USING THE NJS AND TSI (V4)

Sven van den Berghe, ***Fujitsu Laboratories of Europe***

Version 4.0.2 25th April 2003

1 Contents

2 Gotchas

This is a list of problems, inconsistencies and configuration errors that you should watch out for (even if you do not read the rest of this)

- Review some IDB commands using the njs_admin test_commands command.

- The IDB entry COPY_CMD must incarnate to a command that follows symbolic links rather than copying them. Under Mac OS X this means using "cp –RL" and under Linux you should use "cp –rL".
- IDBs that use #DEFINE SH_CMD should place this **after** the lines #DEFINing commands that have this is a substring e.g. KSH_CMD, CSH_CMD
- TSI file permissions. Generally the permissions on the TSI files should be set to read only for the owner. The exception is `tsi_ls`, which **must** be readable for all Unicore users (the TSI script directory permissions must also be set so that all Unicore users can find it and find files in it).

  WARNING As the TSI is executed as root you should not leave any of these files (or the directories) writable after any update.
- njs.properties file. The file format allows comment lines but **not** inline comments.
- IDB Configuration. Remember to set the path to the "tsi_ls" file to where the TSI is stored.
- IDB setup, the INVOCATION of the RUN section should contain a path to the executable file e.g. `./<RUNCOMMAND>`

3 NJS

The NJS uses a number of files and directories during start up and processing. These files are:

- a configuration file (usually called njs.properties)
- a file to define resource and incarnation information (the IDB)
- certificates for SSL connections
- UUDB (mappings from public certificates to Xlogins)
- directory in which to place log files
- directory in which to save restart information

The NJS is delivered with some scripts to start the NJS, list log files and start the administration interface. These scripts assume a certain directory structure as follows:

```
njs/
    docs/
        NJS and TSI documentation
    conf/
        njs.properties
        IDB file
        logs/
    lib/
        ajo.jar
        njs.jar
        jsse.jar
        jnet.jar
        jcert.jar
    bin/
        scripts to control the NJS
```

"conf/" is the NJS configuration directory

"njs.properties" is the NJS configuration file.

"IDB" is the site-specific Incarnation Database

"logs/" is the directory that the NJS will use for the log files

"lib/" is the directory containing all the code used by the NJS (in the files listed) You may need to get the jsse.jar, jnet.jar and jcert.jar files separately (these are not required if you are using Java 1.4).

The NJS assumes that the files that it uses are stored in the configuration directory (njs/conf below) and all file paths given in the configuration files are interpreted as are relative to this directory.

3.1 Configuring

3.1.1 Security considerations

It is assumed that NJS is installed on a dedicated machine and that this machine is configured so that there are no other users apart from the user used to run the NJS (note that this should be an ordinary user and not root).

If these assumptions are not met, then the TSI connection should be carefully managed to prevent unauthorised user accessing the TSI connection. If normal user logins are allowed on the machine running the NJS, then the TSI shepherd should never be allowed to stay running if the NJS is not running. The NJS stop sequence should be:

```
njs_admin tsi stop
```

```
njs_admin stop_now
```

The TSI should then be restarted when the NJS is restarted.

The permissions on NJS configuration files should always be limited to allow only authorised users to change them.

3.1.2 Consigning sub-AJOs

NJSs can consign sub-AJOs to other NJSs by establishing SSL connections to their Gateways. In order to do this it needs to have a certificate acceptable to the remote Gaetway and other certificates to validate the credentials of the remote Gateways. The locations of these certificate files are supplied in the configuration parameters: **`njs.njs_cert_loc`** and **`njs.unicore_ca_loc`**.

To use SSL set the following line in the configuration file:

`njs.use_ssl=true`[1]

The NJS can use more than one certificate. If the first connection attempt to a Gateway fails when it uses one certificate, then the NJS will present each of the remaining certificates in turn until a connection is accepted.

The locations of the NJS's certificates is given by **`njs.njs_cert_loc`**, this is a list of file names (in the NJS configuration directory or the full path to a file) delimited by ":".

The NJS also requires the public certificates of the Certificate Authorities that issue Gateway certificates. The locations of these certificates is given by **`njs.unicore_ca_loc`**, which is a ":" delimited list of file names (in the NJS configuration directory or the full path to a file).

Note that as the NJS's certificates contain both the private and public keys an unlocking password has to be supplied every time the NJS is started. A password

[1] This is the default. If you do not want to have your NJS consign sub-AJO, then add the line `njs.use_ssl=false` to the NJS configuration file. This will stop the NJS reading the certificates. The NJS will execute local AJOs correctly but will fail any AJO that tries to execute sub-AJOs.

(the same one for all files) can be supplied either in the NJS configuration file or interactively using the **start_njs** script.

3.1.3 Gateway

The NJS receives UPL requests from a Gateway over sockets. It listens for connection requests from the Gateway on the socket number given by the `njs.properties` file entry **`njs.gateway_port`**. It will only accept connections from one Gateway machine; the name (or IP address) of this machine is given by the `njs.properties` file entry **`njs.gateway`**.

The sockets can be configured to use SSL by setting the `njs.properties` file entry **`njs.gateway_ssl`** to `true,` otherwise they will be ordinary sockets.

4 Logging

There are 6 levels of logging. In order of severity they are:

Severe: serious problem. Execution cannot or should not continue.

Warning: there is a problem or unexpected condition. Processing can continue - usually correctly, but occasionally may result in other problems (e.g. a client presented an invalid certificate or there was an error during the execution of an AbstractAction.)

Information: useful information, not necessarily related to any problems
Configuration: messages relating to the NJS configuration, reading of values, files and their interpretation.

Talk: verbose output,. May be useful to trace execution or gain hints about precursors to problems.

Debug: very verbose output.

If the logging level is set to a particular severity then messages of that severity and higher will be logged. Messages of lower severity will not be logged.

The default logging level is Configuration (which will log C, I, W and S messages).

The NJS generates structured names for the log files that will give the same order when sorted alphabetically as when sorted by time. The “list_log_files” script exploits this fact.

Lines of the log file are also structured. The first word as the time of creation of the message, the second word the date, the third word notes the severity level of the message with important messages (severe or warning) ending in a “*”, the rest are terminated by a “:”.

Log lines about AbstractActions always continue with some information about the AbstractAction, including its name, type, identifier and the identifier of its root AJO.

