STAR scheduler - User Manual

1. Introduction

The STAR scheduler was developed to allow users to submit a job on several input files, residing on NFS or on the various local disks of the farm machines. The scheduler will divide the job into different processes that will be dispatched to different machines. In order for the scheduler to do that, the user has to follow some simple rules when writing his program, and to write an XML file describing the job he wants to be done.

In this manual we show the steps the user needs to take to use the scheduler

2. An example

2.1 Preparing the program

What will you basically submit to the scheduler is a command line. The program will have to receive instruction from the scheduler which input files it should use. For your convenience, there are two ways this is done.

2.1.1 Getting the input files from the filelist

The scheduler will generate a file list like this for each process:

```
/star/data21/reco/productionCentral/FullField/P02gc/2001/312/st_physics_2312011_raw_0017.MuDst.root
/star/data21/reco/productionCentral/FullField/P02gc/2001/312/st_physics_2312011_raw_0016.MuDst.root
/star/data21/reco/productionCentral/FullField/P02gc/2001/312/st_physics_2312011_raw_0015.MuDst.root
```

The environment variable $FILELIST will be set to the name of the file list. You can use this variable on the command line to retrieve the name as a parameter of your program. For example, submitting cat $FILELIST will display in the standard out all the input files that were assigned to that particular process.

Be careful that the output file is different for every process it might be dispatched, otherwise two processes that were dispatched to different machines and are running in the same moment will interfere with each other. You can, for example, have the name of the output files depend on the filelist name, or from the job ID, which you can retrieve from the environment variable $JOBID. In case you have an output file for each input file, and all your input files have different names, you can have the output file depend on the particular input. All the input files are assigned to only one process.

2.1.2 Getting the input files from environment variables

Another way to retrieve the file list is to look for the $INPUTFILECOUNT variable that tells you how many files where assigned to the process. Then you can look for the $INPUTFILE0...$INPUTFILEn for each input file name.

Also here you have to be careful for the output file names.

2.2 Job description

Once your program is ready, you can write the job description. A job description looks like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<job>
 <command>root4star -q -b StHbtDiHadron.C\(1000000,100,-1,\"\",\"$FILELIST\"\)</command>
 <stdout URL="file:/star/u/carcassi/scheduler/out/$JOBID.out" />
 <input URL="file:/star/data21/reco/productionCentral/FullField/P02gc/2001/312/st_physics_2312011_raw_0017.MuDst.root" />
 <input URL="file:/star/data21/reco/productionCentral/FullField/P02gc/2001/312/st_physics_2312011_raw_0016.MuDst.root" />
 <input URL="file:/star/data21/reco/productionCentral/FullField/P02gc/2001/312/st_physics_2312011_raw_0015.MuDst.root" />
  <SandBox>
   <Package>
    <File>file:./StHbtDiHadron.C</File>
   </Package>
 </SandBox>
</job>
```

The first line tells the parser what set of characters was used. Don't worry about it. If you are not used to XML, you can see that there are some tags that open and close. The main tag you see is <job> that closes at </job>. It means that everything inside that describes a job. You could have more than one, but you won't need to do that. Not having specified the working directory, the current directory will be used.

Inside the job, you can see another tag <command>. Inside that you will put the command line you will want to be executed. Then you have the <stdout> tag: it tells where the output of the process will be saved. After that you see many <input> tags. These don't have a </stdout> or </input> because they are directly closed at the end (notice the '/' in the ending '/>'). URL is an attribute of all the tags that will specify a file (input, stdout, stdin, stderr). For example, in the first input tag you are telling the scheduler that you require that file as an input for your job. After "file:" you have to put a full path to an NFS mounted file.

There are other options you can specify in your job description, and you can find the details later in this manual.

2.3 Job submission

Once you have prepared your XML file, you are ready. You just type:

```
star-submit jobDescription.xml
```

where jobDescription.xml is the name of the file that contains your job description. In this version, you have to specify the position of the star-submit command. In the final version, that won't be required.

The scheduler will create the scripts and the file lists for the processes. These will be created in your directory. Also this will change in the final version, and they will be put in a scratch directory.

3 Job description specification

Here is the full specification of what the XML for the job description can contain.

3.1 XML syntax

You don't need to know XML very well to understand how to use the scheduler. For the sake of being complete, here are a few concept that you will see in this description. It's not necessary to know them, but since XML is being more and more used, it might satisfy a general curiosity. Remember that HTML is kind of a subset of XML.

The basic notion in XML is the element (or entity), which is basically two tags with the same name: a start tag and an end tag. For example, <job>...</job> is an element .

An element can have some attributes, that better specify it. For example, in <job title="My title"> title is an attribute of the job element . The attributes are specified in the start tag.

An element generally contain some data, that can be other elements, or just some text, for example in:

```
<job> <command>echo test</command> </job>
```

The element job contains the element command that in turn contain the data "echo test". An element can also contain no data at all, and just have some attributes. For example, in <input URL="file:/star/u/carcassi/myfile"/> we have the element input, with a URL attribute, the ends right away. That is the start tag and the end tag are put together in one tag (notice the '/' in the end).

That is all the XML we are using in the scheduler.

3.2 The <Job> element

Here is an example of the <job> element with all the attributes:

```
<job simulateSubmission ="true" mail="true">
....
</job>
```

The <job> element can have the following attributes:

Attribute	Allowed values	Meaning	Default
datasetSplitting (optional)	"eventBased" or "fileBased"	By selecting "eventBased" with the *maxEvents* attribute single files can be split between multiple jobs where each job takes a different event range in the same file. This method only permits one file per job. The range of events each job should process is determined by the environment variables *${EVENTS_START}* and *${EVENTS_STOP}* which is defined in the environment of your job so it can be passed to the program processing the file. The user must provide the implementation in there analysis program. Warning: Multiple input files will point to the same dataset files, be careful not to over-count. Warning: This mode permits only one input file per job which can produce excessive numbers of jobs, please be careful not to flood or crash the batch system. Note: This option should not be selected for basic splitting by file using the *minEvents* and *maxEvents* attributes. Note: This feature is implemented on version 1.10.8 and up.	"fileBased"

splitBy (optional)	Any Catalog key	The attribute *splitBy* of the job tag will split the dataset (make a new job) whenever the catalog attribute it references changes. It can be used in conjunction with the *inputOrder* attribute to sort your input files. For example if you wanted to make sure all jobs only have files from the same run number, you could request:
		`<job inputOrder="runnumber" splitBy="runnumber">`
		You can even split finer by combining with the *maxFilesPerProcess* attribute for example:
		`<job inputOrder="runnumber" splitBy="runnumber" maxFilesPerProcess="30">`
		When working with this feature for the first time it is recommended that you set *simulateSubmission* to true, in order to check that the dataset splitting that SUMS has prepared is as you expected (you can look at the *.dataset file and report file), then use the session file to submit the jobs. If you are getting your dataset from other sources other than the catalog, SUMS will not be able to recover catalog dataset attributes, though some, such as the path and file name are still available, this will not fail, so please check the output before submitting. Also If you do not use the same *inputOrder* attribute as the *splitBy* attribute without ordering you may get too many splits and there for too many jobs.
		
		Note: This feature works on SUMS version 1.10.15 and up
		Note: For a more detailed overview of how SUMS prepares datasets please refer to this *flowchart*.
			none

simulateSubmission (optional)	"true" or "false"	Tells the scheduler whether to dispatch the actual jobs. If true, the file scripts are created, but they are not actually submitted. This is useful to check whether everything is functioning correctly.	false - by default the jobs are submitted
name (optional)	any string	Gives the job a name by which it can be identified in the underlying batch system.	none
mail (optional)	"true" or "false"	Tells the scheduler whether to allow the submission of a job that will returns it's output by mail. If not this is not set, or is not equal to true, the scheduler will fail if a stdout wasn't specified. This option is here to prevent a user to accidentally send himself all the outputs by mail.	false - by default no mail is allowed
nProcesses (optional)	any integer	Tells the scheduler how many processes should be dispatched for this request. This attribute can be used only if no input is present. When input is present, the number of processes are determined by the number of inputs and other job constraint.	Determined by the input
maxEvents (optional)	any integer	Tells the scheduler how many input events to assign to each process at maximum. This may be split across many jobs. For example a maxEvents value of 500Events may be satisfied by three files one with 50Events, one with 250Events and one with 175Events (note it does not insure exactly the maxEvents in each job). Note: The scheduler will not check the number of events inside each file. It can only take the number of events from the catalog or file list (if provided). If it can't parse the value it will assume one event and print a warning. Note: The *softLimits* attribute has an effect on how the *minEvents* and *maxEvents* algorithm behaves. Setting *softLimits* to *"false"* or not setting it at all will aggressively force the limits by dropping files that cannot be made to fit within the limits. Note: This attribute cannot be mixed with *minFilesPerProcess*, *maxFilesPerProcess*, or *filesPerHour*. Note: For a more detailed overview of how SUMS prepares datasets please refer to this *flowchart*. Note: Available in version 1.10.6 and up	Infinite
minEvents (optional)	any integer	Tells the scheduler how many input events to assign to each process at minimum. This may be split across many jobs. For example a *minEvents* value of 500Events may be satisfied by three files one with 78Events, one with 250Events and one with 310Events (note it does not insure exactly the *minEvents* in each job). Note: The scheduler will not check the number of events inside each file. It can only take the number of events from the catalog or file list (if provided). If it can't parse the value it will assume one event and print a warning. Note: The *softLimits* attribute has an effect on how the *minEvents* and *maxEvents* algorithm behaves. Setting *softLimits* to *"false"* or not setting it at all will aggressively force the limits by dropping files that cannot be made to fit within the limits. Note: This attribute cannot be mixed with *minFilesPerProcess*, *maxFilesPerProcess*, or *filesPerHour*. Note: For a more detailed overview of how SUMS prepares datasets please refer to this *flowchart*. Note: Available in version 1.10.6 and up	N/A
	"true" or	Tells the *minEvents* and *maxEvents* algorithm(s) if they are to be strict or allow exceptions as needed in order avoid dropping files from the dataset. The softLimits keyword controls the behavior of the *minEvents* and *maxEvents* algorithms. When *softLimits* is set to true: Use of the *minEvents* and *maxEvents* the min/max event splitter will use all the files it recovers, even if doing so violates the *minEvents* and *maxEvents* limits. For example if the *maxEvents* is set to 500 and you get 2,000 events in one file. The splitter will give that file its own job even though it is bigger than the max limit. When the *minEvents* and *maxEvents* limits are violated (cannot be met) a warning is printed showing the distribution, however scheduling will not stop.	

softLimits	"false"	When *softLimits* is set to false: This will force hard limits. For example if the *maxEvents* is set to 500 and you get 2,000 events in one file. The splitter will drop that file from the dataset. The min/max event splitters also cannot reorder the dataset. For example if you set *minEvents* to 100 and *maxEvents* to 200 and the current job has a total of 99 events in it and the next file to be added to the job has 150 events in it the file with 150 events in it will be dropped. Some final attempt at redistribution is made at the end of the last file, or at splits made by prior splitters, if it is incomplete to avoid dropping files needlessly. If it is desirable to get as large a dataset as possible and hard limits are required it is best to leave a large gap between the *minEvents* and *maxEvents*. Note: Available in version 1.10.6 and up. Before this version all limites are soft.	false
maxFilesPerProcess (optional)	any integer	tells the scheduler how many input files to assign to each process at maximum. This number should represent the number of files that your program, by design, is not allowed to have (e.g. after 150 files memory use has increased too much due to a memory leak). The actual number of files dispatched to the process is decided by the scheduler, which takes into account user requirements (i.e. minFiles, maxFiles and filesPerHour) and farm resources (i.e. length of the different queues). Note: For a more detailed overview of how SUMS prepares datasets please refer to this *flowchart*. Note: This attribute cannot be mixed with *minEvents*, *maxEvents*, or *eventsPerHour*.	Infinite
minFilesPerProcess (optional)	any integer	Tells the scheduler the minimum number of files each process should run on. The scheduler will do its best to keep this requirement, but it's not guaranteed to succeed. If a correct distribution is not found, the user will be asked to validate it. Note: For a more detailed overview of how SUMS prepares datasets please refer to this *flowchart*. Note: This attribute cannot be mixed with *minEvents*, *maxEvents*, or *eventsPerHour*.	N/A
eventsPerHour (optional)	integer or floating point	Tells the scheduler how many events your job can process in one hour. This information is used by the scheduler to determine an estimate of the job execution time. This is necessary to determine the correct usage of resources (e.g. use the long or short queue). This can be combined with minEvents and/or maxEvents to select a queue. Note: This attribute cannot be mixed with *minFilesPerProcess*, *maxFilesPerProcess*, or *filesPerHour*. Note: Available in version 1.10.6 and up	Infinite
filesPerHour (optional)	integer or floating point	Tells the scheduler how many input files per hour the job is going to analyze. This information is used by the scheduler to determine an estimate of the job execution time. This is necessary to determine the correct usage of resources (e.g. use the long or short queue). By combining the use of filesPerHour and minFilesPerProcess, you can basically tell the scheduler what is the minimum time required by your job, and force the use of long queues. If this attribute is not provided, the job is assumed to be instantaneous (e.g. the processes will be dispatched to the short queue no matter how many input files it has). The run time of a job can be calculated as: (60/(filesPerHour))*Nfile = JobRunTime example: A job with 4 files where each file takes two hours to run (in other words .5 files per hour) would take (60/.5)*4 = 480 minutes to run. Note: In version 1.10.0+, if no input files are defined inside the request (such as when using nProcesses) the filesPerHour parameter is treated directly as the number of hours each job will run for. Note: For a more detailed overview of how SUMS prepares datasets please refer to this *flowchart*. Note: This attribute cannot be mixed with *minEvents*, *maxEvents*, or *eventsPerHour*.	Infinite

fileListSyntax (optional)	"paths" or "rootd" or "xrootd"	Tells the scheduler which syntax to use for the file list.
		
		"paths" syntax returns both files on local disk and on distributed disk as a normal path used by the filesystem. This syntax is useful within scripts. The "paths" syntax looks like this
		
		`/path1/file1`
		`/path2/file2`
		`/path3/file3`
		`...`
		
		"rootd" syntax returns files on distributed disk with paths, and files on local disk with the rootd syntax. It also appends the number of events contained in each file. This file syntax is designed to work with the MuDST makers, and has two advantages:
		
		• It allows root to access files that are on the local disk of a different node, making it possible to guarantee the minFilesPerProcess
		• By giving the number of events in the files, the MuDST maker doesn't have to pre-scan the files, slightly improving performance.
		
		The "rootd" syntax looks like this
		
		`/NFSpath1/file1 nEvents1`
		`/NFSpath2/file2 nEvents2`
		`root://machine//path3/file3 nEvents3`
		`root://machine//path4/file4 nEvents4`
		`...`
		
		"xrootd" (eXtended rootd) syntax returns files in almost the same way as rootd syntax. By using the xrootd syntax comes many advantages/features:
		
		• It allows to access files from HPSS that can be dynamically staged from HPSS to a local disk (Currently **DISABLED**)
		• It gives additional fault-tolerance to running jobs, missing requested file can be staged from HPSS and job will not die (Currently **DISABLED**)
		• It offers load balancing mechanism (a single overloaded/missing node serving requested file will not prevent your job from running)
		• It offers faster submission to queue (remote reading of a file is preferred prior to local reading and direct submission constraint of a specific node)
		
		The "xrootd" syntax looks like this
		
		`/NFSpath1/file1 nEvents1`
		`/NFSpath2/file2 nEvents2`
		`root://xrdstar.rcf.bnl.gov:port//path3/file3 nEvents3       ... local file`
		`root://xrdstar.rcf.bnl.gov:port//path4/file4 nEvents4       ... HPSS file`
		
		At the end xrootd syntax is slightly different from rootd syntax by directing all requests to specific node (xrdstar.rcf.bnl.gov).
		
		• Note: SUMS version 1:10:11 and before will drop non-MuDst files using xrootd fileListSyntax and starting at SUMS version 1:10:12 SUMS will except all files under the xrootd fileListSyntax.
		• Note: xrootd staging from HPSS is turned off because it has proven to be too inefficient
		• Note: For a more detailed overview of how SUMS prepares datasets please refer to this *flowchart*.
			paths

~~minStorageSpace~~ *Use <ResourceUsage> Instead (click here)*	any integer	Tells the scheduler the minimal storage space (Disk space) a job will need to run (in MB). A job will not be scheduled on a node having less space. Note: 1) Only version 1.7.0 and up. 2) Deprecated in version 1.10.0 and higher, use <ResourceUsage> element instead (also in this manual). 3) Including this parameter may significantly affect the length of time (shorter or longer) your jobs stays in the pending state.	none (whatever the batch system allows)
~~maxStorageSpace~~ *Use <ResourceUsage> Instead (click here)*	any integer	Tells the scheduler the maximum storage space (disk most likely) a job will need to run (in MB). If not specified the job may fail if it has not enough space. This value may be used for advanced reservation of storage space. This is necessary to determine the correct usage of resources. Note: 1) Only version 1.7.0 and up. 2) Deprecated in version 1.10.0 and higher, use <ResourceUsage> element instead (also in this manual). 3) Including this parameter may significantly affect the length of time (shorter or longer) your jobs stays in the pending state.	none (whatever the batch system allows)
~~minMemory~~ *Use <ResourceUsage> Instead (click here)*	any integer	Minimum memory expected for an individual job (in MB). Setting this value will affect the scheduling priority. To most batch systems this usually represents the minimum memory that must be available before the job is started. Note: 1) Only version 1.7.0 and up. 2) Deprecated in version 1.10.0 and higher, use <ResourceUsage> element instead (also in this manual). 3) Including this parameter may significantly affect the length of time (shorter or longer) your jobs stays in the pending state.	none (whatever the batch system allows)
~~maxMemory~~ *Use <ResourceUsage> Instead (click here)*	any integer	Maximum memory an individual job is expected to use (in MB). Setting this value will affect scheduling priority. To most batch systems this usually represents the maximum memory a job is allowed to consume before it is terminated. Note: 1) Only version 1.7.0 and up. 2) Deprecated in version 1.10.0 and higher, use <ResourceUsage> element instead (also in this manual). 3) Including this parameter may significantly affect the length of time (shorter or longer) your jobs stays in the pending state.	none (whatever the batch system allows)
inputOrder (optional)	a string that is a catalog attribute	Tells the scheduler that you want your input files ordered according to the value of some catalog attribute. This is not going to provide the filelists always in sequence: there can always be gaps. It's only going to reorder the filelists after they are produced. Note: This options is only possible if all the inputs are catalog queries.	none (whatever the batch system allows)
~~minWallTime~~ *Use <ResourceUsage> Instead (click here)*	any integer	This attribute is allowed, however at the current time it has no function. Note: 1) Only version 1.7.5 and up. 2) Deprecated in version 1.10.0 and higher, use <ResourceUsage> element instead (also in this manual).	none (whatever the batch system allows)
~~maxWallTime~~ *Use <ResourceUsage> Instead (click here)*	any integer	The maximum wall clock time for which a job can run before it is terminated . Note: 1) Only version 1.7.5 and up. 2) Deprecated in version 1.10.0 and higher, use <ResourceUsage> element instead (also in this manual).	none (whatever the batch system allows)
logControl	a string. The values can be "standard" or "UCM"	Setting logControl to UCM will make the scheduler send its logging information about jobs submitted to the UCM logger. Note: 1) Only version 1.10.1 and up. 2) Not ready for public use (only used by beta testers)	standard

Below is a list of dispatcher and the job tag attributes each supports. It should be noted that the attributes (simulateSubmission, nProcesses, maxFilesPerProcess, minFilesPerProcess, filesPerHour, fileListSyntax, inputOrder) are supported by all dispatchers, as they are observed at a higher level.

Dispatcher Class	name	mail	maxStorageSpace	minStorageSpace	maxMemory	minMemory	minWallTime	Priority	maxWallTime	kill	status	resubmit
BOSSDispatcher	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
CondorDispatcher	YES	YES	NO	YES	NO	NO	NO	YES	NO	YES	YES	YES
CondorGRSLDispatcher	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES	YES
LSFDispatcher	YES	YES	NO	YES	YES	YES	NO	NO	NO	YES	YES	YES
PBSDispatcher	YES	YES	NO	NO	NO	NO	NO	NO	YES	NO	NO	NO
SGEDispatcher	YES	YES	NO	YES	NO	YES	NO	NO	NO	YES	YES	YES

3.3 The <command> element

The <command> element doesn't have any attributes, and the data that it contains is the actual command script that will be submitted using a csh script.

You can use environment variable to retrieve special information, such as the JOBID, the FILELIST, or more. But remember that the command line will be passed as it is, and therefore if csh doesn't perform the substitution (for example, because part of your command containing the variable is between '...'), the scheduler won't. Refer to csh man pages.

If you have doubts on the correct execution of your command, you can simulate the submission and manually check the script.

Some environment variable you may use are:

$FILEBASENAME - This is only set when the job has one input file per process. It is the name of that input file with the extension stripped off.

$FILELIST - The path to a list of your jobs input files, in plane text file format.

$INPUTFILECOUNT - The number of input files you are using for that given job.

$INPUTFILE[n] - Where [n] is an integer. In input file numbered zero - n. The formatting can be paths, rootd, or xrootd.

Example of rootd: root://rcas6302.rcf.bnl.gov//data1/starlib/reco/productionHigh/FullField/P05ib/2004/051/st_express_5051111_raw_1070015.Mu Dst.root

Example of paths: /data1/starlib/reco/productionHigh/FullField/P05ib/2004/051/st_express_5051111_raw_1070015.Mu Dst.root

$JOBID - A unique identifier given to each job in the format of [$REQUESTID]_[$PROCESSID] example: 10AF9AD61E8F0F2FA172DD9BD007286E_0

$JOBINDEX - A unique number given to each job numbered n though 0. This is not unique among request. Examples 5, 4, 3 , 2, 1, 0
Note: This replaces JOBINDEX in newer versions

$REQUESTID - A hex string that is the same for each job in a request, but unique among requests. Example: 10AF9AD61E8F0F2FA172DD9BD007286

$SCRATCH - The location of the scratch space, this is a space where you have write access. It will be deleted when the job is done. This location is different on most sites. Eaxmple: /tmp/$USER/$JOBID

> **Warning:** The $SCRATCH variable is the path to your scratch area this is where your job starts running. The $SCRATCH path and all it's sub directories are deleted after completion of your job. Do not attempt to reset the variable as it will delete the directory it is set to when the job finishes running.

$SUBMITTINGDIRECTORY - The directory the job where submited from. Example: /star/u/lbhajdu/temp

$SUBMITTINGNODE - The node the job was submitted from. Example: rcas6009.rcf.bnl.gov

$SUMS_nProcesses - The total number of jobs being submitted. Examples 5, 4, 3 , 2, 1, 0

Some examples of the command element are:

```
<command>echo test</command>
```

```
<command>root4star -q -b numberOfEventsList.C\(\"$FILELIST\"\)</command>
```

```
<command>
stardev
root4star -q -b findTheHiggs.C\(234,\"b\",\"$JOBID\",\"$FILELIST\"\)
</command>
```

3.4 Resource requirements : <ResourceUsage> element and sub-elements

Note: This element was added is SUMS version 1.10.0 and above.

Specifying the resource requirements (run time (wall or CPU), ram memory, disk space) of your job allows the scheduler to encode and pass the parameters to the batch system and to select the queue which satisfies the requirements. If you have reason to suspect that your jobs have a defect which may cause them to exceed reasonable resource requirements setting the Max attributes will in most batch systems cause the job to be killed if the job attempts to consume more resources then the ceiling you set. Not all batch systems support all resource parameters, so in some cases it is not always possible to pass some resource requirements, in which case they will be omitted.

Element	Sub-Elements	Sub-Elements	DataType	Meaning	Default
<ResourceUsage> (optional) Must have at least one sub-element	<Memory> (optional) Must have at least one sub-element	<MinMemory> (optional)	unsigned integer	Minimum memory expected for an individual job (in MB). Setting this value will affect the scheduling priority. To most batch systems this usually represents the minimum memory that must be available before the job is started. Note: Including this parameter may significantly affect the length of time (shorter or longer) your jobs stays in the pending state.	none
		<MaxMemory> (optional)	unsigned integer	Maxmum memory expected for an individual job (in MB).	none
	<StorageSpace> (optional) Must have at least one sub-element	<MinStorage> (optional)	unsigned integer	Tells the scheduler the minimal storage space (Disk space) a job will need to run (in MB). A job will not be scheduled on a node having less space. Note: Including this parameter may significantly affect the length of time (shorter or longer) your jobs stays in the pending state.	none
		<MaxStorage> (optional)	unsigned integer	Tells the scheduler the maximum storage space (disk most likely) a job will need to run (in MB). If not specified the job may fail if it has not enough space. This value may be used for advanced reservation of storage space. This is necessary to determine the correct usage of resources. Note: Including this parameter may significantly affect the length of time (shorter or longer) your jobs stays in the pending state.	none
	<Times> (optional) Must have at least one sub-element	<MinWallTime> (optional)	unsigned integer	This attribute is allowed, however at the current time it has no function.	none
		<MaxWallTime> (optional)	unsigned integer	The maximum wall clock time for which a job can run before it is terminated .	none
	<Priority> (optional)	none	unsigned integer from 0-100	A relative job priority for the user. Number between 1...100. The highest the better. Note: This element was added is SUMS version 1.10.1 and above (curently only works with condor)	50

Some examples are examples:

This job takes at least one hour of wall time:

```
<?xml version="1.0" encoding="utf-8" ?>
<job>

   <command>root4star -q -b numberOfEventsList.C\(\"$FILELIST\"\)</command>

   <ResourceUsage>
        <Times>
              <MinWallTime>1</MinWallTime>
        </Times>
   </ResourceUsage>

   <stderr URL="file:./sched$JOBID.error.out" />
   <stdout URL="file:./sched$JOBID.out" />
   <output fromScratch="*.root" toURL="/star/u/lbhajdu/" />

</job>
```

Below is the "all out assault" example. These jobs will each require about 5MB of memory (ram), 100MB of storage and one hour of run time. The job may be terminated if it uses more then 16MB of memory or 100MB of disk space or more then 2 hours of wall time.

```
<?xml version="1.0" encoding="utf-8" ?>
<job>
```

```
        <command>root4star -q -b numberOfEventsList.C\(\"$FILELIST\"\)</command>

        <ResourceUsage>
                <Memory>
                        <MinMemory>5</MinMemory>
                        <MaxMemory>16</MaxMemory>
                </Memory>
                <StorageSpace>
                        <MinStorage>100</MinStorage>
                        <MaxStorage>200</MaxStorage>
                </StorageSpace>
                <Times>
                        <MinWallTime>1</MinWallTime>
                        <MaxWallTime>2</MaxWallTime>
                </Times>
                <Priority>75</Priority>
           </ResourceUsage>

           <stderr URL="file:./sched$JOBID.error.out" />
           <stdout URL="file:./sched$JOBID.out" />
           <output fromScratch="*.root" toURL="/star/u/lbhajdu/" />

   </job>
```

3.5 I/O streams: <stdin> <stdout> <stderr> elements

To specify the standard input/output/error, you have to use the three elements <stdin> <stdout> and <stderr>. All these elements must have the URL attribute specified to point their location.

Attribute	Allowed values	Meaning	Default
URL	a file URL without host (e.g. "file:/path/filename")	tells the scheduler to which file redirect the standard input, output or error. The URL must be of the file protocol (that means that is a local file, accessible via file system), and it should be visible on all machines (for example, a file on NFS or AFS). Remember that the stdout and the stderr must be different for every process, otherwise all the process that the scheduler will divide your job in will overwrite the same file. To achieve that, you can use the $JOBID environment variable.	no default
discard	"true" (valid only for stdout and stderr)	tells the scheduler to discard the stream. This attribute is valid only for stdout and stderr. Be careful when using this option: when using the GRID you don't know where your job is going to run, and the standard output/error are crucial to understand what went wrong.	no default

Some examples are:

```
<stdin URL="file:/star/u/user/pion/central/myinput.in"/>
<stdout URL="file:/star/u/user/pion/central/$JOBID.out"/>
<stderr URL="file:/star/u/user/pion/central/$JOBID.err"/>
```

```
<stdin URL="file:/star/u/user/scheduler/inputs/goldFullField.param"/>
<stdout URL="file:/star/u/user/scheduler/gold/fullField/$JOBID.out"/>
<stderr URL="file:/star/u/user/scheduler/err/goldFullField$JOBID.err"/>
```

```
<stdout discard="true" />
```

3.6 The <input> element

The <input> element is used to specify data input files. Input files can be specified by either a path and filename resident on network mounted disks, such as AFS or NFS; it can be a file on a local disk; it can be a query to the file catalog. We suggest that you use the latter, because it provides the system more flexibility on how to allocate your process.

One can specifies more than one input file element. You can mix NFS files with local files and catalog queries. You can have more than one catalog query. To specify the location of the input files, you still use an URL.

Attribute	Allowed values	Meaning	Default

URL	a URL starting with file: or catalog:	Tells the scheduler which input files to associate to the processes.
		
		Catalog query: To specify a query to the file catalog, you should write "catalog:star.bnl.gov?query". catalog: tells that the URL is a catalog query; star.bnl.gov tells you are querying the catalog for star at BNL, and query is the actual query. The query is a comma separated keyword value pair ("keyword1=value1,keyword2=value2") that will be forwarded to the file catalog. The syntax is the same allowed for the command line interface of the file catalog at the -cond parameter.
		
		Network file: To specify a file that is accessible on all machines on the same file path, you should write:
		
		`file:/path/name`
		`file:/path/name`
		`file:/path/name`
		`...`
		
		File on local disk: To specify a file that is resident on a target machine, that is a machine on which the scheduler is allowed to submit the job, you should write:
		
		`file://hostname/path/name`
		`file://hostname/path/name`
		`file://hostname/path/name`
		`...`
		
		Filelist: You can specify a text file that is going to contain a list of files on which to run your analysis. You should write:
		
		`<input URL="filelist:/path/nameOfList" />`
		
		An example of format of the list its self is:
		
		`file:/star/data32/reco/productionHigh/FullField/P04ik/2004/034/st_physics_5034018_raw_1010002.MuDst.root`
		`file://rcas6072.rcf.bnl.gov/data1/starlib/reco/dAuTOF/FullField/P03ia/2003/069/st_physics_4069015_raw_0040052.MuDst.root`
		`...`
		
		Note: In version 1.10.2 and up you may also enter any kind of string you want in a file list. For example a list of run numbers. This will be split up in your file list (.list file) just like any other file would be. Some examples of formats you can use are:
		
		`string:8177035`
		`string:11157001`
		`string:1234567890`
			no default

```
string:"aaa bbb 123"
string:'any string'
...
```

Note: In version 1.10.2 and up you may enter file names and paths without the "file:" in front of them. However this format does not let you specify which node the file is on. Example:

```
/star/data32/reco/productionHigh/FullField/P04ik/2004/034/st_physics_5034018_raw_1010002.MuDst.root
/star/u/user/myrequest/input.root
...
```

The format of the fileList is specified by the fileListSyntax[s] keyword in the <job> element, some options are (paths, rootd, xrootd). See the *job element section* for deatils.

Command: You can use the command format if you want to use another program to produce the list of files to be used as the input. The program will be executed and its standard output stream will be piped back to the parent program and used as the file list. It must be written in one of the same formats the file list uses. The command can include arguments, but do not assume fancy shell tricks will work if passing exotic arguments, that sort of thing should be taken care of internal to your program. Example:

```
<input URL="command:/usr/bin/find /star/u/bob/myfiles/" nFiles ="30" />
```

Note:The command input can only be used in STAR Unified Meta Scheduler 1.10.6 and higher.

Some examples are:

```
<input URL="file:/star/data15/reco/productionCentral/FullField/P02ge/2001/322/st_physics_2322006_raw_0016.MuDst.root" />
<input URL="file:/star/data15/reco/productionCentral/FullField/P02ge/2001/*/*.MuDst.root" />
<input URL="file://rcas6078.rcf.bnl.gov/home/starreco/reco/productionCentral/FullField/P02gd/2001/279/st_physics_2279005_raw_0285.MuDst.root" />
<input URL="filelist:/star/u/user/username/filelists/mylist.list" />
<input URL="catalog:star.bnl.gov?production=P02gd,filetype=daq_reco_mudst,storage=local" nFiles="2000" />
```

The file catalog is actually a separate tool from the scheduler. You can find the documentation for the file catalog, specifying all the keywords and options, at: *http://www.star.bnl.gov/comp/sofi/FileCatalog.html*. It should be noted that the scheduler will drop duplicate files if they are on different hosts but have the same name and path. This is why sometimes the number of files returned for the catalog query is higher then the number selected.

If the URL represents a file catalog, more attributes are available to better specify the query:

Attribute	Allowed values	Meaning	Default
nFiles (optional)	an integer or "all"	The number of files returned by the query	100
singleCopy (optional)	"true" or "false"	Specify if the query should return one copy for each file, or it should return multiple copies if they are available. For example, suppose one file has two copies: one on rcas6060.star.bnl.gov and one on NFS. By selecting "true", only one of them is returned. By selecting "false", both of them can be returned. In the second case, you job will actually run on two copies of the same file. By default only one copy of the same file is returned.	true
preferStorage (optional)	"NFS" or "local" or "HPSS"	When multiple copies are available for one file, this attribute is used to choose which particular copy to get. This attribute has meaning if singleCopy is not set to false. If more than one copy is available which the preferred storage (for example, a file is available on two different machines), one copy is chosen at random. This attribute was introduced because small jobs on small set of files are penalized when dispatched on local files: they have to wait for a particular machine to free up, and that might take a long time even if the rest of the farm is free. Executing on local files make each job faster, but it might increase the waiting time before the job gets executed. Therefore, NFS is recommended only for testing your analysis on a small set and local when you run on the entire set. Remember that the query might return local files even if you chose NFS. If you want _only_ NFS or local files, then put "storage=NFS" inside your query. The preferStorage can be HPSS only if the attribute fileListSyntax is equal xrootd. Then if the file has multiple copies, HPSS file is preferentially selected. Remember that accessing files from HPSS will take more time than from distributed or local disk.	N/A
limit (optional)	deprecated Don't use.	The number of files returned by the query	N/A
start (optional)	deprecated Don't use.	As in the file catalog interface, the offset on which the query should start	N/A

More examples of files catalog queries:

```
<input URL="catalog:star.bnl.gov?filetype=daq_reco_mudst,collision=AuAu200,magscale=FullField,trgsetupname=ProductionMinBias" preferStorage="NFS" nFiles="all" />
<input URL="catalog:star.bnl.gov?production=P03ia,trgsetupname=dAuMinbias,filetype=daq_reco_MuDst,magscale=ReversedFullField" />
<input URL="catalog:star.bnl.gov?storage=NFS,trgsetupname=dAuMinBias,filetype=daq_reco_MuDst,magscale=FullField" nFiles="1000" />
<input URL="catalog:star.bnl.gov?production=P03ia,filetype=daq_reco_mudst,collision=dAu200,magscale=FullField,trgsetupname=dAuTOF" nFiles="150" />
<input URL="catalog:star.bnl.gov?production=P03ia,runnumber=4044036,filetype=daq_reco_MuDst" preferStorage="local"/>
<input URL="catalog:star.bnl.gov?production=P03ia,filetype=daq_reco_MuDst,storage=NFS,trgsetupname=dAuMinBias" nFiles="all" />
```

3.7 The <output> element

The <output> element is used to specify the output produced by your code. With this tag, you will be able to write your output on a local scratch directory on the node the job will be dispatched to, and the scheduler will copy your output at the end of the job. This will make better use of I/O resources.

The environment variable $SCRATCH will contain a path available for your job. This space is unique for each process your job will be divided into, and will be deleted after you job ends. With the <output> element you are able to specify which files you want to bring back. You don't need to bring everything back, of course, but the output won't be available anymore later.

Remember that your job will be divided into different processes, and that all the processes should use different output filenames, or otherwise they will rewrite their outputs. You can always use the $JOBID to create unique filenames.

Attribute	Allowed values	Meaning	Default
fromScratch	a file / wildcard / directory accessible from the scratch directory	With this attribute you specify either a file, a wildcard or a directory to be copied back. The file, wildcard or directory must be expressed relative to the $SCRATCH directory. That is, to retrieve all the .root files your job saved in the $SCRATCH, simply use *.root in this attribute. If the file you need to recover is in the current working directory and that is not the $SCRATCH directory you must put ./ before the file name, for example ./*.root.	no default
	a URL starting	tells the scheduler where to copy the output. The URL must represent either a network file or directory. Network file. To specify a file, you should write "file:/path/name". You can specify a file here only if the output you specified is a file (you are not allowed to copy a directory in one file). You can specify a different name so that the file will be brought back with the different name.	no

toURL	with file:	Network directory. To specify a directory, you should write "file:/path/". For grid services. To specify a directory, you should write "gsiftp:/path/". This uses "globus-url-copy" otherwise the "cp" command is used.	default

Let's make some examples:

```
<output fromScratch="*.root" toURL="file:/star/u/carcassi/scheduler/out/" />
<output fromScratch="a.root" toURL="file:/star/u/carcassi/scheduler/out/$JOBID.root" />
<output fromScratch="out/" toURL="file:/star/u/carcassi/scheduler/out/" />
```

The first example copies all the files in the $SCRATCH directory that end in .root to the specified NFS directory.

The second example, copies the a.root file in the $SCRATCH directory, and copies it in the specified directory as $JOBID.root. This is useful if you can't/don't want to change the script to generate unique filenames.

The third example copies the entire out directory in the $SCRATCH location in the target directory

4.0 The <sandbox> element

Special Notes:

- This tag is available from scheduler version 1.8.7 and beyond.
- Only one sandbox tag is allowed per request

This element allows you to copy or soft link files and directory trees to be placed in the location your job is to be executed from or another alternative location on the worker node.

Attribute	Allowed values	Meaning	Default
installer	ZIP PACMAN (This attribute is optional)	If this attribute is blank the sand box will not package and un-package the files it will simply link and or copy them to the startup location of your job. If the install tag is present and set to "ZIP" the sandbox will create a folder in the current working directory in which it will copy the necessary files. This directory will be zipped then copied to the location your job starts up. The "PACMAN" implementation uses prepackage PACMAN packages which are unpackaged and installed. This implementation is still in development.	none, no installer is to be used and files are just coped or linked

4.1 The <Package> element of <sandbox>

The package element states what packages are to be copied. A package can be a set of files, or a prepackaged zip or packman file.

Attribute	Allowed values	Meaning	Default
name	any string, the name of the package	This attribute is used when the sandbox installer is set to ZIP or PACMAN, this states the name of the package. If using the ZIP implementation and the repository attribute is missing it will look in the current working directory for the package. A ".zip" extension is generally used on this file, however if it is missing, it will be assumed that it should have one.	none in case of the ZIP implementation a string will be hashed for the zip package name
version	a string, the version number	This option is only relevant when using the PACMAN installer. It allows you to select the version software you wish to install. **Information on available packages will be posted later, as this function becomes ready.	none
installdir	URI	By default the sandbox copies packages to the directory your job starts in. This behavior can be changed by setting the "installdir" attribute.	The directory your job starts in.
repository	URI	In the case of PACMAN this attribute must hold the directory path with the server name.	Your current working directory

4.2 The <File> element of <Package> and some examples

The "File" element has no attributes it simply encloses the the files name. A package does not need to have any files if it's "name" attribute holds a referents to an existing package.

Here are some examples:

```
<SandBox>
        <Package>
                <File>file:./test2.C</File>
                <File>file:./macro/</File>
                <File>file:/star/u/bob/mis/</File>
        </Package>
</SandBox>
```

Note: We are using the word "copy" loosely here, if the "installer" attribute of the sandbox is not set and the job is local the files will nearly be soft linked, so be careful in such cases not to modify the file thinking that it is a copy. On the grid a copy of the file is required, unlike the zip flavor of the sandbox this does not preserve directory paths and file attributes or permissions.

The first file element in the package ("file:./test2.C") says copy test2.C from my current working directory (from where I run the scheduler) to the place my job starts.

The next element does the same except this time we are copying a directory and all its sub directories and files. This makes testing from your home directory and running from another directory simple, as sandboxed directories will still be relative to the current working directory.

The last file element in the example above is an absolute path. In the example above the mis directory tree will appear in the jobs current working directory.   It should be noted that inside a relative path the ".." parent directory can on be referred to. In other words <File>file../mystuff<File> or <File>file./../mystuff<File> or <File>file./macros/../mystuff<File> are all forbidden, however <File>file:/star/u/bob/mis/../macros<File> is alright.

