NorduGrid ARC 6 Documentation

Release ARC6

NorduGrid Collaboration

Nov 08, 2024

ARC OVERVIEW

The *Advanced Resource Connector (ARC)* middleware, developed by the NorduGrid Collaboration, is an open source software solution enabling e-Science computing infrastructures with emphasis on processing of large data volumes. ARC is being used to enable national and international e-infrastructures since its first release in 2002.

This document is dedicated to the ARC Version 6 collecting all relevant information in one place. You should be able to find information regarding the code, documentation, testing activities, support channels, … and so on here. The information is refreshed daily, a snapshot of the development version can be found here.

If you are new to ARC start reading the *Try ARC6* quickstart guide to get an overview of main operations in the simple test case.

For production *Computing Element* deployment follow the *Installation and Configuration Guide* that contains the structure and pointers to precise configuration of every ARC subsystem.

In case you are migrating to ARC 6 from an ARC 5 installation read the *Migration Guide*. For an overview of the main changes compared to ARC 5, please visit *Main changes in ARC 6 compared to ARC 5*.

The ultimate description of the new ARC 6 configuration can be found in the *ARC Configuration Reference Document*.

ARC OVERVIEW

Birds-eye overview of the ARC services, including the architecture figure of the ARC CE can be found in the following documents:

1.1 ARC CE components and the infrastructure ecosystem around

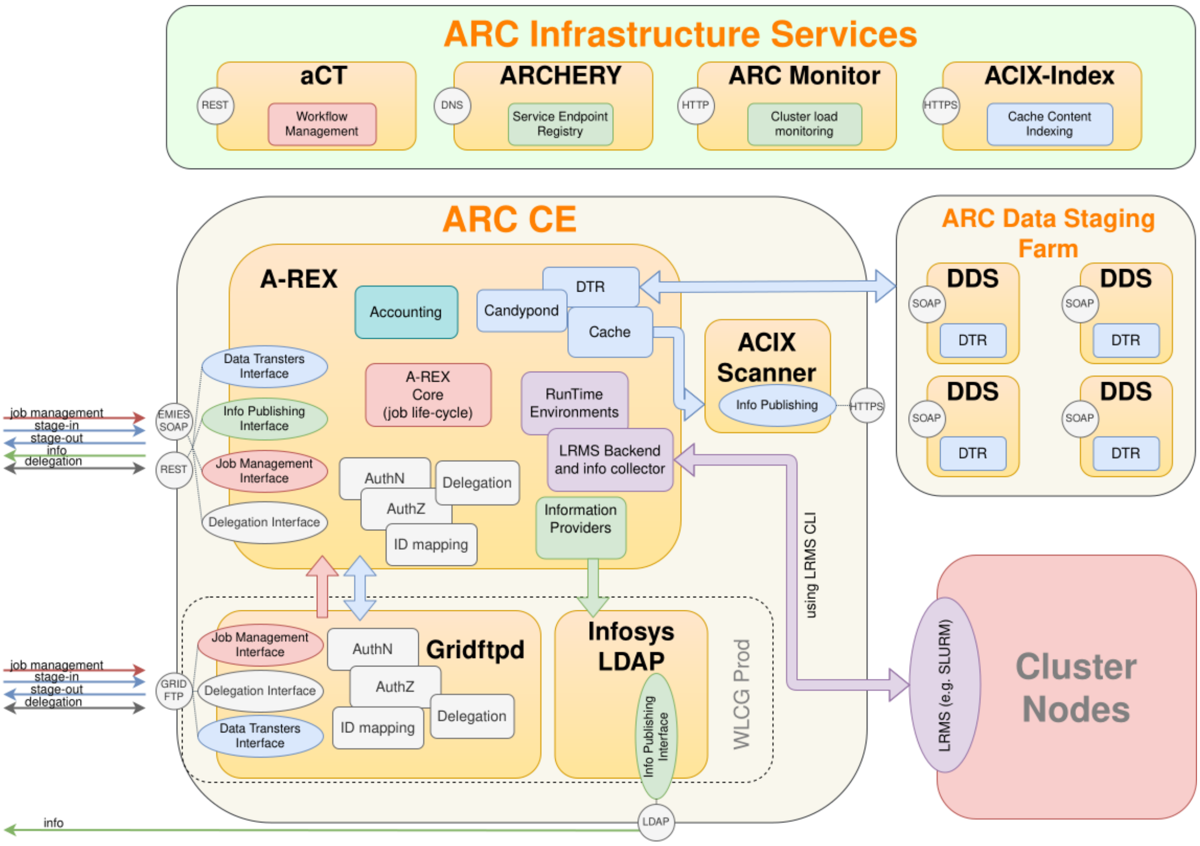


Fig. 1.1: ARC6 CE: internals, interfaces and the infrastructure ecosystem services around

CHAPTER

TWO

OBTAINING THE SOFTWARE

ARC is available for variety of GNU/Linux flavors via stable *Repositores* or *Nightly Builds* if you want to test the latest development release of ARC 6.

For the latest development of the not yet released ARC 7: Nightly Next Builds

The source code is hosted in NeIC's Coderefinery GitLab repository.

CHAPTER

THREE

SUPPORT AND COMMUNITY

User support and site installation assistance is provided via the nordugrid-discuss mailing list, and the Nordugrid Bugzilla.

CHAPTER

FOUR

DOCUMENTATION FOR INFRASTRUCTURE ADMINS

This section contains a documentation about all ARC middleware services deployment, configuration and operations. If you are looking for ARC Computing Element setup instruction or performance tuning parameters you are in the right place.

4.1 ARC Configuration Reference Document

4.1.1 General configuration structure

This is the `arc.conf` REFERENCE DOCUMENT defining the configuration blocks and configuration options for the ARC services.

The `arc.conf` configuration file consists of the following blocks:

```
[common]
[authgroup:groupname]
[mapping]
[authtokens]
[lrms]
[lrms/ssh]
[arex]
[arex/cache]
[arex/cache/cleaner]
[arex/data-staging]
[arex/ws]
[arex/ws/jobs]
[arex/ws/publicinfo]
[arex/ws/cache]
[arex/ws/candypond]
[arex/ws/argus]
[arex/jura]
[arex/jura/sgas:targetname]
[arex/jura/apel:targetname]
[arex/jura/archiving] (removed in 6.8.0)
[arex/ganglia]
[gridftpd]
[gridftpd/jobs]
[gridftpd/filedir]
[infosys]
[infosys/ldap]
[infosys/nordugrid]
[infosys/glue2]
[infosys/glue2/ldap]
[infosys/glue1]
```

(continues on next page)

(continued from previous page)

```
[infosys/glue1/site-bdii]
[infosys/cluster]
[queue:name]
[datadelivery-service]
[acix-scanner]
[acix-index]
[userlist:name]
[nordugridmap]
[custom:blockname]
```

[block]

A block configures an ARC service, a service interface, a utility or a subsystem. Enabling (turning on) a functionality, a service or an interface requires the presence of the appropriate configuration block. To disable a service or an interface, simply delete or comment out the related `arc.conf` block (you may need to rerun the corresponding startup script).

The `[common]` block is mandatory even if not a single option is specified within. The presence of the block turns on the default values for the configuration options within the block.

As an example, in order to set up a minimalistic ARC CE offering no external interfaces you need to configure at least the `[common]`, `[mapping]`, `[arex]`, `[lrms]`, `[infosys]` and `[queue:name]` blocks.

As another example, an ARC-based data offloader would require the `[common]` and the `[datadelivery-service]` blocks.

A block is identified by its block header. A block header may consist of keywords and optionally block identifiers. Keywords may be separated by `/` and used to label subblocks (e.g. `[arex/jura]`), while block identifiers are separated by `:` from keywords. For example, in the `[queue:short]` block header `queue` is a keyword while `short` is an identifier, e.g. the name of the queue. Block headers must be UNIQUE.

A block starts with a unique `[keyword:identifier]` blockheader and ends where the next block starts, that is at the next `[blockheader]` directive.

A block may have sub-blocks e.g. the various interfaces of the AREX service are configured via sub-blocks (e.g. `[arex/ws]`). When a sub-block is enabled then the corresponding parent block must also appear in the `arc.conf` file.

Configuration blocks contain (config option, config value) pairs following the syntax in single line:

```
config_option=value element [optional value element]
```

Note: quotes around the configuration value(s) must NOT be used any longer.

Note: the `arc.conf` is CASE-SENSITIVE!

Space handling syntax in `arc.conf` for configuration lines:

```
(stripped space)option(stripped space)=(stripped space)value(saved␣
↪space)(value)(stripped space)
```

and for block headers:

```
[keyword:(stripped space)space is NOT allowed within identifier(stripped space)]
```

Detailed textual definition:

a) All trailing and leading spaces on each confiuration line are stripped and ignored. This aplies both to block headers and block content.

b) All spaces around the = sign in `option=value` kind of string (after 'a' is applied) are stripped and ignored. For example line `hostname = myhost.info` is treated as identical to `hostname=myhost.info`.

c) In block headers of `[keyword]` kind (after 'a' is applied) no additional spaces are allowed around `keyword` and inside `keyword`.

d) In block headers of `[keyword:identifier]` kind (after 'a' is applied) no additional spaces are allowed around `keyword` and inside both `keyword` and `identifier`. The spaces ARE allowed around `identifier` part and stripped and ignored.