The logging level and changing of log files can be controlled through the configuration values (**njs.logging_level** and **njs.log_file_change_interval**) and through the Administration interface (njs_admin **logging** subcommands).

5 Scripts

The NJS distribution includes scripts to start the NJS and to access the log files[2].

These scripts require an NJS configuration directory which is determined using the following rules:

- Check the arguments to the script for the directory

[2] A further script is supplied to access the Administration interface, this operates in a different way which is detailed in a section below.

- Use the value of the environment variable “UNICORE_NJS”
- Use the local directory.

Note that the scripts also assume that certain files are named as described in Section 2. The best way to use files with names different to these is to supply a link e.g.

```
ln –s ajo_3.5.1-build1.jar ajo.jar
```

The scripts also assume that the names of the log files are the ones created by the NJS.

The scripts check to see if an instance of the NJS is running. The test for a running NJS is based on the LAST_PID file in the configuration directory that, if it exists, contains the process id of the last started NJS. If there is no process with this pid or there is no LAST_PID file, then the NJS is assumed not to be running.

5.1 The Scripts

5.1.1 list_log_files

```
list_log_files type configuration_directory
```

Return the names of some or all of the log files in the default logging directory (“configuration_directory”/logs).

If the type is “a”, then the names of all the log files are returned.

If type is “l”, then the names of the log files created since the Gateway was last started are returned (“L” returns the complement of this, the names of all the log files *except* those created since the last start).

Otherwise, type must be an integer, say n. If n is postive, then the names of the latest n files are returned e.g.

```
list_log_files 1
```

Returns the name of the current log file.

If n is negative the complement is returned e.g.

```
list_log_files –1
```

Returns the names of all except the current log file.

`configuration_directory` is optional.

5.1.2 start_njs

```
start_njs configuration_directory
```

Start the NJS (if not running).

This will start the Gateway and execute it as a background process (“nohup”ed) with all output redirected to a logging file.

`configuration_directory` is optional.

If there is no password in the njs.properties file, then **start_njs** will prompt for a password (and write it to !configuration_directory”/password for the NJS to read, and delete).

5.2 Example uses of the scripts

These examples assume that UNICORE_NJS has been set to the NJS’ configuration directory.

View the current log file:

```
cat `list_log_files 1` | more
```

Purge the log directory, leaving the files since last started (note the capital “L”):

```
rm `list_log_files L`
```

Implement a site specific policy for log file cycling:

- Set the configuration parameter "njs.log_file_change_interval" to some large value (e.g. 9999) to prevent the NJS changing log files at the wrong time.
- Create a cron job as follows:

```
/usr/unicore/my_njs/bin/njs_admin logging new_file

mv `/usr/unicore/my_njs/bin/list_log_files 1 \
/usr/unicore/my_njs/conf`  our_name
```

(This will of course invalidate the list_log_files script.)

Check for any possible problems since the last start (Severe or Warning messages):

```
cat `list_log_files 1` | grep " .\*"
```

Review the Gateway's processing of its configuration

```
cat `list_log_files 1` | grep " C: "
```

6 NJS configuration file

The NJS configuration file defaults to a file named "njs.properties" in the NJS configuration directory.[3]

Lines starting with "#" or "!" are ignored.

Entries are single line, but "\" can be used to continue over a line.

The remainder of a line following the "=" is treated as the value.

Inline comments are **not** allowed.

This section describes the meaning of each entry in the NJS configuration file giving its default value after the name.

njs.operation_mode=*full*

If the NJS is operating in "full" mode, then all AbstractActions are processed.

If the NJS is operating in "broker" mode, then only those AbstractActions relating to brokering (CheckQoS, CheckResources, getResources), flow control (ForGroup, If etc) and control (ControlAction, CopyJob etc) are processed. All other actions will fail.

A brokering NJS does not need a TSI.

Only the GENERAL section of the IDB is read and recognised by a broker mode NJS.

A UUDB is still required by a broker mode NJS.

Note that an NJS operating in full mode can still run a broker.

A broker mode NJS reports its type as BROKER when registering with a Gateway. This means that it will not appear in the ListVistesReply (which only contains full NJSs) but will appear in a ListPorts reply.

njs.vsite_name=*no name for this Vsite*

The name of the Vsite that this NJS provides services for.

njs.incarnationdb=*incarnationdb*

The location of the IDB file. This is either the full path to a file (starts with a /) or the name of a file in the NJS configuration directory.

njs.gateway_port=*8181*

[3] This can be changed by editing the start_njs script and changing the second argument to the NJS process

The port number on which the NJS listens for connect requests from the Gateway.

njs.gateway=*(no default)*

If the NJS is *not* registering with Gateways, then this is the name of the Gateway machine (name only, no port number) e.g.

```
njs.gateway=arcon.fle.fujitsu.com
```

If the NJS is registering with Gateways, then this is a *comma* separated list of machines:port_numbers that run Gateways. The NJS will only accept Gateway connections from machines in this list e.g.

```
njs.gateway=arcon.fle.fujitsu.com:4433
```

njs.redirecting_gateway=*(no default)*

A *comma* separated list of machine:port_number that run Gateways. The NJS will register with these Gateways but will not accept connections from them (the targeted Gateway will list the address of the first entry in the njs.gateway list).

njs.gw_registration=*false*

If this is true, then the NJS will maintain a registration with all the Gateways lists in the **njs.gateway** list.

njs.gateway_ssl=*false*

The type of sockets that the Gateway will use to connect to the NJS. If this is true, then SSL will be used, otherwise ordinary sockets are used.

njs.use_ssl=*true*

If the value is `true`, then the NJS will initialise itself to use SSL (and so be able to consign sub-AJOS to other NJSs). Otherwise, SSL will not be initialised and the execution of sub-AJOs will fail (all other NJS processing will proceed nornally).

Initialising SSL means that the entries njs.njs_cert_loc, njs.unicore_ca_loc and njs.ssl_password will be read. Uninitialised SSL means that these entries will not be read.

njs.njs_cert_loc=*njs_cert.p12*

The locations of the SSL certificates for the NJS. This is a ":" delimited list of file names. Each file name is either the full path to a file (starts with a /) or the name of a file in the NJS configuration directory.

njs.unicore_ca_loc=*njs_ca.pem*

The locations of the public certificates of the Certificate Authority that issues Gateway certificates. The NJS will only be able to contact Gateways whose SSL certificate has been issued by this CA.