```
   <SandBox installer="ZIP">
          <Package name="mydemox">
                 <File>file:./demo/</File>
                 <File>file:./demo2/</File>
          </Package>
   </SandBox>
```

The example above will create a directory called mydemox.package (the name will be different if you already have that file) in the current working directory. It will copy into this file the directories ./demo/ and ./demo2/ the mydemox.package will be zipped in a file called mydemox.zip and copied if needed and unzip in the users current working directory. The next time the job is run, if the zip file is still there (or one with the same name), it will just use the zip file instead of rebuilding the zip file.

!!! Warning : ZIP packages are only built once. Any changes made to files or directories referred to by the package after the packing of the package will not be applied unless the old package is deleted. If preferred the zip file can be also modified directly. This means if you submit a job, then change your sandboxed compiled code, the package will still be using the old code. please be positive you understand this, otherwise you will be pulling your hair out, not understanding why your macro is not working.

5.0 The <output> actions (still in development)

Note: This section only applies to scheduler version 1.8.1 and above:

The scheduler allows a more detailed way to specify what to do with the output. Right now, it allows you to perform three operations: copy, link and register. Copy allows you to copy the file to another location, even a remote site, and register allows you to register a copy of the file in the file catalog [NB the scripts that provides these functionalities have still to be prepared].

5.0.1 The <copy> element (a branch of the output element):

This tag is used to copy a file from scratch to another location. It work both local and via the grid (site to site). It should be noted that you still need the appropriate authentication certificates to get resource access via the grid.

no default

Attribute	Allowed values	Meaning	Default
id	any string (optional)	Allow the assignment of a ID (name) to the result of this action element, so that it can be referenced later by another action element by way of the referto attribute. Using this attribute has no affect on the action. Note: The the element containing the id most appear above (referring to the order of the elements) the element containing the referto.	no default

referto	any string that is the id of another copy or link element in the same output branch	If there is no referto this tag will act on the file specified by the fromScratch element of the parent output tag, However if there is a referto referring to another action tag it will act on the output of that tag. In the example below the file a.root is copied from the scratch directory on a worker node somewhere on the grid to the users home directory star/u/john/ under the name b.root. The the file /star/u/john/b.root is copied to the directory star/u/john/temp/ under the name c.root. If the referto was not there then a.root and would be copied to c.root in /star/u/john/temp/.
		
		`<output fromScratch="a.root">`
		`     <copy id="myFile" to="file://star/u/john/b.root" Type="grid"/>`
		`     <copy referto="myFile" to="file://star/u/john/temp/c.root" Type="local"/>`
		`</output>`
		
		Note: The element containing the referto attribute should appear below the element containing the id attribute. This is because you can not reference a file(s) that does not exist yet.
			no default

to	a path (required)	The path the file should be copied to.	no default
retries	a positive integer (optional)	How many times should it try to copy the file over before it declares failure to copy the file.	1
sleep	a positive integer (optional)	How long the action should sleep (delay) in-between retries.	0
Type	"local" or "grid"	Defines the type of the copy that is needed. For a simple copy with mapped drives a simple "cp" command can be used. With site to site copies (grid) the scheduler will employ a more appropriate command such as globus-url-copy or SRM-copy depending on configuration.	no default

5.0.2 The <link> element (a branch of the output element):

Creates a link to a file.

Attribute	Allowed values	Meaning	Default
id	any string (optional)	Allow the assignment of a ID (name) to the result of this action element, so that it can be referenced later by another action element by way of the referto attribute. Using this attribute has no affect on the action. Note: The the element containing the id must appear above (referring to the order of the elements) the element containing the referto.	no default

referto	any string that is the id of another copy or link element in the same output branch	If there is no referto this tag will act on the file specified by the fromScratch element of the parent output tag, However if there is a referto referring to another action tag it will act on the output of that tag. In the example below the file a.root is copied from the scratch directory on a worker node somewhere on the grid to the users home directory star/u/john/ under the name b.root. A link is created from called /star/u/john/temp/root.link, it links to the file /star/u/john/b.root. If the referto was not there then a.root and would be linked to root.link (which is not on the local cluster !).
		
		`<output fromScratch="a.root">`
		`      <copy id="myFile" to="file://star/u/john/b.root" Type="grid"/>`
		`      <link referto="myFile" to="file://star/u/john/temp/root.link" Type="local"/>`
		`</output>`
		
		Note: The element containing the referto attribute should appear below the element containing the id attribute. This is because you can not reference a file(s) that does not exist yet.
			no default

to	a path (required)	The path where the link can be placed. If the name of the link is different from that of the original original file it can be added to the end of the path. In the example below the first link action makes a link in the directory /star/u/john/temp/ named root.link the second link action makes a link in the same directory called a.root. Because no name was specified it took the name of the original file.
		
		`<output fromScratch="a.root">`
		`        <link referto="myFile" to="file://star/u/john/temp/root.link" Type="local"/>`
		`        <link  to="file://star/u/john/temp/" Type="local"/>`
		`</output>`
			no default

retries	a positive integer (optional)	How many times should it try to make the link before it declares failure to make the link (assuming there is a problem trying to create the link).	1

sleep	a positive integer (optional)	How long the action should sleep (delay) in-between retries.	0
kind	"hard" or "soft"	Defines whether the link should be a hard link or a soft link.	no default

5.0.3 The <register> element (a branch of the output element):

Registers a file in the file catalog.

Attribute	Allowed values	Meaning	Default
id	any string (optional)	Allow the assignment of a ID (name) to the result of this action element, so that it can be referenced later by another action element by way of the referto attribute. Using this attribute has no affect on the action. Note: The the element containing the id must appear above (referring to the order of the elements) the element containing the referto.	no default
referto	any string that is the id of another copy or link element in the same output branch	If there is no referto this tag will act on the file specified by the fromScratch element of the parent output tag, However if there is a referto referring to another action tag it will act on the output of that tag. Example: <register to="catalog?dataset=mydata&amp;production=myprod"/>	no default
to	a path (required)	The file two which the catalog entry is made.	no default
retries	a positive integer (optional)	How many times should it try to make the link before it declares failure to make the link (assuming there is a problem trying to create the link).	1
sleep	a positive integer (optional)	How long the action should sleep (delay) in-between retries.	0

5.1 The <Generator> Element

Note: This section only applies to STAR Scheduler version 1.10.0c and above.

The <Generator> element doesn't have any attributes, it is placed somewhere inside of the job tag, preferably group with all the other IO tags. It contains sub elements (tags) that redirect the location of files the scheduler needs to write. It will allow you to write all the messy file output generated by SUMS to a location other then your current working directory. The tag <Generator> can contain the <Location> element, <ReportLocation> element, <ScriptLocation> element, and/or the <ListLocation> element. None of these sub elements have attributes as well, and the data they contain is a string which is the path to which the scheduler should write the files to. This string may be relative (./bla) or absolute (/star/users/bob/jobs).

Note: The paths in these tags may not contain environment or scheduler specific variables because they will not be resolved.

Note: The paths in these tags must also already exist because the scheduler will not create it for you.

Below is a definition of the sub elements of <generator> followed by examples:

Tag	Meaning	Default

<Location>	Sets the default for all file paths, unless otherwise specified by <ScriptLocation>,< ReportLocation > and or < ListLocation>
	
	Example: Write all the *.csh, *.list, *.condor, *.report into
	the ./myFiles/ directory.
	
	`<Generator>`
	`                <Location>./myFiles</Location>`
	`</Generator>`
		./

<ScriptLocation>	Sets the location to which the scheduler will write the *.csh files to.
	Note: The scheduler will also write the *.condor and *.condorg files to this location if using the condor or condorg dispatcher.
	
	Example: Write all the *.csh (and *.condor or *.condorg) files to ./myScripts and all other files (*.list, *.condor, *.report) into
	the ./myFiles/ directory.
	
	`<Generator>`
	`                <Location>./myFiles</Location>`
	`                <ScriptLocation>./myScripts</ScriptLocation>`
	`</Generator>`
	
	Example: Write all the *.csh (and *.condor or *.condorg) files to ./myScripts and all other files to my current working directory.
	
	`<Generator>`
	`     <ScriptLocation>./myScripts</ScriptLocation>`
	`</Generator>`
		value of <Location> tag

<ReportLocation>	Sets the location to which the scheduler will write the *.report file to.
	
	Example: Write all the *.report and *.condor.log (if using condor) to the ./myReports location.
	
	`<Generator>`
	`          <ReportLocation>./myReports</ReportLocation>`
	`</Generator>`
		value of <Location> tag

	Sets the location to which the scheduler will write the *.list file to. Example: Write all the *.list to the ./myLists location.	value of

<ListLocation>	`<Generator>`
	`        <ListLocation>./myLists</ListLocation>`
	`</Generator>`
		<Location> tag

6 Running Other Actions From Within Your Request

6.0.0 The <Action> element:

The action tag is used to intersperse the execution of commands and other jobs in before, between, and or after the main jobs of your request.

Attribute	Allowed values	Meaning	Default
frequency	A positive integer or zero (required)	See table below.	no default
position	The string: "FIRST" or "LAST" or "BEFORE" or "AFTER" (required)	See table below.	no default

frequency	position			
	"FIRST"	"LAST"	"BEFORE"	"AFTER"
0	Execute the <Exec> block as a local forked process before submitting any jobs. -This guarantees this will run before the first job starts. -This block is run as part of the interactive submission and will hang submitting of jobs until it exits. *Implemented in version 1.10.0+*	Execute the <Exec> block, as a job submitted to the batch system after all other jobs have finished. -If even one job fails the <Exec> block will not run. By the job failing we mean that the job script exits with a non-zero value not that the users part of the job has failed. *Implemented in version 1.10.8+*	Execute the <Exec> block as a job submitted to the batch system before any other jobs have been submitted. -It is guaranteed this job will start running and finish before any other jobs start. *Not Implemented Yet*	Execute the <Exec> block as a job submitted to the batch system after all other jobs have been submitted. -It is guaranteed this job will not start running before all other jobs finish. *Not Implemented Yet*
1	Execute the <Exec> block before each job is submitted to the batch system. The <Exec> block has access to the same environment variables as the job itself. -This block is run as part of the interactive submission and will hang submitting of jobs until it exits. *Implemented in version 1.10.8+*	Execute the <Exec> block as a job submitted to the batch system after all other jobs have been submitted. -There is no guaranteed this job will start after all other jobs have started, or that this job will finish after all other jobs have finished, only that it is submitting to the batch system last. *Not Implemented Yet*	Execute the <Exec> block as a separate job submitted to the batch system each and every time before the main job starts running. The main job will not be started before the <Exec> block is done. There may be some delay before the main job starts. *Will only work with the bnl_condor_dag policy* *Implemented in version 1.10.6+*	Execute the <Exec> block as a job submitted to the batch system each and every time after a job has finished. *Not Implemented Yet*
N > 1	N/A (Not applicable)	N/A (Not applicable)	Execute the <Exec> block as a job submitted to the batch system before any of the next N jobs are allowed to start this job must finish. Note: If N is bigger then the total number of jobs produced. The <Exec> block will be executed as a single job submitted to the batch system. The job must finish before the last job is allowed to complete. *Not Implemented Yet*	Execute the <Exec> block as a job submitted to the batch system only after the N previous jobs have finished. Note: If N is bigger then the total number of jobs produced. The <Exec> block will be executed as a single job submitted to the batch system and will only be allowed to run after the last job has completed execution. *Not Implemented Yet*

Examples:

```
<Action frequency="0" position="BEFORE">
        <Exec>
                mkdir $HOME/temp/ActionDir
        </Exec>
</Action>
```

7 Cut and paste examples

For the lazy ones, here are some examples of job descriptions you can modify and use right away.

1. *Executing a simple command*
2. *Executing a root macro on the microDST of a given production*
3. *Executing a long analysis*
4. *Examples with the file catalog*
5. *Specify the input using files and wildcards*
6. *Executing a root macro that takes a single file as an argument*
7. *Writing a parameterized job description*
8. *Creating and submitting a job template*
9. *Using different fileListSyntax in job description*

7.1 Simple command

Executes a simple command, with standard input and standard output mapped to a file

```
<?xml version="1.0" encoding="utf-8" ?>
<job>
    <command>echo 'hello world'</command>
    <stdout URL="file:/star/u/carcassi/scheduler/out/$JOBID.out" />
    <stdin URL="file:/star/u/carcassi/scheduler/input.txt" />
</job>
```

7.2 Executing a root macro on the microDST of a given production

To execute a on all files of the microDST for production P03ia, dAuMinbias reverse full field, you can write

```
<?xml version="1.0" encoding="utf-8" ?>
<job maxFilesPerProcess="500" filesPerHour="100">

   <command>root4star -q -b numberOfEventsList.C\(\"$FILELIST\"\)</command>

   <stdout URL="file:/star/u/carcassi/scheduler/out/$JOBID.out" />
   <input URL="catalog:star.bnl.gov?production=P03ia,trgsetupname=dAuMinbias,filetype=daq_reco_MuDst,magscale=ReversedFullField" />

   <SandBox>
       <Package>
          <File>file:./rootMacros/numberOfEventsList.C</File>
       </Package>
   </SandBox>

   <output fromScratch="*.root" toURL="file:/star/u/carcassi/scheduler/out/" />

</job>
```

Remember to escape the command line as you would on the prompt. The input is specified through a catalog query (consult the following *example* on catalog use)

The environment variable $FILELIST will contain a filename of the input file list the process will be working on. The file list is a text file in which each line is a full path to an input file. Remember that the output files should have a name that is unique for all the processes into which the job will be divided. You can do that if your output file maps the name of the input file, the file list, or the job ID

The macro is going to write the output in the $SCRATCH directory, and the scheduler is going to copy the output in the target directory as it is specified in the output tag.

Notice also the filesPerHour tag, that tells the scheduler how many input files your analysis is going to work in an hour. This will allow the scheduler to dispatch the correct number of files to use the short queue.

7.3 Executing a long analysis:

```
<?xml version="1.0" encoding="utf-8" ?>
<job minFilesPerProcess="100" maxFilesPerProcess="500" filesPerHour="50">
        <command>root4star -q -b numberOfEventsList.C\(\"$FILELIST\"\)</command>
        <stdout URL="file:/star/u/carcassi/scheduler/out/$JOBID.out" />
        <input URL="catalog:star.bnl.gov?production=P02gd,filetype=daq_reco_mudst,storage=local" nFiles="all" />
        <SandBox>
                <Package>
                <File>file:./numberOfEventsList.C</File>
        </Package>
        </SandBox>
</job>
```

The key here are the min/maxFilesPerProcess and the filesPerHour tags. With min/max you define in which range of input files your analysis is going to work. With filesPerHour you define how many input files your analysis is going to analyze. In this case, the minimum is 100 files, and the program is going to analyze 50 files per hour. Therefore, the minimum amount of time it will need is 2 hours. If the short queue doesn't allow a job of such length, the scheduler will decide to use the long queue automatically.

(Note that at PDSF and at BNL the length of the short queue is different: the scheduler will size the jobs accordingly).

7.4 Examples with the file catalog:

The file catalog is actually a separate entity from the scheduler. You can find its full documentation *here*. Here is an example of a description that will execute a macro over the entire collection of microDST for production P02gd.

```
<?xml version="1.0" encoding="utf-8" ?>
<job maxFilesPerProcess="500" filesPerHour="100">
    <command>
stardev
root4star -q -b numberOfEventsList.C\(\"$FILELIST\"\)
    </command>
    <stdout URL="file:/star/u/carcassi/scheduler/out/$JOBID.out" />
    <input URL="catalog:star.bnl.gov?production=P02gd,filetype=daq_reco_mudst" nFiles="all" />
    <SandBox>
        <Package>
                <File>file:./numberOfEventsList.C</File>
        </Package>
    </SandBox>
</job>
```

Notice that you need not care where the files are: the scheduler will select one (and only one) copy for you. The part in the input URL before '?' should be just copied (it tells the scheduler where to find the catalog). The part after the '?' is a generic -cond parameter that you can pass to the file catalog script. nFiles is the attribute that tells the scheduler how many files to retrieve. It can be omitted, and the default value is 100. To get all the files use nFiles="all".

Another example of a job running on a catalog query:

```
<?xml version="1.0" encoding="utf-8" ?>
<job maxFilesPerProcess="500" filePerHour="100">
    <command>
starver SL02i
root4star -q -b numberOfEventsList.C\(\"$FILELIST\"\)
    </command>

    <stdout URL="file:/star/u/carcassi/scheduler/out/$JOBID.out" />
    <input URL="catalog:star.bnl.gov?production=P02gc,trgsetupname=ProductionMinBias,filetype=daq_reco_MuDst,magscale=ReversedFullField,runnumber[]2254002-2276026" nFiles="all" />

        <SandBox>
                <Package>
                        <File>file:./numberOfEventsList.C</File>
                </Package>
    </SandBox>

</job>
```

This will run on the production minbias, reverse full field microDST of production P02gc, from run 2254002 to run 2276026.

7.5 Specify the input using files and wildcards:

You can specify the input by providing all the file names, or by providing a wildcard. Notice that **you shouldn't use this method for production files**: they should be accessed by the file catalog, which is aware when files are moved around. This method is available for all those analysis that require two passes, and in the second you are going to analyze the output of the first.