Mandatory configuration options are indicated by an asterix prefix to the option name e.g: `*mandatory_configoption`. Mandatory options with undefined values will result in service stop during the startup process.

Each of the configuration options have well-defined default that is specified in this reference file. The default can take either a pre-set value, a special substitution or the keyword `undefined`. Configuration options within an enabled block take their default values in case they are missing (or commented out). Configuration parameters with `undefined` defaults takes no values. Furthermore, configuration options within disabled blocks takes no values either.

Configuration blocks are ORDER-DEPENDENT. To be safe, please use the order as indicated in the list of blocks in this reference. This is especially important for configuration blocks related to authorization and mapping. The order dependency is also honoured within options inside a certain block.

This means for instance that configuration blocks related to authorization MUST appear before used in the blocks such as `[mapping]`, `[arex/ws/jobs]` or `[gridftp/jobs]`. Order dependency within a block is for instance important when it comes to authorization decisions, as the first matching rule is used. For more details see the specific block reference.

Below we give a detailed description of all the configuration options of the different configuration blocks. Every configuration option is described in a dedicated paragraph with the following reference syntax notation. This file is parsed at buildtime to assist in configuration default parsing and validation script and so it is important that it follows the agreed syntax: For each block or option please add explanatory text with two `##` followed by a space at the beginning of the line and then an example with a single `#` and no spaces at the beginning of the line.

example_config_option

Synopsis: `example_config_option = value [optional values]`

Description: Here comes the explanation of the config option. Mandatory configuration options are indicated by an asterix prefix to the option name e.g: `*mandatory_configoption` vs. `optional_configoption`. The explanation can be followed by the special keywords in a separate line:

- `multivalued` - used to indicate that config option can be specified multiple times. This forms a set of values for the same configuration option irrespective of lines order.
- `sequenced` - used to indicate that config option is a part of the sequence and its effect on configuration depends on the lines order. Sequenced option can be specified several times in the configuration sequence independently.

Missing such keywords means the config option can only occur once in the `arc.conf`. By default the `arc.conf` config options are optional and single-valued. For some config options only a fix set of values are allowed. These are listed in a separate line after the `allowedvalues` keyword. The default of every config option is explicitly given in the `default:` line. Default can be a pre-set value, a substitution or the `undefined` keyword. The last line of the paragraph is always a valid example preceded by a single #

This option in **multivalued**.

Allowed values: `12`, `34`, `56`

Default: `34`

Example:

```
example_config_option=56
```

4.1.2 Configuration blocks and options

[common] block

Common configuration affecting all ARC components, usually related to networking or security or service behaviour. This block is mandatory. The common block options may be overridden by the specific sections of the components later. The `[common]` always appears at the beginning of the config file. The config options set within this block are available for all the other blocks thus shared by the different components of ARC.

hostname

Synopsis: `hostname = string`

Description: The FQDN of the frontend on which the ARC services are deployed.

Default: `$EXEC{hostname -f}`

Example:

```
hostname=myhost.org
```

> **Warning:** CHANGE: MODIFIED semantics in 6.0.0, not mandatory any longer.

http_proxy

Synopsis: `http_proxy = url`

Description: The http proxy server. This setting affects all client HTTP(s) requests that initiated by ARC core services, including data staging, SAML communications, and pushing SGAS accounting records. This variable is similar to setting the ARC_HTTP_PROXY environmental variable.

Default: `undefined`

Example:

```
http_proxy=proxy.mydomain.org:3128
```

> **Warning:** CHANGE: MOVED in 6.0.0 from the old `[grid-manager]` block.

x509_host_key

Synopsis: `x509_host_key = path`

Description: (previously x509_user_key) Server credential location. Sets the full path to the host private key. These variables are similar to the GSI enviroment variable `X509_USER_KEY` If indicated, the variable can be set individually for each service/component in the corresponding block.

Default: `/etc/grid-security/hostkey.pem`

Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

Warning: CHANGE: RENAMED in 6.0.0.

x509_host_cert

Synopsis: `x509_host_cert = path`

Description: (previously x509_user_cert) Server credential location. Sets the full path to the host public certificate. These variables are similar to the GSI environment variable `X509_USER_CERT` If indicated, the variable can be set individually for each service/component in the corresponding block.

Default: `/etc/grid-security/hostcert.pem`

Example:

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

Warning: CHANGE: RENAMED in 6.0.0.

x509_cert_dir

Synopsis: `x509_cert_dir = path`

Description: Location of trusted CA certificates. This variable is similar to the GSI enviroment variable `X509_CERT_DIR` If indicated, the variable can be set individually for each service/component in the corresponding block.

Default: `/etc/grid-security/certificates`

Example:

```
x509_cert_dir=/etc/grid-security/certificates
```

x509_voms_dir

Synopsis: `x509_voms_dir = path`

Description: the path to the directory containing *.lsc files needed for verification of VOMS service signature in the proxy-certificate.

Default: `/etc/grid-security/vomsdir`

Example:

```
x509_voms_dir=/etc/grid-security/vomsdir
```

voms_processing

Synopsis: `voms_processing = keyword`

Description: Defines how to behave if errors in VOMS AC processing detected. The following keywords are supported:

relaxed
: use everything that passed validation.

standard
: same as relaxed but fail if parsing errors took place and VOMS extension is marked as critical. This is a default.

strict
: fail if any parsing error was discovered

noerrors
: fail if any parsing or validation error happened.

Allowed values: `relaxed`, `standard`, `strict`, `noerrors`

Default: `standard`

Example:

```
voms_processing=strict
```

[authgroup:groupname] block

These configuration blocks contain authorization rules. An `[authrgroup:groupname]` block always defines a group of users where members of the group are those who satisfy the authorization rules. The rules within the block determine which user belong to the authgroup. Then, access control and identity mapping of ARC services are implemented via associating a authgroup with an interface, queue or a mapping rule using one of the `allowaccess`, `denyaccess` or `[mapping]` block parameters. For more info please read `Security Framework of ARC` at http://www.nordugrid.org/documents/arc-security-documentation.pdf The authgroup should not be mistaken for a virtual organisation (VO). An authgroup may match a single VO if only a single check (rule) on VO membership is perfomed.

IMPORTANT: Rules in an authgroup are processed in their order of appearance. The first matching rule decides the membership of the user to the authgroup being evaluated and the processing STOPS within that authgroup. This does not mean that the same user is not processed for the next authgroup: all `[authgroup:groupname]` blocks are evaluated, even if a user already has a match with one of the earlier groups.

All the objects used in the rules MUST be defined before it may be used. For example, to create group of authgroups you must first defined the child groups.

There are positively and negatively matching rules. If a rule is matched positively then the user tested is accepted into the respective group and further processing is stopped. Upon a negative match the user would be rejected for that group - processing stops too. The sign of rule is determined by prepending the rule with + (for positive) or - (for negative) signs. + is default and can be omitted. A rule may also be prepended with `!` to invert result of rule, which will let the rule match the complement of users. That complement operator (`!`) may be combined with the operator for positive or negative matching.

Warning: CHANGE: RENAMED the block in 6.0.0.

subject

Synopsis: `subject = certificate_subject`

Description: Rule to match specific subject of user's X.509 certificate. No masks, patterns and regular expressions are allowed.

This is **sequenced** option.

Default: `undefined`

Example:

```
subject=/O=Grid/O=Big VO/CN=Main Boss
subject=/O=Grid/O=Big VO/CN=Deputy Boss
```

file

Synopsis: `file = path`

Description: Processes a list of DNs stored in an external file one per line in grid-mapfile format (see map_with_file from `[mapping]` block, unixname is ignored) and adds those to the authgroup.

This is **sequenced** option.

Default: `undefined`

Example:

```
file=/etc/grid-security/local_users
file=/etc/grid-security/atlas_users
```

Warning: CHANGE: MODIFIED semantics in 6.0.0. The external file should only contain DNs, no complex rules. No need to change the code.

voms

Synopsis: `voms = vo group role capabilities`

Description: Match VOMS attribute in user's credential. Use * to match any value.

This is **sequenced** option.

Default: `undefined`

Example:

```
voms=nordugrid Guests * *
voms=atlas students prodman *
```

authgroup

Synopsis: `authgroup = group_name [group_name ...]`

Description: (previously group) Match user already belonging to one of specified authgroups. The authgroup referred here must be defined earlier in `arc.conf` configuration file. Multiple authgroup names may be specified for this rule. That allows creating hierarchical structure of authorization groups like `all-atlas` are those which are `atlas-users` and `atlas-admins`.

This is **sequenced** option.