This is a ":" delimited list of file names. Each file name is either the full path to a file (starts with a /) or the name of a file in the NJS configuration directory.

njs.ssl_password=*none*

The password that the NJS uses to unlock its certificate(s) (file names are listed in njs.njs_cert_loc).

If this is not the default value, then it is used to unlock the certificate. If it is not supplied (or is the default value), then the password is read from the file `config_dir/password` (this file is deleted once the NJS has read it).

Note that the same password is used for all the NJS certificates.

njs.memory=*remember*

Whether or not the NJS saves state. The default ("remember") is for the NJS to save state so that if the NJS is stopped normally it can be restarted without losing information about executing jobs. If this is set to "forget", then the NJS will not save state.

For backwards compatibility the line:

```
njs.memory=olorin.utils.MemoryImpl
```

has the same effect as "remember".

njs.save_completed_ajos=*false*

If this is set to true, then the NJS will save the AJO, Outcome object and user information of all "interesting" AJOs after the client has deleted the AJO from the NJS.

The NJS will **not** save any files created by the AJO's execution (including stdout and stderr).

This value can be changed at run time through the **archive** administration command.

njs.save_dir=*save*

The directory that the NJS will use to save its state. This directory is also used to save completed AJOs and is the location of the preserved AJOs.

If this is not a full path name, then it is interpreted as relative to the configuration directory.

njs.admin_type=*default*

The type of interface to use for the administrator interface.

This is not yet used

njs.admin_port=*7272*

The port number that the NJS uses to listen for contacts from administrators.

njs.log_file_change_interval=*24*

Sets how frequently the NJS will change log files.

If this is a number, then the log file will be changed every **log_file_change_interval** hours.

If this starts with the letter "h", then a new log file will be opened every hour on the hour.

If this starts with a "d", then a new log file will be opened every day on the day change (midnight).

This value can be changed at run time through the **logging** administration command.

njs.logging_level=*C*

Sets the level of logging.

Acceptable values are `S(evere), W(arning), I(nformation), C(onfiguration), T(alk) and D(ebug).`

This value can be changed at run time through the **logging** administration command.

njs.tsi_worker_limit=*5*

The NJS will ask the TSI daemon to create TSI a process when it detects the need for one. TSI processes are cached by the NJS and reused. This parameter sets the maximum number of processes that the NJS is allowed to use. The actual number used at any time will depend on the NJS load - the value of the

"threads.Incarnations" parameter and the number of "interactive" AJOs running.

The NJS will always create at least two TSI workers.

njs.tsi_update_interval=*5000*

Sets how frequently the NJS asks the TSI for an update to the status of jobs executing on the Batch sub-system (in milliseconds). The NJS makes this request only if there are jobs executing on the BSS. This should be set to a balance between the acceptable load on the TSI and BSS to update the information and how accurate the NJS reports of status should be. The default value favours accurate NJS information (at most five seconds out of date).

njs.checker_class=*com.fujitsu.arcon.njs.priest.CheckerImpl*

The name of the class to load to check the values in instances of org.unicore.ajo.ExecuteTask. This must implement the com.fujitsu.arcon.njs.interfaces.ExecuteTaskChecker interface.

threads.Incarnations=*3*

The number of threads to allocate to the execution of AbstractActions that are executed by the TSI. This must be at least the same as the number of TSIs that will be started on the target system and can be more.

uudb.class_name=*com.fujitsu.arcon.njs.uudb.UUDBImpl*

The name of the class to load to perform the UUDB functions. This class must extend the com.fujitsu.arcon.njs.interfaces.UUDB class.

uudb.directory=./

For the default UUDB class, this is the directory in which the UUDB has been created (file name is "UUDB").

uudb.check_signers=*false*

For the default UUDB class changing this value to "true" means that the UUDB will accept certificates if their **signer** has been entered into the UUDB (in addition to accepting them if the certificate is in the UUDB). All certificates signed by a signer will be incarnated with the Xlogin that is in the UUDB for that certificate.

This option can be used if you wish to accept Globus proxy certificates (first generation only) by entering the user's Globus certificate into the UUDB.

aft.factories=*none*

A colon separated list of class names that implement the com.fecit.arcon.njs.interfaces.AFT.AFTFactory interface. These factories will be used to transfer files to different Vsites instead of standard UPL streamed files route.

The NJS contains a sample implementation of this interface that uses the "rcp" protocol to transfer files. To use this implementation set alt.factories to "com.fecit.arcon.utils.AFTGateway" and initialise the factory using the following properties:

rcp_aft.port=*none*

A port used by the local AFT code to listen for requests from remote peers (through the local Gateway) for file and user name mappings.

rcp_aft.tsi_location=*none*

The name of the machine running the local TSI

rcp_aft.peers=*none*

Entries separated by commas with the following format: vsite@machine:port. Where *vsite* is the name of a remote Vsite that will accept AFT via rcp,

*machine*is the name of the Gateway for the remote NJS and *port* is the remote Gateway port.

njs.sso_type=*none*

If the parameter vale is “globus”, then the NJS will write the contents of the SSO field of any incoming root AJOs to a file named “.proxy” in the root directory of the Uspace created for the AJO. The SSO is deleted from the AJO once it has been written (for security). The NJS also sets its type to “Globus”,

If the parameter vale is “gridftp_proxy”, then the NJS will write the contents of the SSO field of any incoming root AJOs to a file named “.proxy” in the root directory of the Uspace created for the AJO. The SSO is deleted from the AJO once it has been written (for security).

If this is omitted, then the NJS will ignore the AJO’s SSO objects.

njs.dynamic_resource_file=*none*

The name of a file whose contents will be added to the resources advertised by the NJS. The NJS polls this file and updates the resource set contents as the file contents change.

This is a simple initial implementation to test how useful this concept is. The only resource type updated is “org.unicore.unicore.TextInfoResoutce”. The file’s contents are in the Java “Properties” format i.e. :

```
tag = value
```

For example:

```
cpu_load=75%

message=Routine maintenance at 14:00 UTC Today
```

6.1.1 Other files

The NJS may read other files:

`conf_dir/password.` This is one of the ways that the NJS can read a password. It looks for this if starting SSL and there is no `njs.ssl_password` entry in the `njs.properties` file.