```
<?xml version="1.0" encoding="utf-8" ?>
<job>
    <command>myscript.csh</command>
    <stdout URL="file:/star/u/carcassi/scheduler/out/$JOBID.out" />

    <input URL="file:/star/data15/reco/productionCentral/FullField/P02ge/2001/322/st_physics_2322006_raw_0016.MuDst.root" />
    <input URL="file:/star/data15/reco/productionCentral/FullField/P02ge/2001/322/*.root" />
    <input URL="file://rcas6068.rcf.bnl.gov/home/starreco/reco/ProductionMinBias/ReversedFullField/P02gd/2001/263/st_physics_2263005_raw_0004.MuDst.root" />
    <input URL="file://rcas6080.rcf.bnl.gov/home/starreco/reco/ProductionMinBias/ReversedFullField/P02gd/2001/263/st_physics_2263006_raw_0112.MuDst.root" />
    <input URL="file://rcas6068.rcf.bnl.gov/home/starreco/reco/ProductionMinBias/ReversedFullField/P02gd/2001/263/st_physics_2263006_raw_0061.MuDst.root" />
    <input URL="file://rcas6068.rcf.bnl.gov/home/starreco/reco/ProductionMinBias/ReversedFullField/P02gd/2001/263/st_physics_2263006_raw_0150.MuDst.root" />
    <input URL="file://rcas6068.rcf.bnl.gov/home/starreco/reco/ProductionMinBias/ReversedFullField/P02gd/2001/263/st_physics_2263001_raw_0040.MuDst.root" />
    <input URL="file://rcas6080.rcf.bnl.gov/home/starreco/reco/ProductionMinBias/ReversedFullField/P02gd/2001/263/st_physics_2263006_raw_0126.MuDst.root" />
    <input URL="file://rcas6068.rcf.bnl.gov/home/starreco/reco/ProductionMinBias/ReversedFullField/P02gd/2001/263/st_physics_2263006_raw_0062.MuDst.root" />

</job>
```

Files that reside on NFS are the ones that have only one slash after "file:", while files that reside on machine are the ones with two slashes and the name of the machine. Inside the script, you can either use the $FILELIST environment variable that points to a file that contains a full path to an input file to use; or you can use the $INPUTFILECOUNT environment variable to know how many input files and $INPUTFILE0, $INPUTFILE1, ..., to know the full name of each input.

7.6 To execute a root macro that takes a single file as an argument you can write:

You might have a script or a macro that only accepts one file at a time. You might want to use that in the scheduler, but it would be more efficient if it could get more than one file at a time, so that it doesn't generate many short jobs. You can do that by writing something like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<job maxFilesPerProcess="10" >
    <command>
@ nFile=0
while ( $nFile &lt; $INPUTFILECOUNT )
eval set filename = '$INPUTFILE'$nFile

# put your script here, with filename as input file
echo $filename

@ nFile++
end
    </command>
    <stdout URL="file:/star/u/carcassi/scheduler/out/$JOBID.out" />
    <input URL="catalog:star.bnl.gov?production=P02gd,filetype=daq_reco_mudst" nFiles="100" />
</job>
```

In the command section there is a simple script that iterates over all the input, and calls the file one by one. If you substitute "echo $filename" with your script or macro, your job will be able to accept more than one file, making things more efficient.

7.7 Writing a parameterized job description:

Sometimes you might want to prepare job descriptions that are almost identical to one another, except for one or two parameters. You can use XML entities to achieve this result. For example, say we want to analyze the data in one run, and put the resulting histogram in a subdirectory that contains that runnumber. We can write

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE note [
<!ENTITY runnumber "4049066">
]>
<job maxFilesPerProcess="500" filesPerHour="100">

        <command>
stardev
root4star -q -b numberOfEventsList.C\(\"$FILELIST\"\)
        </command>

        <stdout URL="file:/star/u/carcassi/scheduler/out/&runnumber;/$JOBID.out" />
        <input URL="catalog:star.bnl.gov?runnumber=&runnumber;,production=P02gd,filetype=daq_reco_mudst" nFiles="all" />

        <SandBox>
        <Package>
                        <File>file:./numberOfEventsList.C</File>
                </Package>
        </SandBox>

</job>
```

7.8 Creating and submitting a job template:

A template is a job description that has some some undefined values that are going to be defined during job submission. For example, you can leave a parameter in your query, a directory in the path of your output, an argument for your macro undefined in the xml, and then decide it when you submit your job. For example:

```
<?xml version="1.0" encoding="utf-8" ?>
<job minFilesPerProcess="1" maxFilesPerProcess="700" filesPerHour="0.5" simulateSubmission="true">

   <command>
stardev
root4star -q -b myMacro.C\(\"$FILELIST\",\"$SCRATCH/dAu200_MC_$JOBID.root\"\, &trigger;)
   </command>

   <output fromScratch="*" toURL="file:/star/u/carcassi/schedulerDevelopment/out/&trigger;/" />
   <stdout URL="file:/star/u/carcassi/schedulerDevelopment/out/&trigger;/sched$JOBID.out"/>
   <stderr URL="file:/star/u/carcassi/schedulerDevelopment/out/&trigger;/sched$JOBID.out"/>
   <input URL="catalog:star.bnl.gov?collision=&collision;,trgsetupname=&trigger;,filetype=MC_reco_MuDst,storage=NFS" nFiles="all" />

   <SandBox>
                <Package>
                        <File>file:./myMacro.C</File>
                </Package>
    </SandBox>

</job>
```

Notice that it has two entities (&collision; and &trigger;) in various places: these should hold a value, but this value is not yet defined in the XML. In fact, this XML is not valid, because these values are not defined. Since the XML is not valid, you can't give it directly to star-submit. Instead, you should use star-submit-template like this:

```
star-submit-template -template requests/EntityExample.xml -entities trigger=minbias,collision=dAu200
```

The template will be expanded, and the values will be substituted (by adding a DOCTYPE section, if you really want to know :-)), and the resulting xml will be submitted through the scheduler.

7.9 Using FILEBASENAME to give your output file the same name as the input file

If a csh script generated by the scheduler only has one file there will be an environment variable with the base name of that file created called FILEBASENAME. In other words, the FILEBASENAME environment variable only exists if maxFilesPerProcess is set to one or if your input is only one file.

(Note : The base name of a file is the files name without a path or any extensions.)
(Note : This will only work in version 1.6.2 and up)

```
<?xml version="1.0" encoding="utf-8" ?>
<job name="Test1" maxFilesPerProcess="1" simulateSubmission="false">



                <command>root4star -q -b numberOfEventsList.C\(\"$FILELIST\"\)</command>

                <stdout URL="file:/star/u/lbhajdu/temp/$FILEBASENAME.out" />
                <input URL="catalog:star.bnl.gov?collision=auau200,production=P02gc,filetype=daq_reco_mudst,runnumber&lt;4000000" nFiles="3" />
                <input URL="catalog:star.bnl.gov?collision=auau200,production=P02gd,filetype=daq_reco_mudst,runnumber&lt;4000000" nFiles="4" />

                <SandBox>
                        <Package>
                                <File>file:./numberOfEventsList.C</File>
                        </Package>
        </SandBox>

</job>
```

7.10 Using different fileListSyntax in job description

In SUMS there exists 3 implementations of fileListSyntax (paths, rootd, xrootd) described in *job element section*. Every syntax has different form of file URL and implies also different access to a file. In case of paths, all files are read locally by submitting the job to a specific node etc. See the *job element section.*

Example:

```
 <job maxFilesPerProcess="20" fileListSyntax="xrootd" filesPerHour="120"/>



        <command>
/star/u/starlib/ROOT/xrootd/bin/preStageFileList ${FILELIST}
root4star -q -b macroTest.C\(\"$FILELIST\"\)
        </command>

        <input URL="catalog:star.bnl.gov?collision=auau200,production=P02gd,filetype=daq_reco_mudst,runnumber&lt;4000000" nFiles="4" />
        <stdout URL="file:/star/data08/users/pjakl/pjakl_$JOBID.log" />
        <stderr URL="file:/star/data08/users/pjakl/pjakl_$JOBID.err" />

    <output fromScratch="*" toURL="file:/star/data08/users/pjakl/" />



        <SandBox>
                <Package>
                        <File>file:./macroTest.C</File>
                </Package>
        </SandBox>

 </job>
```