different sprites should not control the same runs (although this probably won't matter, but slight timing difference could confuse separate sprites.)

useDebugger

To control running of the jobs in the debugger or not. This will set the environment variable \$SPUSEDEBUGGER for the sprite session, so all runs controlled by sprite will be run in the debugger. To use this set to 'yes' or '1', to get rid of the option, comment it out, or set to 'no' or '0'. It will be checked on each cycle of sprite, so you can turn the debugger use on and off without restarting sprite.

useLocal

If set to 'yes' or '1', the batch job will write the output root file to its local temporary area and then copy it over to the run directory when the job ends. This can reduce the load on the network. The temporary location can be defined by the environment variable SPLOCALDIR in local-simul-setup; if SPLOCALDIR is not defined then /tmp is used. Default is 'no' or '0'.

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1), spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

spsiteplot – Create a plot of number of events by location, either cumulative or by day

SYNOPSIS:

spsiteplot

DESCRIPTION:

This command runs a script that gets number of events by day, type and location from the prod_job table of the e-logbook. It then uses PGPLOT to plot this data into a GIF, with a separate line for each location. The GIF is currently written into the ProdTools directory, but this will be made an option.

On running the script, the user is prompted for start and end dates and for which type (or all types) of events should be included, then for which locations should be included, what the output file should be, and whether the plot should be cumulative or day-by-day. The prompting should be self-explanatory and includes rudimentary error-checking for dates.

Running PGPLOT requires the existence of certain OS-dependent files, and at the moment, this means that the script will only successfully produce a plot if run on a Solaris machine.

NAME:

spsub – submit a simulation run to the batch queues

SYNOPSIS:

spsub [options] [run numbers]

DESCRIPTION:

After the run directories have been built with `spbuild`, they can be submitted to the batch queue with this command. The command can be used to submit all stages of the simulation job at once, or each stage individually. Control of the program can come from options on the command line, or from job records stored in a database.

The command will perform checks on the files in the run directories before submission, and checks on properly formed procspecs for the jobs. To do this certain options need to be on the command line, or in the database. Further documentation of the proper procspec can found in the `spbuild` man page.

The run numbers can be a single run or specified as a range of numbers, with a dash separating the numbers, such as 250101-250200, to build 100 runs.

Also more than one range of numbers can be placed on the line, such as, "250101 250105 250151-250200 250205" to build a set of 53 runs. This option works, but use it with care, you might get run behaviors you do not expect.

OPTIONS:

-B x#, or, **--background x#** Sets the background alias.

This corresponds to an Objectivity collection defined in the file `bkgcoll.db` within `ProdTools`. Needed to check properly formed procspec for the `mixr` step.

--debug

Print out debug messages to the screen.

-d #, or, **--delay #**

Delay submission of the run. This is used to put a delay between submissions of a set of runs, where a large fast submission of run may freeze up a system because of database access at startup. The option is an integer number of seconds.

-D <tag>, or, **--dftag <tag>**

Specify the tag of the `ProdDecayFiles` package to use in running.

-G <3|4>, or, **--Geant <3|4>**

Specify the version of `geant` being used for the run, 3 or 4. This chooses the use of `bbsim` or `bogus` for the run. The executables which get run are chosen during the building of the directories this option is needed here only to check on the proper procspec used for the run. You can not build the directories to use one programs, and then decide to use a different program at submission time. If not set it defaults to 3.

-h, or, **--help**

Print out a help message.

-m <minutes>, or, **--minutes <minutes>**

Limit the number of minutes the job can run.

-M <version>, or, **--mixrver <version>**

The version of the `mixr` stage you are submitting to be run. Needed for multiple versions of this stage used in one run.

- mixrrel** <release>
Used to specify the release used in the mixr stage, must match the release used in building the mixr stage. Can not be used to specify a different release to be used at submission time.
- mixrspec** <procspec>
Used to specify the procspec used in the mixr stage.
- N** <simu|mixr|reco>, or, **--next** <simu|mixr|reco>
Specify the next stage to be run after the one submitted. If you wish to submit simu and then mixr only, you can use a combo of -t simu -N mixr. If you wish to submit all three stages at once you would use -t simu -N reco, where the mixr stage would be filled in. All three stages submitted is the default when --type and --next options are not used. Not needed with Moose.
- nodb** Not used any more. The option is still in the system so it does not generate an error if used, but it doesn't do anything anymore. The old description is: Do not use information from the database. Needed for personal use and in sites without a database. All needed information for submission must be specified on the command line.
- nodebugger**
Do not run the job in the debugger. If the job crashes a core file will be produced.
- n**, or, **--norun**
Do not submit the run, just echo command to the screen.
- o** <options>, or, **--opts** <options>
Specify other options for the batch submit command.
- procspec** <procspec>
Specify the procspec used for the submitting job.
- p**, or, **--prestage**
Prestage in xdr files from the storage system before submission. This option does not actually work.
- q** <queue>, or, **--queue** <queue>
Specify the queue used for the job. If not set it defaults to amslong queue for production running, and bfobjy queue for personal use.
- R** <version>, or, **--recover** <version>
The version of the reco stage you are submitting to be run. Needed for multiple versions of this stage used in one run.
- recorel** <release>
Use to specify the release used for the reco step in the run. Must match the release used when the run directories were built.
- r** <release>, or, **--release** <release>
The release used for the run, must match the release used in building the run. Used in check the procspecs and run directories used. If not set it defaults to the value in the database.
- top** Put the job at the top of the batch queue, partial supported at certain sites.
- T** <F|P|R>, or, **--trigger** <F|P|R>
The trigger used when building the mixr stage of the run. If not set it defaults to 'F'.
- t** <simu|mixr|reco>, or, **--type** <simu|mixr|reco>
The stage of the run to be submitted. Used to only submit one stage of the run at a time. If not set spsub will submit all stages at one, in one job. This 'job chaining' is tricky when multiple versions of stages are used. Make sure you have properly set the --mixrver and --recover options.
- usedebugger**
Run the job in the debugger. The jobs script will choose dbx for solaris machines, gdb for linux machines, and ladebug for osf machines; before using this feature make sure your site has these debuggers setup. Also there have been cases where the env. variables for the job are not