7 NJS Administration

7.1 Starting the NJS

The NJS can be started using the supplied script (**bin/start_njs**).

The NJS will write most messages to a log file in the logging directory. However, if the NJS has problems before the logging is started messages will be appended to a file named **startup.log** in the logging directory.

7.2 Stopping the NJS

The NJS can be stopped through the administrator interface (see below for the commands to use).

If the NJS is brought to a controlled stop using the administrator interface, then it will save the state of all "interesting" AJOs that are on the NJS and when restarted it will restore the state of the "interesting" AJOs.

The NJS keeps a record of significant changes of states and so can restore the state of executing AJOs even if it is not brought to a controlled stop (e.g. if it crashes). However this way of restarting does lose some information, mainly the Outcome log (it should not lose any files and should "reconnect" with jobs executing on the BSS). Furthermore it should be noted that it is possible that the NJS stopped this way was in an inconsistent state and so some AJOs may not reload.

The NJS is a multithreaded application and the pausing and state saving runs in a thread of its own. This means that it may be possible to save the state of an apparently hung NJS (so before killing a hung NJS process it is worth while trying to save a consistent state using "njs_admin stop now").

Stopping the NJS does not stop the execution of tasks that are running on the BSS.

7.2.1 Pause the NJS first

The NJS is a multithreaded event-driven application simultaneously interacting with a number of applications (multiple clients, multiple TSIs and multiple client NJSs). It has no direct control over the behaviour of these other applications and so can only be restored to a state if the saved state is consistent.

The NJS is brought into a consistent state by pausing it. Pausing the NJS has two effects; it prevents any new interaction being started and it signals all existing external activity to stop execution *when a consistent point has been reached*. This means that some execution can continue for some time after a pause command.

The NJS should be stopped only when pause says that the NJS is fully paused.

The default stop command will wait until the NJS is paused before stopping. This can take a while if the NJS is involved in, for example, a long file transfer[4].

Alternatively, the "stop now" command will stop the NJS giving a short grace period for execution to stop. However, you should bear in mind that it is possible that some AJOs will not reload properly after such a stop.

The safest way to stop the NJS is to use the interactive option on the njs_admin command and issue pause command(s) until the NJS is properly paused and then stop the NJS.

7.2.2 Interesting AJOs

The NJS categorises AJOs into two types: interesting and uninteresting.

[4] Note that if no TSI is connected the stop command may hang until a TSI connects.

Interesting AJOS are those that contain tasks that do things on the main system, such as execute a job and manipulate files. Other interesting AJOs are those that apply controls to other AbstractActions (e.g hold, cancel)

Uninteresting AJOs are all other AJOs. These are "service" AJOs that can easily be recreated and reconsigned by clients. The results of "uninteresting" AJOs become irrelevant if the NJS is stopped and restarted some time later.

7.2.3 NJS State

The NJS will save its state into a number of files in the directory given by the `njs.save_dir` configuration option.

Each "interesting" AJO is saved to a different file. These files have different extensions depending on their state.

Files with an "sajo" extension contain the state of AJOs that have been saved by the NJS while they are still executing. These will be read at the next NJS restart and deleted.

Files with a "dajo" extension contain the state of AJOs that have finished their execution but not yet deleted by the client. These files will be read at the next NJS restart, deleted and immediately rewritten[5].

Files with a "kajo" extension contain the state of AJOs that have finished execution and been deleted by the client (are completed). These are created if the appropriate value of the `njs.save_completed_ajos` configuration option is set. These files are not read at restart.

The NJS may also create other files in this directory that contain further state information.

Note that part of the state of finished and execution AJOs is also kept on the execution system(s) - in the Uspace, Outcome and Spool directories, as well as in any jobs executing on the BSS.

Deleting an AJO from the NJS removes all files from the execution system (Uspace, outcome directory). "kajo" files do not contain these files.

It is possible to change some of the NJS configuration and have the NJS restart correctly. If the root directories of the Uspace, Outcome or Spool areas are changed, then any existing files must be moved to the new areas before restarting. Note also that on restart the AbstractActions are matched to the new resources offered by TSIs, if there is a big change in resources it may be possible that some AbstractAction fail or that incarnated BSS jobs cannot be found

7.3 Executing the njs_admin command

The njs_admin command can be run in two modes, as a single command or interactively. All commands except for `pause` will work in both modes. Pause can only be used in interactive mode.

To execute a single command the required command is supplied as the arguments to njs_admin e.g.

```
njs_admin stop now
```

If there are no arguments, then njs_admin starts interactively (if authenticated by the NJS).

The following optional arguments are interpreted by the njs_admin script and not passed to the NJS:

-h prints a help message about the admin command

-v prints version information

[5] So that the state information is encapsulated in instances of the most up to date Java classes.

The following optional arguments are interpreted by the njs_admin script and not passed to the NJS (and so if only these appear on the command line njs_admin will start interactively):

-m <name> contacts the NJS on the named machine

-p <number> contacts the NJS using the given port number (the NJS is listening on the port number given in the **njs.admin_port** comfiguration value).

These values are optional and the njs_admin script itself can be edited to set local values for them.

7.3.1 Authentication

The NJS authenticates administrators by asking the script to write a file in the NJS's configuration directory. This means that the NJS administrators are all the users that can write to the configuration directory[6]. This also means that administrators can contact the NJS from remote workstations only if the configuration directory is on a shared file system.

IMPORTANT

The NJS administration command set contains commands that can delete user's jobs and delete user's Outcome files. It is essential that the permissions and/or access control on the NJS configuration directory is set to allow only those users who are Unicore Administrators write permission.

IMPORTANT

7.3.2 Excluding administration commands

It is possible to remove an administration command from the NJS by adding lines to the configuration file e.g.

To prevent the NJS from installing the **cancel** command add the line:

```
admin.no_cancel
```

7.4 NJS administration commands

All admin commands accept **help** and will print a summary of their options.

debug_scripts [on|off]

Toggles the production of verbose (debug) output by the NJS wrapper scripts (setting of –vx flags)

test_commands xlogin

Run some simple tests to check that the NJS built in Unix commands are properly initialised in the IDB (e.g. that the paths are correct). These tests are not exhaustive.

You should study the output of this command to see if the tests work as expected.

Successful completion of these test does not mean that the commands will always work correctly.

xlogin is a valid user that will be used to execute the commands.

gw_registrar [list|delete|update|add]

Administer NJS registrations with Gateways.