Default: `undefined`

Example:

```
authgroup=local_admins
authgroup=local_admins remote_users
```

Warning: CHANGE: RENAMED in 6.0.0.

userlist

Synopsis: `userlist = ulist_name [ulist_name ...]`

Description: (previously vo) Match user belonging to ulist_name defined in an earlier `[userlist:ulist_name]` block. Multiple userlist names are allowed for this rule.

This is **sequenced** option.

Default: `undefined`

Example:

```
userlist=biousers
```

Warning: CHANGE: RENAMED in 6.0.0.

plugin

Synopsis: `plugin = timeout path [arg1 [arg2 [arg3...]]]`

Description: Run external executable or function from shared library. Rule is matched if plugin returns 0. Any other return code or timeout are treated as rule not matched. In arguments following substitutions are supported:

- `%D` - subject of certicate
- `%P` - path to proxy

ARC ships with LCAS plugin that can be enabled with following plugin configuration. For more information about configuring LCAS refer to 'Using LCAS/LCMAPS' document.

This is **sequenced** option.

Default: `undefined`

Example:

```
plugin=10 /usr/libexec/arc/arc-lcas %D %P liblcas.so /usr/lib64 /etc/lcas/lcas.db
```

authtokens

Synopsis: `authtokens = subject issuer audience scope group`

Description: Match OIDC token claims. Use * to match any value.

This is **sequenced** option.

Default: `undefined`

Example:

```
authtokens=e83eec5a-e2e3-43c6-bb67-df8f5ec3e8d0 https://wlcg.cloud.cnaf.infn.it/ * * *
```

Warning: CHANGE: INTRODUCED in 6.6.0.

all

Synopsis: `all = yes|no`

Description: Matches any or none user identity. For `yes` argument this rule always returns positive match. For `no` it is always no match.

This is **sequenced** option.

Default: `undefined`

Example:

```
all=yes
```

Warning: CHANGE: MODIFIED options in 6.0.0.

[mapping] block

This block defines the grid-identity to local UNIX identity mapping rules used by various ARC components.

Rules in the `[mapping]` block are processed IN A SEQUENCE in line order of the configuration file (from top to bottom).

There are two kind of rules:

- mapping rules that defines how the particular `authgroup` members are mapped
- policy rules that modifies the mapping rules sequence processing

Default policy for mapping rules processing is:

- processing CONTINUES to the next rule if identity of user DO NOT match `authgroup` specified in the rule (can be redefined with `policy_on_nogroup` option)
- processing STOPS if identity of user match `authgroup` specified in the mapping rule. Depend on whether this mapping rule returns valid UNIX identity the processing can be redefined with `policy_on_map` and `policy_on_nomap` options.

Policy can be redefined at the any point of configuration sequence and affects all mapping rules defined after the polcy rule.

Note: if mapping process STOPS and there is still no local UNIX identity identified, the user running A-REX will be used.

Note: when grid-identity is mapped to `root` account - request processing fails implicitely.

Warning: CHANGE: NEW block in 6.0.0.

map_to_user

Synopsis: `map_to_user = authgroup_name unixname[:unixgroup]`

Description: the users that belongs to specified authgroup are mapped to `unixname` local UNIX account that may be optionally followed by a `unixgroup` UNIX group. In case of non-existing `unixname` account the mapping rule treated as a rule that did not returned mapped identity (nomap).

This is **sequenced** option.

Default: `undefined`

Example:

```
map_to_user=authgroupA nobody:nobody
```

map_to_pool

Synopsis: `map_to_pool = authgroup_name directory`

Description: the user that belong to specified authgroup is assigned one of the local UNIX accounts in the pool. Account names that are part of this pool are stored line-by-line in the `pool` file inside the `directory`. The `directory` also contains information about used accont names stored in another files. If there are no more available accounts in the defined pool for mapping then accounts not used for a configurable time period may be reassigned. The pool behaviour, including account reuse, is configureable with the opional `directory/config` file that has INI syntax (line-by-line `key=value`). Possible keys of the `config` file are:

`timeout`
: Define the timeout in days (default is `10`) after which the UNIX account can be reassigned to another user if not used. The `0` value means no lease expiration.

This is **sequenced** option.

Default: `undefined`

Example:

```
map_to_pool=atlas /etc/grid-security/pool/atlas
```

map_with_file

Synopsis: `map_with_file = authgroup_name file`

Description: for users that belongs to specified authgroup the DN of certificate is matched against a list of DNs stored in the specified `file`, one per line followed by a local UNIX account name. The DN must be quoted if it contains blank spaces. This rule can be used to implement legacy grid-mapfile aproach.

This is **sequenced** option.

Default: `undefined`

Example:

```
map_with_file=authgroupB /etc/grid-security/grid-mapfile
```

map_with_plugin

Synopsis: `map_with_plugin = authgroup_name timeout plugin [arg1 [arg2 [...]]]`

Description: run external `plugin` executable with specified arguments to find the UNIX account name to which users that belogns to specified authgroup will be mapped to. A rule matches if the exit code is `0` and there is a UNIX account name printed on stdout (optionally followed by a UNIX group name separated by colon). The exit code 1 designates failed mapping. Any other code or timeout means fatal failure and will abort any further mapping processing. That will also cause rejection of corresponding connection. Plugin execution time is limited to `timeout` seconds.

In the arguments the following substitutions are applied before the plugin is started:

- `%D` - subject of user's certificate,
- `%P` - path to credentials' proxy file.

ARC ships with LCMAPS plugin that can be enabled with the corresponfing configuration. For more information about configuring LCMAPS refer to 'Using LCAS/LCMAPS' document.

This is **sequenced** option.

Default: `undefined`

Example:

```
map_with_plugin=authgroupC 30 /usr/libexec/arc/arc-lcmaps %D %P liblcmaps.so /usr/
↪lib64 /etc/lcmaps/lcmaps.db arc
```

policy_on_nomap

Synopsis: `policy_on_nomap = continue/stop`

Description: redefines mapping rules sequence processing policy in case identity of user match `authgroup` specified in the mapping rule and mapping rule DO NOT return valid UNIX identity. Default policy is `stop` processing the furhter rules. For example this policy will be triggered if pool is depleted, certificate subject is missing in the map file used for defined authgroup or plugin execution failed.

This is **sequenced** option.

Default: `undefined`

Allowed values: `continue`, `stop`

Example:

```
policy_on_nomap=continue
```

policy_on_map

Synopsis: `policy_on_map = continue/stop`

Description: redefines mapping rules sequence processing policy in case identity of user match `authgroup` specified in the mapping rule and mapping rule return valid UNIX identity. Default policy is `stop` processing the furhter rules. This policy will be triggered if rule successfully returns the result (allocated in pool, matched in map file, plugin call was successful).

This is **sequenced** option.

Default: `undefined`

Allowed values: `continue`, `stop`

Example:

```
policy_on_map=stop
```

policy_on_nogroup

Synopsis: `policy_on_nogroup = continue/stop`

Description: redefines mapping rules sequence processing policy in case identity of user DO NOT match `authgroup` specified in the mapping rule. Default policy is `continue` processing the furhter rules.

This is **sequenced** option.

Default: `undefined`

Allowed values: `continue`, `stop`

Example:

```
policy_on_nogroup=stop
```

[authtokens] block

This block activates processing of OIDC tokens as defined in WLCG profile.

Warning: CHANGE: NEW block in 6.6.0.

[lrms] block

This block specifies the characteristics of the Local Resource Manager System (batch system) underneath the ARC CE. This block contains all the lrms-specific parameters and information. Configuration values in this block are available for A-REX, the backends, accounting and infosys ARC subsystems.

ARC support the most common LRMS flavours.

Warning: CHANGE: NEW block in 6.0.0. Contains parameters previously set in `[common]`, `[infosys]`, `[cluster]`, `[queue]`

lrms

Synopsis: `*lrms = lrmstype [defaultqueue]`

Description: Sets the type of the LRMS (queue system) and optionally the default queue name. ONLY ONE LRMS IS ALLOWED. MULTIPLE LRMS ENTRIES WILL TRIGGER UNEXPECTED BEHAVIOUR.

For lrmstype, the following values can be chosen:

- fork - simple forking of jobs to the same node as the server
- sge - (Sun/Oracle) Grid Engine
- condor - Condor
- pbs - PBS (covers Torque and other old PBS flavours e.g. OpenPBS, older PBSPro, etc)
- pbspro - Altair PBS Professional
- lsf - LSF
- ll - LoadLeveler
- slurm - SLURM
- boinc - Boinc
- slurmpy - new EXPERIMENTAL SLURM scripts (contains the ssh remote batch management as well)

The optional `defaultqueue` parameter specifies the name of an existing LRMS queue in the cluster that will be used by AREX as the default queue to submit grid jobs in case the client does not specify a queue name during the job submission procees. This queue name must match one of the `[queue:queue_name]` blocks.