maintained inside the debugger, depending on how your site is configured. Jobs will work at slac, but you have been warned.

-y, or, --yes

Submit the runs without prompting for each run.

DECAY FILES:

A small description of how decay files are handled: This is here because where the decay files are found, is decided at run time in ProdTools, not at build time or even submission time, but this is the closest place this should go. Plus I have described this repeatedly to enough people that it is obvious some sort of write up is needed.

The decay files used in ProdTools are kept in a package in CVS named ProdDecayFiles. This package gets updated and tagged from time to time, and is under the control of the production coordinator. Only tagged versions of the ProdDecayFiles are used in production, the tag is saved in the log files while running so people can tell what has been done later.

ProdTools will try to get the decay files from the default directory \$BFROOT/prod/packages/ProdDecayFiles/<tag>, where <tag> is a directory named after the tag for ProdDecayFiles. ProdTools searches through the tag directories in reverse time order (newest to oldest) looking for the requested dec file or tcl file. If it finds the requested file, then it uses that directory and records the tag, this is written out to the log file. If the requested file is not in this directory ProdTools will then search in \$BFROOT/prod/ProdDecayFiles for the file. If the file is not found there, then ProdTools will generate an error.

The tag for ProdDecayFiles can be chosen at submission time. The ProdTools will only look in this tag directory for the dec file, and record this tag in the log file.

To override all of this mechanism, you can test dec files with ProdTools for any directory, by using the \$DECAYFILES env variable. If this variable is set at submission time, ProdTools will only look for the dec file or tcl file in the directory stated by the \$DECAYFILES variable. Just set this variable to the directory which contains your test dec file, and ProdTools should use it at run time.

EXAMPLES:

Submitting all three steps, without database, specifying needed information on the command line:

```
spsub -q long -G 4 -r 9.10.1a -B x15 --nodb 100250-100259
```

Submitting each step separately:

```
spsub -t simu -r 7.11.2b 071240-071289
```

```
spsub -t mixr -B x1 -r 7.12.7a 071240-071289
```

```
spsub -t reco -B x1 -r 7.12.7a 071240-071289
```

Submitting all three steps with a properly filled database:

```
spsub 100200-100250
```

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1), spupdate(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu

NAME:

spupdate – update the database on the status of simulation production runs.

SYNOPSIS:

spupdate <options> [run numbers]

DESCRIPTION:

spupdate will parse the log file from the simulation production run for information, and update the database on the status of run, based on this information.

OPTIONS:

-B x#, or, **--background** x#

Sets the background alias. This corresponds to an Objectivity collection defined in the file bkg-coll.db within ProdTools. Needed to check properly formed procspec for the mixr step.

--debug

Print out debug messages to the screen.

--dofail

Allow runs to be marked failed. Default behavior will not mark runs as failed.

--finished

?

-G <3|4>, or, **--Geant** <3|4>

Specify the version of geant being used for the run, 3 or 4.

-h, or, **--help**

Print out a brief help message with short descriptions of the options.

-l <location> or, **--location** <location>

Specify the location to be updated, this defaults to \$BFSITE.

--markfail

Force a run to be marked failed.

-q, or, **--quiet**

Do not display output on procspecs to the screen.

-M <version>, or, **--mixrver** <version>

Only update mixr jobs with this version.

--mixrspec <procspec>

Only update mixr jobs with this procspec.

--mixrrel <release>

Only update mixr jobs with this release.

--nodb Do not attempt to connect to the DB at all.

-R <version>, or, **--recover** <version>

Only update reco jobs with this version.

-r <release>, or, **--release** <release>

Only update jobs with this release.

-T <F|R|P>, or, **--trigger** <F|R|P>

Only update jobs with this trigger.

-t <simu|mixr|reco>, or, **--type** <simu|mixr|reco>

Only update jobs with this type.

-u <user name>, or, **--user** <user name>

The user name of the person running this command. If not set, it defaults to the env. variable \$USER. If a shared account is used for production, this variable must be set.

-V <version>, or, **--version** <version>

Only update jobs with this version number

EXAMPLES:

Update production database with status of all runs

spupdate -u douglas

SEE ALSO:

spbuild(1), spcheck(1), spclean(1), spcontrol(1), spexport(1), spjobs(1), spmerge(1), sprite(1), spsub(1)

AUTHOR:

Douglas Smith - douglas@slac.stanford.edu