This command is only available if the **njs.gw_registration** configuration value is set to **true**.

---

[6] At some time an alternative administrator interface may be developed that uses a Unicore PKI and the UUDB to authenticate administrators.

In the commands below *gateway_uri* stands for a Gateway machine address, plus port number as below:

```
arcon.fle.fujitsu.com:4433
```

list

List the current registration with the time and status of the last registration attempt

delete *gateway_uri*

Delete the Gateway from the list of gateways to register with (this does not cancel a current registration but will not renew it so in time, at most 30 minutes, the Gateway will stop contacting and listing the NJS)

update *gateway_uri*

Register with the Gateway (e.g. if the Gateway has been restarted this will immediately reregister rather than wait for the normal update – up to 15 minutes)

add *getway_uri*

Add the Gateway to the list of registrations.

pause_njs [number]

Pause the execution of the NJS, waiting **number** seconds before returning.

If all the components of the NJS are in a consistent save state this command returns with:

```
NJS paused
```

Note that this command can only be executed in interactive mode.

resume_njs

Resume the execution of a paused NJS

stop [nosave] [now]

Stop the NJS executing waiting until the NJS is in a consistent state.

The **now** option will force the NJS to stop after a short time even if some parts are not definitely in a consistent state.

If the **nosave** option is used, then the NJS stops without saving state.

sync [now]

Write the current state of executing interesting jobs to disk and continue executing.

The **now** option will force the NJS to sync after a short time even if some parts are not definitely in a consistent state.

status

List the status of the components of the NJS that can be paused

help

Print help about the commands recognised by the NJS.

Note that each of the commands will print detailed help if the help option is selected e.g.

```
stop help
```

quit

Stops execution of the njs_admin command.

Note that you can not quit if the last command was a pause, you must either resume or stop the NJS before quitting.

list [short|detailed|long] [*selection*]

List information about the selected AbstractActions at the selected level (default is `short`)

The default ***selection*** is all AJOs on the NJS.

If ***selection*** is “ajos” all the AJOs on the NJS are also listed.

If ***selection*** is “all”, then all actions on the NJS are selected.

If ***selection*** can be interpreted as a number (hexadecimal), then the action with that Identifier is selected.

Otherwise, ***selection*** is treated as an expression.

Expressions can combine selections using an and operator (e.g. &) or an or operator (|).

Expressions can be grouped using brackets.

The expression terms are:

status *value*

Selects all actions on the NJS that have the status ***value***. The valid values are derived from the AJO class org.unicore.outcome.AbstractActionStatus. The strings are:

CONSIGNED, DONE, EXECUTING, FAILED_IN_C, FAILED_IN_E, FAILED_IN_I, HELD, KILLED, NEVER_RUN, NEVER_TAKEN, NOT_DONE, NOT_SUCCESSFUL, PENDING, QUEUED, READY, RUNNING, SUCCESSFUL, SUSPENDED.

For example, to produce a short listing for all actions that are currently executioning:

```
list status executing
```

type *value*

Selects all actions on the NJS are of the type ***value***. The valid values are the classes in the AJO class structure. The Strings are printed in the listings.

For example, to produce a long listing of all imports that failed in execution:

```
list long (type IMPORT & status FAILED_IN_E)
```

rootajo *value*

Selects all actions that have the AJO identified by ***value*** as their root AJO.

For example to produce a short listing of all pending actions from the AJO with identifier AA45FG:

```
list status PENDING & rootajo AA45FG
```

user *xlogin*

Selects all actions on the NJS that are executed by the *xlogin* (local user name).

For example to list all root actions that will execute as user ZZFG or XXDE:

```
list (user ZZFG | user XXDE)
```

ulogin *value*

Selects all actions on the NJS that were consigned to the NJS by a Unicore user whose common name (CN) contains the string *value*.

For example to list all root AJOs that were consigned by a user with "Fred" in the common name and that will execute either as ZZFG or as XXDE:

```
list ( ulogin Fred & (user ZZFG | user XXDE))
```

bssid *value*

Selects the action on the NJS (if any) that is incarnated and running with this BSS identifier.

Example:

```
list bssid 9987
```

abort ***selection***

Abort execution of the selected actions on the NJS.

selection is described in the list section

cancel ***selection***

Cancel the execution of the selected actions on the NJS.

This aborts execution and removes any outcomes. Cancelled AJOs are deleted from the NJS and their Uspace and Outcome directories are deleted.

selection is described in the list section.

hold [*selection*]

Hold execution of the selected actions on the NJS.

selection is described in the list section.

resume [*selection*]

Resume execution of the selected actions, if held and whether held by an admin command or by a user command.

selection is described in the list section.

tsi [up] [down] [status] [stop] [refresh]

Controls the TSI and the NJS view of the TSI status.

The **down** option tells the NJS that the TSI is not available for use. The NJS will not try to send any commands to any existing TSI workers and it will not try to create any new TSI workers. Most AbstractActions that require a TSI connection will be HELD until the TSI becomes avaulable (**up** command) but some attempts to use the TSI will result in failures (e.g. new jobs that have data sent with them)

The **up** option tells the NJS that the TSI is available for use, any AbstractActions that were held because the NJS was not available (down) will be RESUMEd

The **stop** option causes the NJS to send a command to the TSI shepherd to stop it executing. The TSI is marked as unavailable (just like the **down** option). The NJS will only start using a TSI that is restarted after a **stop** if it is told that the TSI us available **up** option).

Note that the NJS does not automatically detect that the TSI is unavailable or that it has become available, its view of the TSI is determined by the **up, down** or **stop** options. If the NJS regards the TSI as available and it is not, then the NJS will try to use the TSI and generate errors.

The **refresh** option causes the NJS to stop all current TSI workers but leaves the TSI shepherd running. The TSI is still available and when necessary the NJS will create new workers.

The **status** option lists the current TSI status (according to the NJS)

ls [outcomes|uspaces|spool] [xlogin]

List the names of the directories used to hold Outcomes, Uspaces or spooled files with information about the owning AJO if it can be found on the NJS.

The listing is obtained with the supplied xlogin (if none is supplied, then the value in the IDB for QSTAT_XLOGIN is used)[7].

Uspaces are tagged as "orphaned" if their owning AJO is not on the NJS or is on the NJS and has completed execution (Uspaces should be removed as soon as an AJO has completed execution).