Allowed values: `fork`, `sge`, `condor`, `pbs`, `pbspro`, `lsf`, `ll`, `slurm`, `boinc`, `slurmpy`

Default: `undefined`

mandatory

Example:

```
lrms=pbspro gridlong
lrms=slurm
```

Warning: CHANGE: slurmpy NEW in 6.0.0. pbspro NEW in 6.1.0

lrmsconfig

Synopsis: `lrmsconfig = text`

Description: An optional free text field to describe the configuration of your Local Resource Management System (batch system). The value is published in the infosys, and is not used otherwise.

Default: `undefined`

Example:

```
lrmsconfig=single job per processor
```

Warning: CHANGE: MOVED in 6.0.0 from the `[cluster]` block.

benchmark

Synopsis: `benchmark = string`

Description: Defines the default benchmark specification to store in the accounting AAR records (as it is) if per-job data is missing. It is advised to set it to cluster-wide defaults in case of reporting to APEL to aviod records diversity for failed jobs or buggy backends.

Default: `HEPSPEC:1.0`

Example:

```
benchmark=HEPSPEC:12.26
```

> **Warning:** CHANGE: NEW in 6.8.0

defaultmemory

Synopsis: `defaultmemory = number`

Description: The LRMS memory request of job to be set by the LRMS backend scripts, if a user submits a job without specifying how much memory should be used. The order of precedence is: job description -> defaultmemory. This is the amount of memory (specified in MB) that a job will request.

Default: `undefined`

Example:

```
defaultmemory=512
```

> **Warning:** CHANGE: MOVED in 6.0.0 from the `[cluster]` block.

nodename

Synopsis: `nodename = path`

Description: Redefine the command to obtain hostname of LRMS worker node. By default the value is defined on buildtime and depend on the OS. In most cases `/bin/hostname -f` will be used.

Note: this way of getting WN hostname will be used only in case of particular LRMS backend had no native LRMS-defined way.

Default: `undefined`

Example:

```
nodename=/bin/hostname -s
```

> **Warning:** CHANGE: MOVED in 6.0.0 from the `[arex]` block.

gnu_time

Synopsis: `gnu_time = path`

Description: Path to the GNU time command on the LRMS worker nodes. If time command exists on the node, jobscript will write additional diagnostic information.

Default: `/usr/bin/time`

Example:

```
gnu_time=/usr/bin/time
```

Warning: CHANGE: MOVED in 6.0.0 from the `[arex]` block.

movetool

Synopsis: `movetool = comand`

Description: Redefine the command used to move files during jobscript execution on LRMS worker node (the command should be available on WNs). This in particular applies to files movement from sessiondir to scratchdir in the shared sessiondir case.

Default: `mv`

Example:

```
movetool=rsync -av
```

Warning: CHANGE: NEW in 6.8

pbs_bin_path

Synopsis: `pbs_bin_path = path`

Description: The path to the qstat,pbsnodes,qmgr etc PBS binaries, no need to set if PBS is not used

Default: `/usr/bin`

Example:

```
pbs_bin_path=/usr/bin
```

pbs_log_path

Synopsis: `pbs_log_path = path`

Description: The path of the PBS server logfiles which are used by A-REX to determine whether a PBS job is completed. If not specified, A-REX will use qstat for that.

Default: `/var/spool/pbs/server_logs`

Example:

```
pbs_log_path=/var/spool/pbs/server_logs
```

pbs_dedicated_node_string

Synopsis: `pbs_dedicated_node_string = string`

Description: (previously dedicated_node_string) The string which is used in the PBS node config to distinguish the grid nodes from the rest. Suppose only a subset of nodes are available for grid jobs, and these nodes have a common `node property` string, this case the string should be set to this value and only the nodes with the corresponding `pbs node property` are counted as grid enabled nodes. Setting the dedicated_node_string to the value of the `pbs node property` of the grid-enabled nodes will influence how the totalcpus, user freecpus is calculated. You don't need to set this attribute if your cluster is fully available for the grid and your cluster's PBS config does not use the `node property` method to assign certain nodes to grid queues. You shouldn't use this config option unless you make sure your PBS config makes use of the above described setup.

Default: `undefined`

Example:

```
pbs_dedicated_node_string=gridnode
```

> **Warning:** CHANGE: RENAMED and MOVED in 6.0.0 from `[cluster]`.

condor_bin_path

Synopsis: `condor_bin_path = path`

Description: Path to Condor binaries. Must be set if Condor is used.

Default: `/usr/bin`

Example:

```
condor_bin_path=/opt/condor/bin
```

condor_config

Synopsis: `condor_config = path`

Description: Full path to Condor config file. Must be set if Condor is used and the config file is not in its default location (/etc/condor/condor_config or ~/condor/condor_config). The full path to the file should be given.

Default: `/etc/condor/condor_config`

Example:

```
condor_config=/opt/condor/etc/condor_config
```

condor_rank

Synopsis: `condor_rank = ClassAd_float_expression`

Description: If you are not happy with the way Condor picks nodes when running jobs, you can define your own ranking algorithm by optionally setting the condor_rank attribute. condor_rank should be set to a ClassAd float expression that you could use in the Rank attribute in a Condor job description.

Default: `undefined`

Example:

```
condor_rank=(1-LoadAvg/2)*(1-LoadAvg/2)*Memory/1000*KFlops/1000000
```

condor_requirements

Synopsis: `condor_requirements = string`

Description: Specify additional constraints for Condor resources. The value of `condor_requirements` must be a valid constraints string which is recognized by a `condor_status -constraint ...` command. It can reference pre-defined ClassAd attributes (like Memory, Opsys, Arch, HasJava, etc) but also custom ClassAd attributes. To define a custom attribute on a condor node, just add two lines like the ones below in the `$(hostname).local` config file on the node:

```
NORDUGRID_RESOURCE=TRUE
STARTD_EXPRS = NORDUGRID_RESOURCE, $(STARTD_EXPRS)
```

A job submitted to this resource is allowed to run on any node which satisfies the `condor_requirements` constraint. If `condor_requirements` is not set, jobs will be allowed to run on any of the nodes in the pool. When configuring multiple queues, you can differentiate them based on memory size or disk space, for example.

Default: `undefined`

Example:

```
condor_requirements=(OpSys == "linux" && NORDUGRID_RESOURCE && Memory >= 1000 &&␣
↪Memory < 2000)
```

sge_bin_path

Synopsis: `sge_bin_path = path`

Description: Path to Sun Grid Engine (SGE) binaries, Default is search for qsub command in the shell PATH

Default: `undefined`

Example:

```
sge_bin_path=/opt/n1ge6/bin/lx24-x86
```

sge_root

Synopsis: `sge_root = path`

Description: Path to SGE installation directory. MUST be set if SGE is used.

Default: `/gridware/sge`

Example:

```
sge_root=/opt/n1ge6
```

sge_cell

Synopsis: `sge_cell = name`

Description: The name of the SGE cell to use. This option is only necessary in case SGE is set up with a cell name different from 'default'

Default: `default`

Example:

```
sge_cell=default
```

sge_qmaster_port

Synopsis: `sge_qmaster_port = port`

Description: The SGE port options should be used in case SGE command line clients require SGE_QMASTER_PORT and SGE_EXECD_PORT environment variables to be set. Usually they are not necessary.

Default: `undefined`

Example:

```
sge_qmaster_port=536
```

sge_execd_port

Synopsis: `sge_execd_port = port`

Description: The SGE port options should be used in case SGE command line clients requre SGE_QMASTER_PORT and SGE_EXECD_PORT environment variables to be set. Usually they are not necessary.

Default: `undefined`

Example:

```
sge_execd_port=537
```

sge_jobopts

Synopsis: `sge_jobopts = string`

Description: Additional SGE options to be used when submitting jobs to SGE

Default: `undefined`

Example:

```
sge_jobopts=-P atlas -r yes
```

slurm_bin_path

Synopsis: `slurm_bin_path = path`

Description: Path to SLURM binaries, must be set if installed outside of normal PATH

Default: `/usr/bin`

Example:

```
slurm_bin_path=/usr/bin
```

slurm_wakeupperiod

Synopsis: `slurm_wakeupperiod = numsec`

Description: How long should infosys wait before querying SLURM for new data (seconds)

Default: `30`

Example:

```
slurm_wakeupperiod=15
```

slurm_use_sacct

Synopsis: `slurm_use_sacct = yes/no`

Description: Indicates whether ARC should use sacct instead of scontrol to obtain information about finished jobs Not supported by slurmpy.

Allowed values: `yes`, `no`

Default: `yes`

Example:

```
slurm_use_sacct=yes
```

slurm_requirements

Synopsis: `slurm_requirements = string`

Description: Use this option to specify extra SLURM-specific parameters.

Default: `undefined`

Example:

```
slurm_requirements=memory on node >> 200
```

Warning: CHANGE: NEW in 6.0.0.

slurm_query_retries

Synopsis: `slurm_query_retries = number`

Description: Number of sacct/scontrol retries performed in scan-SLURM-job If slurm is overloaded the sacct/scontrol command call may fail. If retries > 1 sacct/scontrol is retried after some seconds for that(those) particular job(s). If all retry attempts fail, the next scan-SLURM-job institiation will pick up the job(s) from last time.