Outcomes are tagged as "orphaned" if their owning AJO cannot be found on the NJS.

Spool directories are tagged as "orphanned"if the NJS cannot find information about which Unicore User created them.

remove [outcome|uspace|spool] identifier xlogin [project]

Delete the Upsace, Outcome or Spool directory owned by the identified AJO (or Portfolio in the case of Spool directories). The xlogin must be supplied and match the owner of the actual directories[8].

Uspaces and Outcomes can only be removed if they are tagged as orphaned by the **ls** command (otherwise the owning AJO should be **cancel**led).

Spool directories can always be removed, care should be taken that the person who spooled the files has really finished with them.

archive [on|off]

Turn saving of the completed AJOs on or off (see also the configuration value **njs.save_completed_ajos**).

logging [new_file|level|interval|info]

Control NJS logging.

new_file Close the current log file and open a new one

level ***new_level [area]*** Set a new logging level

Values for *new_level* are S, W, I, C, T. D (see also the configuration parameter njs.logging_level).

area is optional. If it is not given, then all NJS logging is changed to the new level. Otherwise, the logging is changed only for the given area of the NJS. Recognised areas are: "g" for general NJS functions, "a" for the execution of AbstractActions, "r" for accepting new AJOS and consigning sub-AJOs, "u" for user authorisation and "t" for TSI interaction.

interval Set a new interval for changing the log file (see also the configuration value **njs.log_file_change_interval**) The value following this is interpreted as follows

If this is a number (n), then the log file will be changed every **n** hours.

If this starts with the letter "h", then a new log file will be opened every hour on the hour.

---

[7] The incarnation will substitute $USER and $HOME in the Uspace and Outcome root directories and so "xlogin" is required if these values appear in the IDB definitions of Uspace or Outcome root directories.

[8] Remove uses the IDB Incarnation value for the CLEANUP task. If this is set to keep the directories rather than remove them, then the **remove** command cannot delete them. Furthermore, the standard incarnation is **rm –rf** which means that no errors will be reported from the execution of this command.

If this starts with a “d”, then a new log file will be opened every day on the day change (midnight).

info Summarises current logging parameters

clear_fifo identifier user_id [project]

Sends an EOF to all fifos in the Outcome of the AJO nominated by **identifier**. The **user_id** and (optional) **project** are used to incarnate the script to write the EOF.

This command is intended to clear any fetches of Outcomes by client applications that are blocked because they have accidentally read an open fifo that has no application writing data to it.

7.5 Remote Administration through AJOs

NJS administration can also be performed using AbstractJobs. If an AbstractJob contains an instance if the class com.fujitsu.arcon.nsj.interfaces.AdministrationService, then the command string is executed as if it was received through the `njs_admin` command described above (the raw response from the NJS is returned in the AJO’s Outcome).

7.5.1 Authorising users to execute administration AbstractActions

Users must be authorised to execute administration commands. Normal users cannot execute the administration AbstractAction.

A user is authorised to execute administration AbstractActions if she is entered with the ADMINISTRATOR role in the UUDB[9].

7.5.2 Allowing admin commands

Each admin command must be explicitly allowed for remote administration. Commands that are not listed as allowed cannot be executed using AbstractActions[10].

To allow a command add the following line to the NJS configuration file (njs.properties):

`remote_admin.allow_<command>`

Where <command> is the name of the command that is allowed e.g.

`remote_admin.allow_list`

Allows listing of AbstractActions on the NJS.

The selection of commands to allow will be determined by the site’s policy for remote administration but it is not recommended that any of `stop`, `pause_njs` or `resume_njs` are ever allowed for AJO execution.

[9] To add a user with the ADMINISTRATOR role into the FLE UUDB issue the command “add admin-<xlogin>” where xlogin is as for normal users.

[10] There is a mechanism to specify the commands that a local administrator can use (see Section 7.3.2) but this *excludes* commands whereas AJO administration commands must be *included*.

8 TSI administration

The TSI `requires` Perl 5.004 as pervious versions of Perl have security vulnerability. If you use a previous version of Perl, then you should read and change the first few lines of the "tsi" file.

The TSI now uses an auxiliary script to list directories. This is supplied with the TSI code and the IDB should be changed so that incarnations of ListDirectory use the "tsi_ls" file.

8.1 TSI script file permissions

The permissions on the TSI script files should be set to read only for the owner. As the TSI is executed as root you should not leave any of these files (or the directories) writable after any update.

The execption is `tsi_ls` which **must** be world readable (the directory permissions must also be set so to world readable).

8.2 Execution model

The TSI has two modes of execution. The first process to be started is the "TSI shepherd" which will respond to NJS requests and start up "TSI workers" to do the work for the NJS.

Since the TSI runs with setuid permissions it must authenticate the source of commands as a genuine NJS. To do this the TSI is initialised with the address of the machine that runs the NJS together with a port number that the NJS will listen on for TSI worker connections. The TSI shepherd will only accept requests from the NJS machine and all TSI processes will establish connections to the NJS machine/port.

If the NJS process dies any TSI workers that are connected to the NJS will also die. However, the TSI shepherd will continue executing and will supply new TSI processes when the NJS is restarted. Therefore, it is no longer necessary to restart the TSI daemon when restarting the NJS.

If a TSI worker stops execution the NJS will request a new one to replace it.

If the TSI shepherd stops execution, then all TSI processes will also be killed. The TSI shepherd must then be restarted, this does not happen automatically.

8.3 Configuring

The TSI can be configured by editing the TSI files. The configuration has been concentrated into the "tsi" file and the part of this file that should be changed is clearly marked.

Changes outside this part should not be necessary, but if they are made they should be passed on to Fujitsu Labs or Pallas so that they can be incorporated into future releases of the scripts.

The necessary changes can be different for different systems and so you should read the first part of your TSI script where the required changes are marked and commented.

8.3.1 Directories used by the TSI

The TSI must have access to the directories specified in the IDB to hold Uspaces, Outcomes and Spooled files. These directories are written with the TSI's uid set to the xlogin for which the work is being performed and so must be writable by all xlogins[11].

[11] Note that the TSI no longer writes files to its current directory so that it is no longer necessary to start up the TSI in a particular directory.

8.4 Starting the TSI

8.4.1 Starting

The TSI can be started with or without command line arguments.

When executed with command line arguments the format is:

```
perl tsi njs_machine njs_port my_port
```

where the NJS is executing on `njs_machine` and is listening for TSI worker connections on `njs_port` (`njs_port` must match the first port number in the **`SOURCE`** entry of the **`EXECUTION_TSI`** section in the NJS's `IDB` file). A TSI process in shepherd mode will listen for NJS requests on `my_port` (`my_port` must match the second port number in the **`SOURCE`** entry of the **`EXECUTION_TSI`** section in the NJS's `IDB` file).

Alternatively, the TSI can be started without command line arguments. In this case the variables **`$main::njs_machine, $main::njs_port, $main::my_port`** must be set in the **tsi** Perl file for your system.

Depending on the shell used to start the TSI it may be necessary to execute these commands through `nohup` if you want to log out afterwards.

8.5 Stopping the TSI

The TSI can be stopped through the `njs_admin` command `tsi stop.`

All TSI workers can be stopped using the `njs_admin` command `tsi refresh` (the TSI will continue running and the NJS will make new workers as required).

TSI worker processes will stop executing when the NJS stops executing.

It is possible to kill a TSI worker process but this could result in the failure of an AJO (but the NJS will recover and create new TSI processes).

The TSI shepherd can be killed (preferably using SIGTERM). Since this results in the killing of all TSI processes this should only be done when the NJS has been stopped.

8.6 TSI logging

The TSI processes return any logging information to the NJS. The TSI daemon writes log information to stdout and stderr, to save these they should be redirected to a file.

9 APIs to the NJS

9.1 Executable environment

The NJS sets the following environment variable for all executables:

UC_NODES

The number of nodes requested by the job

UC_PROCESSORS

The number of processors requested by the job

USPACE_WD

The initial working directory of the job (this may be a subdirectory of the Upsace diirectory)

USPACE_HOME

The top level directory of the Uspace

UC_ITERATION_COUNTS

The current count of the iteration(s) of any org.unicore.ajo.RepeatGroup or org.unicore.ajo.ForGroup that are parents of the AbstractAction that was incarnated to become the executable.

The structure of the value is a_b_c_d where a, b, c, d etc are the current count of any nested RepeatGroup or ForGroup (d is the innermost loop). An executable within a single loop gets just the number - a – and no underscore.

Naming files as "file_name_$UC_ITERATION_COUNT" will give unique names for all iterations.

The following script extracts the iteration count for the innermost loop:

```
echo $UC_ITERATION_COUNTS | awk 'BEGIN {FS="_"} {print $NF}`
```

UC_DECISION_FILE

The file from which the NJS reads the executable's "Decision".

The "decision" is some text passed from an executable to the NJS and used to control execution of an AJO (through If and RepeatGroup AbstractActions). The NJS reads an executable's Decision from the file named in $UC_DECISION_FILE.

If an AbstractAction contains a resource "MakeReturnCodeDecision", then the NJS will insert code to return the executable's return code as the Decision.

A script can use UC_DECISION_FILE to write some other decision to the file (there must be no "MakeReturnCodeDecision" resource in this case).

9.2 Code APIs

The NJS has a number of APIs that allow installations to add or tailor NJS functions. These APIs are in the Java package "com.fujitsu.arcon.njs.interfaces" and JavaDoc for these interfaces and classes is part of the standard NJS distribution. This section provides a brief introduction to the interfaces.

9.2.1 Alternative File Transfer

Files are transferred between Vsite using the streamed data in the UPL. However, many Vsites have some special relationship that could allow much more efficient methods of file transfer to be used (they could be part of the same Usite etc). The NJS checks before each file transfer to see if there is an alternative route for the files before using the (default) UPL streaming.

Alternative File Transfer code can be created by implementing the `com.fujitsu.arcon.njs.interfaces.AFT` interfaces (these are loaded into the NJS using the **aft.factories** configuration value).

9.2.2 UUDB

The NJS can use site specific implementations of the user database by loading code that extends the `com.fujitsu.arcon.njs.interfaces.UUDB` class.

The NJS uses the class named in the configuration value **uudb.class_name**, which defaults to *com.fujitsu.arcon.njs.uudb.UUDBImpl*

9.2.3 Resource Brokering

The `com.fujitsu.arcon.njs.interfaces.ResourceBroker` interface is used to add brokering and quality of service capabilities to the NJS.

Resource brokering code is loaded if the NJS finds the BROKER keyword in the GENERAL section of the IDB.

If resource brokering code is loaded, then NJS will add instances of the ResourceCheck and QoSCheck resources to its list of advertised resources and so will execute jobs requesting these resources. If there is no resource brokering

code, then these resources will not be added and jobs requesting these will not be executed.

9.2.4 Argument checking

The `com.fujitsu.arcon.njs.interfaces.ExecuteTaskChecker` interface can be used to enforce policies on the arguments that a user can supply to executables (so that, for example, users cannot execute arbitrary code in "interactive" commands).

The NJS uses the class named in the configuration value **njs.checker_class,** which defaults to *com.fujitsu.arcon.njs.priest.CheckerImpl* (this does nothing).

10 Miscellenea

10.1 Trusting other NJSs

Generally other NJSs are not trusted to execute AJOs. They are allowed to consign and retrieve AJOs created and signed by Unicore users but nothing else. However, to run some advanced Unicore functions (e.g. Resource Brokering) NJSs need to be able to create and run AJOs on behalf of users (e.g. to make concrete requests about available resources). This can be allowed, but the trust must be established individually for each external NJS.

This method of allowing trust applies to the default UUDB shipped with the NJS only. Other implementations of the UUDB may or may not allow this, consult their documentation.

To trust another NJS you need to add its public certificate to the UUDB using the "add_njs" command. An Xlogin is required by the command but never used.

Any attempt to execute code on the TSI by a trusted TSI is disallowed. The only AbstractActions that are allowed are those that execute within the NJS and consigning sub-AJOs.

10.2 Allow registration of NJSs

NJSs can register with Gateways. If a registration is accepted by a Gateway, then the Gateway will accepts jobs targeted at the NJS and will advertise the NJS's existence to clients.

Registration means that there is less configuration of the Gateways, since NJSs can supply much of the required information (e.g. AJO class version used by the NJS, NJS address). It also means that an NJS can be moved to different machines without having to stop the Gateways.

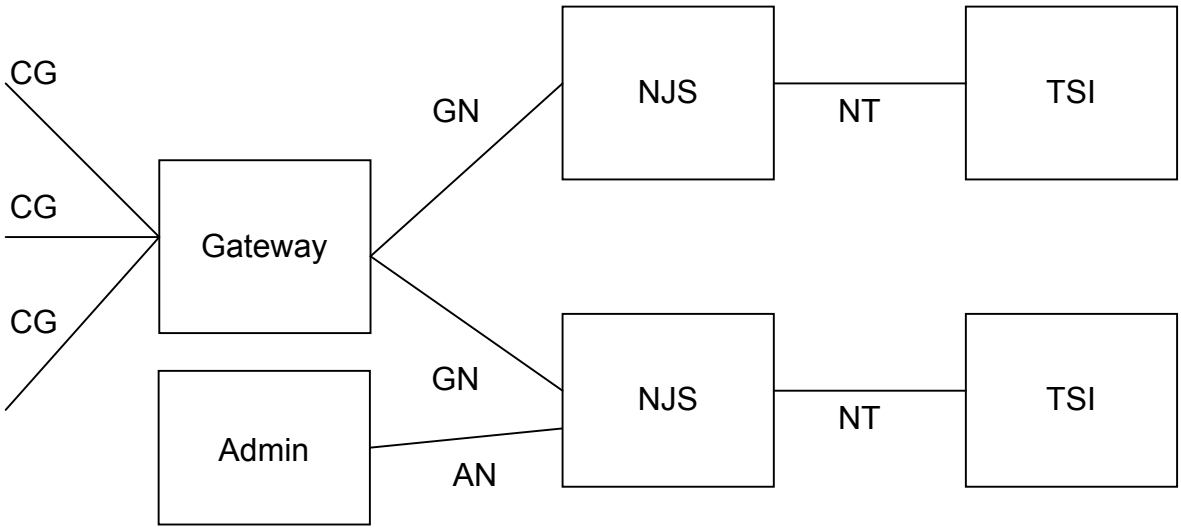
Registrations must be maintained by an NJS. If they are not renewed after a certain time the Gateway will deregister the NJS. An NJS will automatically maintain all registrations while it is running.

Registration is turned on by setting the **njs.gw_registration** configuration value to true. The Gateways contacted are those on the **njs.gateway** list.

Registrations can be checked and updated at run time by using the **gw_registrar** command of the NJS admin utility.

Gateways need to be configured to accept NJS registration.

11 Connections between Unicore processes[12]



There are four types of socket connections used by the Unicore servers.

11.1 CG - Client to Gateway

Clients (JPAs or NJSs) contact Gateways using SSL (the Gateway is the server)

The Gateway listens for clients on the port named in the `gateway.properties` file using the **`gw.port`** entry.

The way that Clients find which machine and port to use depends on their configuration methods. Client NJSs use the values given to them in the AJOs they are consigning.

11.2 GN - Gateway to NJS

Gateways contact NJSs over sockets, the NJS is the server.

The NJS listens for connections from the Gateway on the port named in the `njs.properties` file using the **`njs.gateway_port`** entry and will accept connections *only* from the machine listed in the **`njs.gateway`** entry.

The Gateway finds out which port (and machine) to contact a particular NJS on in the `connections` file or through its configuration parameters.

The Gateway and NJS must be configured to use the same type of socket, both use ordinary sockets by default.

If an SSL connection is required, then SSL must be selected on both (**njs.gateway_ssl** for the NJS and setting the SSL string in the appropriate Vsite definition line in the Gateway).

11.3 NT – NJS to TSI

NJSs talk to the TSI workers over plain sockets, the NJS is the server.

The TSI is told which machine and port to contact the NJS on either as command line arguments or as entry in the **tsi** file (**`$main::njs_machine`** and **`$main::njs_port`**).

The NJS listens for TSI connections on the first port named in the `IDB` using the **`SOURCE`** keyword of the **`TSI`** description section.

[12] Note that the Gateway can actually listen on multiple ports.

A TSI in shepherd mode listens on **$main::my_port**. The NJS contact a TSI shepherd on the **second** port named in the `IDB` using the **`SOURCE`** keyword of the **`TSI`** description section.

11.4 AN – Administrator to NJS

The **njs_admin** process contacts the NJS. The NJS is listening on the **njs.admin_port** defined in the njs.properties file. Edit the **njs_admin** script to set this value.

12 Setting up Alternative File Transfers

This section discusses how to set up the sample AFT implementation. Note that as this uses "rcp" as the alternative transfer mechanism it should only be installed when this route can be trusted. Furthermore, all rcp specific initialisation (.rhosts etc) must have been done.

12.1 Gateway

Set up a connection handler that will pass mapping requests from remote AFT peers through to the AFT code in the NJS by adding the following line to the Gateway "connections" file:

```
vsite=njs1_aft arcon 1204 class=com.fecit.arcon.gateway.AFTVsiteConnectionFactory
type=AFT
```

Where the local NJS - called "njs1" - is running on "arcon" and listening on port "1024" for AFT requests (not normal Unicore traffic).

Note that the convention for the NJS AFT code is that AFT services are provided through a Vsite whose name has "_aft" added to the end of the main Vsite's name e.g the main Vsite is defined as:

```
vsite=njs1 arcon 4444 type=NJS
```

12.2 NJS

Add the following lines to the "njs.properties" files:

```
aft.factories= com.fecit.arcon.utils.AFTGateway
rcp_aft.port=1024
rcp_aft.tsi_location=vpp5000
rcp_aft.peers=remote@somewhere_else.com:4433,\
              remote2@there.com:4433
```

Note that the rcp_aft.port entry matches the Gateway initialisation value and that the peer entries refer to Gateway machines (where the Gateway entries would be "remote" for the main NJS entry point and "remote_aft" for the AFT code entry point etc).

13 Change history

Version 4.0.2 Added remote administration commands (enabling), added clear_fifo admin command.

Version 4.0.1 Added njs.broker_only, debug_scripts

Version 1.7.2 Revised AFT entries (added setting up), added dynamic resources entry

Version 1.7.1 Added *test_commands* to njs_admin

Version 1.7 4.0.0 release, restructured a bit

Version 1.6 New TSI controls in njs_admin

Version 1.5 Handles Globus proxies in AJO SSOs (NJS configuration – njs.sso_type), uudb.check_signers entry