Default: `1`

Example:

```
slurm_query_retries=3
```

lsf_bin_path

Synopsis: `lsf_bin_path = path`

Description: The PATH to LSF bin folder

Default: `/usr/bin`

Example:

```
lsf_bin_path=/usr/local/lsf/bin/
```

lsf_profile_path

Synopsis: `lsf_profile_path = path`

Description: Path to the profile.lsf file. Infoprovider scripts will source profile.lsf to setup LSF utilites environment.

Default: `/usr/share/lsf/conf/profile.lsf`

Example:

```
lsf_profile_path=/usr/local/share/lsf/conf/profile.lsf
```

lsf_architecture

Synopsis: `lsf_architecture = string`

Description: CPU architecture to request when submitting jobs to LSF. Use only if you know what you are doing.

Default: `undefined`

Example:

```
lsf_architecture=PowerPC
```

> **Warning:** CHANGE: MOVED in 6.0.0 from `[queue:name]` block.

ll_bin_path

Synopsis: `ll_bin_path = path`

Description: The PATH to the LoadLeveler bin folder

Default: `/usr/bin`

Example:

```
ll_bin_path=/opt/ibmll/LoadL/full/bin
```

ll_consumable_resources

Synopsis: `ll_consumable_resources = yes/no`

Description: Indicates whether the LoadLeveler setup is using Consumable Resources.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
ll_consumable_resources=yes
```

boinc_db_host

Synopsis: `boinc_db_host = hostname`

Description: Connection strings for the boinc database: host

Default: `localhost`

Example:

```
boinc_db_host=localhost
```

boinc_db_port

Synopsis: `boinc_db_port = port`

Description: Connection strings for the boinc database: port

Default: `3306`

Example:

```
boinc_db_port=3306
```

boinc_db_name

Synopsis: `boinc_db_name = db_name`

Description: Connection strings for the boinc database: db_name

Default: `undefined`

Example:

```
boinc_db_name=myproject
```

boinc_db_user

Synopsis: `boinc_db_user = user`

Description: Connection strings for the boinc database: db_user

Default: `undefined`

Example:

```
boinc_db_user=boinc
```

boinc_db_pass

Synopsis: `boinc_db_pass = pwd`

Description: Connection strings for the boinc database: pwd

Default: `undefined`

Example:

```
boinc_db_pass=password
```

boinc_app_id - ID of the app handled by this CE. Setting this option makes database queries much faster in large projects with many apps.

Default: `undefined`

Example:

```
boinc_app_id=1
```

boinc_project_dir - Base directory of the BOINC project

Default: `undefined`

Example:

```
boinc_project_dir=/home/boinc
```

[lrms/ssh] block

This sub-block configures the ssh environment for remote batch management. Currently ONLY the python slurm (slurmpy) EXPERIMENTAL batch module is able to utilize the remote ssh feature. Parameters within this block are relevant if the cluster frontend is remotely located wrt. CE frontend (machine running A-REX). If specified with the parameters below, the session, cache and runtime directories will be mounted from the cluster frontend on the CE frontend using sshfs. Job submission and management will done using ssh (Paramiko).

Warning: CHANGE: NEW sub-block in 6.0.0 to configure ssh support in slurmpy as a part of tech preview.

remote_host

Synopsis: `*remote_host = hostname`

Description: Define the remote cluster frontend machine which contains session, cache and runtime directories and where jobs should be submitted to.

Default: `undefined`

mandatory

Example:

```
remote_host=myremotehost.org
```

private_key

Synopsis: `*private_key = path`

Description: Location of the private key which should be used establish connection to the machine specified by the 'remote_host' attribute.

Default: `undefined`

mandatory

Example:

```
private_key=/etc/grid-security/hostkey-priv.pem
```

remote_user

Synopsis: `*remote_user = user`

Description: User on remote cluster which should be used for mounting directories, submitting and managing jobs, and gathering information about cluster.

Default: `undefined`

mandatory

Example:

```
remote_user=grid
```

remote_sessiondir

Synopsis: `remote_sessiondir = path`

Description: Session directory on cluster frontend to be mounted (sshfs) on CE machine at directory specified by the ‘sessiondir’ attribute in the `[arex]` block.

Default: `undefined`

mandatory

Example:

```
remote_sessiondir=/scratch/grid
```

remote_cachedir

Synopsis: `remote_cachedir = path`

Description: Cache directory on cluster frontend to be mounted (sshfs) on CE machine at directory specified by the ‘cachedir’ attribute in the `[arex]` block.

Default: `undefined`

Example:

```
remote_cachedir=/scratch/cache
```

ssh_timeout

Synopsis: `ssh_timeout = time`

Description: Timeout of establishing ssh connection. Unit in seconds.

Default: `10`

Example:

```
ssh_timeout=10
```

[arex] block

The `[arex]` block, together with its various subblocks, configures the A-REX service hosted in `arched`. A-REX takes care of various middleware tasks on the frontend such as job creation and management, stagein/stageout, LRMS job submission, data caching, etc…

> **Warning:** CHANGE: RENAMED block in 6.0.0.

user

Synopsis: `user = user[:group]`

Description: Switch to a non root user/group after startup. Use with caution because of limited functionality when arex is not run under root.

Default: `root`

Example:

```
user=grid:grid
```

norootpower

Synopsis: `norootpower = yes|no`

Description: If set to yes, all job management processes will switch to mapped user's identity while accessing session directory. This is useful if session directory is on NFS with root squashing turned on.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
norootpower=yes
```

delegationdb

Synopsis: `delegationdb = db_name`

Description: specify which DB to use to store delegations. Currently supported db_names are bdb and sqlite

Default: `sqlite`

Example:

```
delegationdb=sqlite
```

Warning: CHANGE: MODIFIED in 6.0.0 with new default.

watchdog

Synopsis: `watchdog = yes/no`

Description: Specifies if additional watchdog processes is spawned to restart main process if it is stuck or dies.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
watchdog=no
```

loglevel

Synopsis: `loglevel = level`

Description: (previously debug) Set loglevel of the arched daemon hosting A-REX service between 0 (FATAL) and 5 (DEBUG). Defaults to 3 (INFO).

Allowed values: `0`, `1`, `2`, `3`, `4`, `5`, `FATAL`, `ERROR`, `WARNING`, `INFO`, `VERBOSE`, `DEBUG`

Default: `3`

Example:

```
loglevel=3
```

Warning: CHANGE: RENAMED in 6.0.0.

logfile

Synopsis: `logfile = path`

Description: Specify A-REX log file location. If using an external log rotation tool be careful to make sure it matches the path specified here.

Default: `/var/log/arc/arex.log`

Example:

```
logfile=/var/log/arc/arex.log
```

Warning: CHANGE: MODIFIED default (renamed file) in 6.0.0.

joblog

Synopsis: `joblog = path`

Description: Specifies where to store specialized log about started and finished jobs. If path is empty log is NOT written. Controlled by logrotate if deafult name is kept. This log is not used by any other part of ARC so can be safely disabled if you are not interested in storing jobs log.

Default: `/var/log/arc/arex-jobs.log`

Example:

```
joblog=
```

fixdirectories

Synopsis: `fixdirectories = yes/missing/no`

Description: Specifies during startup A-REX should create all directories needed for it operation and set suitable default permissions. If `no` is specified then A-REX does nothing to prepare its operational environment. In case of `missing` A-REX only creates and sets permissions for directories which are not present yet. For `yes` all directories are created and permissions for all used directories are set to default safe values.

Allowed values: `yes`, `missing`, `no`

Default: `yes`

Example:

```
fixdirectories=yes
```

controldir

Synopsis: `controldir = path`

Description: The directory of the A-REX's internal job metadata files. For a heavy loaded computing elements you can consider to locate controldir on a dedicated partition optimized for small random reads and writes. The directory is not needed on the nodes.

Default: `/var/spool/arc/jobstatus`

Example:

```
controldir=/var/spool/arc/jobstatus
```

sessiondir

Synopsis: `sessiondir = path [drain]`

Description: the directory which holds the sessiondirs of the grid jobs. Multiple session directories may be specified. In this case jobs are spread evenly over the session directories. If `sessiondir=*` is set, the session directory will be spread over the `${HOME}/.jobs` directories of every locally mapped unix user. It is preferred to use common session directories. The path may be followed by `drain`, in which case no new jobs will be assigned to that sessiondir, but current jobs will still be processed and accessible.

This option in **multivalued**.

Default: `/var/spool/arc/sessiondir`

Example:

```
sessiondir=/scratch/arcsessions drain
sessiondir=*
```

defaultttl

Synopsis: `defaultttl = [ttl [ttr]]`

Description: The ttl parameter sets the time in seconds for how long a job session directory will survive after job execution has finished. If not specified the default is 1 week. The ttr parameter sets how long information about a job will be kept after the session directory is deleted. If not specified, the ttr default is one month.

Default: `604800 2592000`

Example:

```
defaultttl=2592000
```

shared_filesystem

Synopsis: `shared_filesystem = yes/no`

Description: Specifies if computing nodes can access folders mounted with protocols like NFS with the same pathnames as the frontend. Note that the default 'yes' assumes that the paths to the session directories are the same on both frontend and nodes. If these paths are not the same, then one should set the scratchdir option. The option changes the RUNTIME_NODE_SEES_FRONTEND variable in the submission scripts.

Allowed values: `yes`, `no`

Default: `yes`

Example:

```
shared_filesystem=yes
```

scratchdir

Synopsis: `scratchdir = path`

Description: The path on computing node to move session directory to before execution. If defined should contain the path to the directory on the computing node which can be used to store a jobs' files during execution. Sets the environment variable RUNTIME_LOCAL_SCRATCH_DIR. If the variable is not set, then the session dir is not moved before execution. Don't set this parameter unless you want to move the sessiondir to scratchdir on the node.

Default: `undefined`

Example:

```
scratchdir=/local/scratch/
```

shared_scratch

Synopsis: `shared_scratch = path`

Description: The path on frontend where scratchdir can be found. If defined should contain the path corresponding to that set in scratchdir as seen on the frontend machine. Sets the environment variable RUNTIME_FRONTEND_SEES_NODE.

Default: `undefined`

Example:

```
shared_scratch=/mnt/scratch
```

tmpdir

Synopsis: `tmpdir = path`

Description: A temporary directory used by A-REX.

Default: `/tmp`

Example:

```
tmpdir=/tmp
```

runtimedir

Synopsis: `runtimedir = path`

Description: The directory which holds the additional runtimeenvironment scripts, added by system administrator. Several directories can be specified. To enable RTEs to be advertised in the information system and used during submission the arcctl tool should be used.

This option in **multivalued**.

Default: `undefined`

Example:

```
runtimedir=/var/spool/arc/extraruntimes
runtimedir=/cvmfs/vo/arcruntime
```

maxjobs

Synopsis: `maxjobs = number1 number2 number3 number4 number5`

Description: specifies maximum allowed number of jobs. number1 - jobs which are not in FINISHED state (jobs tracked in RAM) number2 - jobs being run (SUBMITTING, INLRMS states) number3 - jobs being processed per DN number4 - jobs in whole system number5 - LRMS scripts limit (jobs in SUBMITTING and CANCELING) A parameter set to -1 means no limit.

Default: `-1 -1 -1 -1 -1`

Example:

```
maxjobs=10000 10 2000 -1 -1
```

> **Warning:** CHANGE: MODIFIED in 6.0.0. Explicitly indicate `no limit` with -1. `Missing number` should not be allowed.

maxrerun

Synopsis: `maxrerun = number`

Description: Specifies how many times job can be rerun if it failed in LRMS. This is only an upper limit, the actual rerun value is set by the user in his xrsl.

Default: `5`

Example:

```
maxrerun=5
```

statecallout

Synopsis: `statecallout = state options plugin_path [plugin_arguments]`

Description: (previously authplugin) Enables a callout feature of A-REX: every time job goes to `state` A-REX will run `plugin_path` executable. The following states are allowed: ACCEPTED, PREPARING, SUBMIT, FINISHING, FINISHED and DELETED. Options consist of `key=value` pairs separated by comma. Possible keys are:

`timeout`
: defines the timeout in seconds to wait for plugin execution (`timeout=` can be omitted).

`onsuccess`, `onfailure`, `ontimeout`
: defines the action that A-REX should take on successful execution (exit code 0), failed execution (exit code is not 0) or execution timeout respectively.

Possible actions are:

- `pass` - continue executing job,
- `fail` - cancel job,
- `log` - write to log about the failure and continue executing job.

It is possible to use following sugstitutions to construct plugin command line:

- `%R` - session root (value of sessiondir in `[arex]` block)
- `%C` - controldir path
- `%U` - username of mapped UNIX account
- `%u` - numeric UID of mapped UNIX account
- `%g` - numeric GID of mapped UNIX account
- `%H` - home directory of mapped UNIX account as specified in `/etc/passwd`
- `%Q` - default queue (see `lrms` configuration option in `[lrms]` block)
- `%L` - LRMS name (see `lrms` configuration option in `[lrms]` block)
- `%W` - ARC installation path (corresponds to the `ARC_LOCATION` environmental variable)
- `%F` - path to configuration file for this instance
- `%I` - job ID (substituted in runtime)
- `%S` - job state (substituted in runtime)

Plugins included into ARC distribution:

- `arc-blahp-logger` - write accounting log for every finished job in BLAH format

This option in **multivalued**.

Default: `undefined`

Example:

```
statecallout=FINISHED timeout=10,onfailure=pass /usr/libexec/arc/arc-blahp-logger -I
↪%I -U %u -L %C/job.%I.local -P %C/job.%I.proxy
```

> **Warning:** CHANGE: RENAMED in 6.0.0 from authplugin.

wakeupperiod

Synopsis: `wakeupperiod = time`

Description: Specifies how often A-REX checks for new jobs arrived, job state change requests, etc. That is responsiveness of A-REX. `time` is time period in seconds. Default is 3 minutes. Usually no need to change this parameter because important state changes are also triggering out-of-schedule checks.

Note: this parameter does not affect responsiveness of backend scripts - especially `scan-<LRMS>-job`. That means that upper estimation of time for

detecting job finished executing is sum of responsiveness of backend script + wakeupperiod.

Default: `180`

Example:

```
wakeupperiod=180
```

infoproviders_timelimit

Synopsis: `infoproviders_timelimit = seconds`

Description: (previously infoproviders_timeout) Sets the execution time limit of the infoprovider scripts started by the A-REX. Infoprovider scripts running longer than the specified timelimit are gracefully handled by the A-REX (the behaviour depends on the state of the system) Increase this value if you have many jobs in the controldir and infoproviders need more time to process.

Default: `10800`

Example:

```
infoproviders_timelimit=10800
```

Warning: CHANGED: RENAMED and MOVED in 6.0.0 to `[arex]` block.

pidfile

Synopsis: `pidfile = path`

Description: Specify location of file containing PID of daemon process.

Default: `/run/arched-arex.pid`

Example:

```
pidfile=/run/arched-arex.pid
```

mail

Synopsis: `mail = email_address`

Description: Specifies the email address from where the notification mails are sent

Default: `$VAR{user}@$VAR{[common]hostname}`

Example:

```
mail=grid.support@somewhere.org
```

Warning: CHANGE: MODIFIED in 6.0.0.

gnu_time

Synopsis: `gnu_time = path`

Description: The gnu time command

Warning: CHANGE: MOVED in 6.0.0 to `[lrms]`.

nodename

Synopsis: `nodename = path`

Description: The command to obtain hostname of computing node.

Warning: CHANGE: MOVED in 6.0.0 to `[lrms]`.

helper

Synopsis: `helper = user executable arguments`

Description: By enabling this parameter A-REX will run an external `helper` program under the user useraccount. The program will be kept running, every time the executable finishes it will be started again. As a limitation, currently only '.' is supported as username, which corresponds to the user running A-REX.

Default: `undefined`

Example:

```
helper=. /usr/local/bin/myutility
```

helperlog

Synopsis: `helperlog = path`

Description: Configuration option to specify the location of log for helpers.

Default: `/var/log/arc/job.helper.errors`

Example:

```
helperlog=/var/log/arc/job.helper.errors
```

> **Warning:** CHANGE: NEW PARAMETER in 6.0.0.

forcedefaultvoms

Synopsis: `forcedefaultvoms = VOMS_FQAN`

Description: specify VOMS FQAN which user will be assigned if his/her credentials contain no VOMS attributes. To assign different values to different queues put this command into `[queue]` block.

Default: `undefined`

Example:

```
forcedefaultvoms=/vo/group/subgroup
```

[arex/cache] block

This subblock enables and configures the cache functionality of A-REX. A-REX can cache input files downloaded as part of the stage-in process of grid jobs so that subsequent jobs requiring the same file don't have to download it again. The cached file will be symlinked (or copied) into the session directory of the job. To disable to cache functionality simply comment out the `[arex/cache]` config block. It is a good idea to have the cache on its own separate file system that is shared with the nodes. For more information about the cache functionality of A-REX consult the Data Cache technical description in the online documentation.

> **Warning:** CHANGE: NEW block in 6.0.0.

cachedir

Synopsis: `*cachedir = cache_path [link_path]`

Description: Specifies a directory to store cached data. Multiple cache directories may be specified. Cached data will be distributed evenly over the caches. Optional `link_path` specifies the path at which the `cache_path` is accessible on computing nodes, if it is different from the path on the A-REX host. If `link_path` is set to `.` files are not soft-linked, but copied to session directory. If a cache directory needs to be drained, then `link_path` should specify `drain`, in which case no new files will be added to the cache and files in the cache will no longer be used. Setting `link_path` to `readonly` ensures that no new files are written to this cache, but existing files can still be used. Draining and read-only caches are not cleaned by the A-REX cache cleaner. A restart of A-REX is required when changing cache options.

This option in **multivalued**.

Default: `undefined`

Example:

```
cachedir=/scratch/cache
cachedir=/shared/cache /frontend/jobcache
cachedir=/fs1/cache drain
```

Warning: CHANGE: Added readonly option in 6.7

[arex/cache/cleaner] block

This subblock enables the cleaning functionality of the cache. If this block is not enabled then the cache will not be cleaned by A-REX. Either cachesize or cachelifetime should also be set to enable cleaning.

Warning: CHANGE: NEW block in 6.0.0.

logfile

Synopsis: `logfile = path`

Description: (previously cachelogfile) sets the filename where output of the cache-clean tool should be logged. Defaults to /var/log/arc/cache-clean.log.

Default: `/var/log/arc/cache-cleaner.log`

Example:

```
logfile=/tmp/cache-clean.log
```

Warning: CHANGE: RENAMED in 6.0.0.

loglevel

Synopsis: `loglevel = level`

Description: (previously cacheloglevel) specifies the level of logging by the cache-clean tool, between 0 (FATAL) and 5 (DEBUG). Defaults to 3 (INFO).

Allowed values: `0`, `1`, `2`, `3`, `4`, `5`, `FATAL`, `ERROR`, `WARNING`, `INFO`, `VERBOSE`, `DEBUG`

Default: `3`

Example:

```
loglevel=4
```

Warning: CHANGE: RENAMED in 6.0.0.

cachesize

Synopsis: `cachesize = max min`

Description: Specifies high and low watermarks for space used by cache, as a percentage of the space on the file system on which the cache directory is located. When the max is exceeded, files will be deleted to bring the used space down to the min level. It is a good idea to have the cache on its own separate file system.

Default: `100 100`

Example:

```
cachesize=50 20
```

calculatesize

Synopsis: `calculatesize = filesystem/cachedir`

Description: (previously cacheshared) specifies the way the space occupied by the cache will be calculated. If set to cachedir then cache-clean calculates the size of the cache instead of using filesystem used space.

Allowed values: `filesystem`, `cachedir`

Default: `filesystem`

Example:

```
calculatesize=cachedir
```

Warning: CHANGE: RENAMED in 6.0.0 .

cachelifetime

Synopsis: `cachelifetime = time`

Description: Turns on time-based file cleaning. Files accessed less recently than the given time period will be deleted. Example values of this option are 1800, 90s, 24h, 30d. When no suffix is given the unit is seconds.

Default: `undefined`

Example:

```
cachelifetime=30d
```

cachespacetool

Synopsis: `cachespacetool = path [options]`

Description: specifies an alternative tool to `df` that cache-clean should use to obtain space information on the cache file system. The output of this command must be `total_bytes used_bytes`. The cache directory is passed as the last argument to this command.

Default: `undefined`

Example:

```
cachespacetool=/etc/getspace.sh
```

cachecleantimeout

Synopsis: `cachecleantimeout = time`

Description: the timeout in seconds for running the cache-clean tool. If using a large cache or slow file system this value can be increased to allow the cleaning to complete. Defaults to 3600 (1 hour).

Default: `3600`

Example:

```
cachecleantimeout=10000
```

[arex/data-staging] block

This subblock enables and configures the data staging capabilities of A-REX. A subsystem called DTR (Data Transfer Reloaded) is responsible for collecting input data for a job before submission to the LRMS, and for staging out data after the job has finished. Automagic data staging of A-REX is a very powerful feature, disabling this functionality (by commenting out the subblock) is not recommended.

Warning: CHANGE: RENAMED block in 6.0.0.

loglevel

Synopsis: `loglevel = number`

Description: (previously debug) Sets the log level for transfer logging in job.id.errors files, between 0 (FATAL) and 5 (DEBUG). Default is to use value set by loglevel option in `[arex]` section.

Allowed values: `0`, `1`, `2`, `3`, `4`, `5`, `FATAL`, `ERROR`, `WARNING`, `INFO`, `VERBOSE`, `DEBUG`

Default: `$VAR{[arex]loglevel}`

Example:

```
loglevel=4
```

Warning: CHANGE: RENAMED in 6.0.0.

logfile

Synopsis: `logfile = path`

Description: (previously central_logfile) A central file in which all data staging messages from every job will be collected and logged in addition to their job.id.errors files. If this option is not present or the path is empty the log file is not created. This file is not automatically controlled by logrotate unless you name it as /var/log/arc/datastaging.log.

Default: `undefined`

Example:

```
logfile=/var/log/arc/datastaging.log
```

Warning: CHANGE: RENAMED in 6.0.0.

statefile

Synopsis: `statefile = path`

Description: (previously dtrlog) A file in which data staging state information (for monitoring and recovery purposes) is periodically dumped.

Default: `$VAR{[arex]controldir}/dtr.state`

Example:

```
statefile=/tmp/dtr.state
```

> **Warning:** CHANGE: RENAMED and MODIFIED in 6.0.0, new default value.

usehostcert

Synopsis: `usehostcert = yes/no`

Description: Whether the A-REX host certificate should be used for communication with remote hosts instead of the users' proxies.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
usehostcert=yes
```

maxtransfertries

Synopsis: `maxtransfertries = number`

Description: the maximum number of times download and upload will be attempted per job (retries are only performed if an error is judged to be temporary)

Default: `10`

Example:

```
maxtransfertries=20
```

passivetransfer

Synopsis: `passivetransfer = yes/no`

Description: If yes, gridftp transfers are passive. Setting this option to yes can solve transfer problems caused by firewalls.

Allowed values: `yes`, `no`

Default: `yes`

Example:

```
passivetransfer=yes
```

Warning: CHANGE: NEW DEFAULT in 6.0.0.

globus_tcp_port_range

Synopsis: `globus_tcp_port_range = port_range`

Description: In a firewalled environment the software which uses GSI needs to know what ports are available. This parameter is only needed if `passivetransfer=no` was set. These variable are similar to the Globus enviroment variables `GLOBUS_TCP_PORT_RANGE` and `GLOBUS_UDP_PORT_RANGE`.

Default: `9000,9300`

Example:

```
globus_tcp_port_range=9000,12000
```

Warning: CHANGE: MOVED in 6.0.0.

globus_udp_port_range

Synopsis: `globus_udp_port_range = port_range`

Description: In a firewalled environment the software which uses GSI needs to know what ports are available. This parameter is only needed if `passivetransfer=no` was set. These variable are similar to the Globus enviroment variables `GLOBUS_TCP_PORT_RANGE` and `GLOBUS_UDP_PORT_RANGE`.

Default: `9000,9300`

Example:

```
globus_udp_port_range=9000,12000
```

Warning: CHANGE: MOVED in 6.0.0.

httpgetpartial

Synopsis: `httpgetpartial = yes/no`

Description: If yes, HTTP GET transfers may transfer data in chunks/parts. If no - data is always transfered in one piece.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
httpgetpartial=no
```

Warning: CHANGE: NEW default in 6.0.0.

speedcontrol

Synopsis: `speedcontrol = min_speed min_time min_average_speed max_inactivity`

Description: specifies how slow data transfer must be to trigger error. Transfer is cancelled if speed is below min_speed bytes per second for at least min_time seconds, or if average rate is below min_average_speed bytes per second, or no data was transferred for longer than max_inactivity seconds. Value of zero turns feature off.

Default: `0 300 0 300`

Example:

```
speedcontrol=0 300 100 300
speedcontrol=
```

> **Warning:** CHANGE: MODIFIED in 6.0.0: missing parameter `speedcontrol=` should also turn the feature off, not just zero value.

maxdelivery

Synopsis: `maxdelivery = number`

Description: Maximum number of concurrent file transfers, i.e. active transfers using network bandwidth. This is the total number for the whole system including any remote staging hosts.

Default: `10`

Example:

```
maxdelivery=40
```

maxprocessor

Synopsis: `maxprocessor = number`

Description: Maximum number of concurrent files in each of the DTR internal pre- and post-processing states, eg cache check or replica resolution.

Default: `10`

Example:

```
maxprocessor=20
```

maxemergency

Synopsis: `maxemergency = number`

Description: Maximum `emergency` slots which can be assigned to transfer shares when all slots up to the limits configured by the above two options are used by other shares. This ensures shares cannot be blocked by others.

Default: `1`

Example:

```
maxemergency=5
```

maxprepared

Synopsis: `maxprepared = number`

Description: Maximum number of files in a prepared state, i.e. pinned on a remote storage such as SRM for transfer. A good value is a small multiple of maxdelivery.

Default: `200`

Example:

```
maxprepared=250
```

sharepolicy

Synopsis: `sharepolicy = grouping`

Description: (previously sharetype) Defines the mechanism to be used for the grouping of the job transfers. DTR assigns the transfers to shares, so that those shares can be assigned to different priorities. Possible values for `grouping` are dn, voms:vo, voms:role and voms:group:

dn
each job is assigned to a share based on the DN of the user sumbitting the job.

voms:vo
each job is assigned to a share based on the VO specified in the proxy.

voms:role
each job is assigned to a share based on the role specified in the first attribute found in the proxy.

voms:group
each job is assigned to a share based on the group specified in the first attribute found in the proxy.

In case of the voms schemes, if the proxy is not a VOMS proxy, then a default share is used. If sharepolicy is not set then the client-defined priority is applied.

Default: `undefined`

Example:

```
sharepolicy=voms:role
```

Warning: CHANGE: RENAMED in 6.0.0.

sharepriority

Synopsis: `sharepriority = share priority`

Description: (previously definedshare) Defines a share with a fixed priority, different from the default (50). Priority is an integer between 1 (lowest) and 100 (highest).

This option in **multivalued**.

Default: `undefined`

Example:

```
sharepriority=myvo:students 20
sharepriority=myvo:production 80
```

Warning: CHANGE: RENAMED in 6.0.0.

copyurl

Synopsis: `copyurl = url_head local_path`

Description: Configures that DTR should use copy instead of download in case of certain stagein files. URLs, starting from 'url_head' should be accessed in a different way (most probaly unix open). The 'url_head' part of the URL will be replaced with 'local_path' and file from obtained path will be copied to the session directory. NOTE: 'local_path' can also be of URL type.

This option in **multivalued**.

Default: `undefined`

Example:

```
copyurl=gsiftp://example.org:2811/data/ /data/
copyurl=gsiftp://example2.org:2811/data/ /data/
```

linkurl

Synopsis: `linkurl = url_head local_path [node_path]`

Description: Identical to 'copyurl', configures DTR so that for certain URLs files won't be downloaded or copied (in case of copyurl), but soft-link will be created. The 'local_path' specifies the way to access the file from the frontend, and is used to check permissions. The 'node_path' specifies how the file can be accessed from computing nodes, and will be used for soft-link creation. If 'node_path' is missing - 'local_path' will be used.

This option in **multivalued**.

Default: `undefined`

Example:

```
linkurl=gsiftp://somewhere.org/data /data
linkurl=gsiftp://example.org:2811/data/ /scratch/data/
```

use_remote_acix

Synopsis: `use_remote_acix = URL`

Description: (previously acix_endpoint) If configured then the ARC Cache Index, available at the URL, will be queried for every input file specified in a job description and any replicas found in sites with accessible caches will be added to the replica list of the input file. The replicas will be tried in the order specified by preferredpattern variable.

Default: `undefined`

Example:

```
use_remote_acix=https://cacheindex.ndgf.org:6443/data/index
```

Warning: CHANGE: RENAMED in 6.0.0.

preferredpattern

Synopsis: `preferredpattern = pattern`

Description: specifies a preferred pattern on which to sort multiple replicas of an input file. It consists of one or more patterns separated by a pipe character (|) listed in order of preference. Replicas will be ordered by the earliest match. If the dollar character ($) is used at the end of a pattern, the pattern will be matched to the end of the hostname of the replica. If an exclamation mark (!) is used at the beginning of a pattern, any replicas matching the pattern will be excluded from the sorted replicas.

Default: `undefined`

Example:

```
preferredpattern=srm://myhost.ac.uk|.uk$|ndgf.org$|badhost.org$
```

The following options are used to configure multi-host data staging deployment scenario. In that setup a couple of additional data staging boxes are enabled to off-load data transfers.

deliveryservice

Synopsis: `deliveryservice = URL`

Description: The URL to a remote data delivery service which can perform remote data staging.

Default: `undefined`

Example:

```
deliveryservice=https://myhost.org:443/datadeliveryservice
```

localdelivery

Synopsis: `localdelivery = yes/no`

Description: If any deliveryservice is defined, this option determines whether local data transfer is also performed.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
localdelivery=yes
```

remotesizelimit

Synopsis: `remotesizelimit = size`

Description: Lower limit on file size (in bytes) of files that remote hosts should transfer. Can be used to increase performance by transferring small files using local processes.

Default: `undefined`

Example:

```
remotesizelimit=100000
```

[arex/ws] block

A-REX exposes a set of Web Service interfaces that can be used to create and manage jobs, obtain information about the CE and the jobs, handle delegations, access cache information, so on. Comment out this block if you don't want to provide WS-interfaces for various A-REX functionalities.

> **Warning:** CHANGE: NEW block in 6.0.0. Most of the parameters originates from the old `[grid-manager]` block

wsurl

Synopsis: `wsurl = url`

Description: (previously arex_mount_point) Specifies the base URL under which the web service intrefaces will be available. The URL argument must be a full URL consisting of protocol+host+port+path: e.g. https://<hostname>:<port>/<path> Make sure the chosen port is not blocked by firewall or other security rules.

Default: `https://$VAR{[common]hostname}:443/arex`

Example:

```
wsurl=https://piff.hep.lu.se:443/arex
```

> **Warning:** CHANGE: RENAMED and MODIFIED in 6.0.0.

logfile

Synopsis: `logfile = path`

Description: (previously wslogfile) Specify log file location for WS-interface operations.

Default: `/var/log/arc/ws-interface.log`

Example:

```
logfile=/var/log/arc/ws-interface.log
```

> **Warning:** CHANGE: RENAMED in 6.0.0.

pidfile

Synopsis: `pidfile = path`

Description: Specify location of file containing PID of daemon process.

Default: `/run/arched-arex-ws.pid`

Example:

```
pidfile=/run/arched-arex-ws.pid
```

> **Warning:** CHANGE: INTRODUCED in 6.13.0.

max_job_control_requests

Synopsis: `max_job_control_requests = number`

Description: The max number of simultaneously processed job management requests over WS interface - like job submission, cancel, status check etc.

Default: `100`

Example:

```
max_job_control_requests=100
```

max_infosys_requests

Synopsis: `max_infosys_requests = number`

Description: The max number of simultaneously processed info requests over WS interface.

Default: `1`

Example:

```
max_infosys_requests=1
```

max_data_transfer_requests

Synopsis: `max_data_transfer_requests = number`

Description: The max number of simultaneously processed data transfer requests over WS interface - like data staging.

Default: `100`

Example:

```
max_data_transfer_requests=100
```

tlsciphers

Synopsis: `tlsciphers = ciphers_list`

Description: Override OpenSSL ciphers list enabled on server

Default: `HIGH:!eNULL:!aNULL`

Example:

```
tlsciphers=HIGH:!eNULL:!aNULL
```

Warning: CHANGE: INTRODUCED in 6.14.0.

tlsserverorder

Synopsis: `tlsserverorder = yes`

Description: Force priority order of ciphers for TLS connection to be decided on server sid

Default: `no`

Example:

```
tlsserverorder=yes
```

Warning: CHANGE: INTRODUCED in 7.0.0.

tlsprotocols

Synopsis: `tlsprotocols = SSL/TLS protocols`

Description: Specify which protocols to enable This is space separated list of values - SSLv2 SSLv3 TLSv1.0 TLSv1.1 TLSv1.2 TLSv1.3

Default: `TLSv1.2 TLSv1.3`

Example:

```
tlsprotocols=TLSv1.2 TLSv1.3
```

Warning: CHANGE: INTRODUCED in 6.14.0.

tlscurve

Synopsis: `tlscurve = curve`

Description: Specify SSL/TLS ECDH curve name (SN)

Default: `secp521r1`

Example:

```
tlscurve=secp521r1
```

Warning: CHANGE: INTRODUCED in 6.14.0.

[arex/ws/jobs] block

AREX offers a set of web service interfaces implented via either REST or WS-SOAP. This block enables the job management, info query, delegation protocols through both the REST and EMIES interface. Consult the http://svn.nordugrid.org/repos/nordugrid/doc/trunk/tech_doc/emi-es/EMI-ES-Specification_v1.16.pdf for EMIES inteface description and read the http://www.nordugrid.org/documents/arc6/tech/rest/rest.html for the REST interface specification.

Warning: CHANGE: NEW sub-block in 6.0.0.

allownew

Synopsis: `allownew = yes/no`

Description: The 'allownew' config parameter sets if the Computing Element accepts submission of new jobs via the WS-interface. This parameter can be used to close down the CE.

Allowed values: `yes`, `no`

Default: `yes`

Example:

```
allownew=yes
```

> **Warning:** CHANGE: NEW in 6.0.0. Implement support in the code for WS-interface.

allownew_override

Synopsis: `allownew_override = [authgroup ...]`

Description: (previously allowsubmit) Defines which authorization groups are allowed to submit new jobs via the WS-interfaces when the CE is closed with allownew=no. Note that it requires the allownew=no to be set.

This option in **multivalued**.

Default: `undefined`

Example:

```
allownew_override=biousers atlasusers
allownew_override=yourauthgroup
```

> **Warning:** CHANGE: RENAMED and MODIFIED behaviour in 6.0.0. Modified behaviour: possible to specify several groups on a single line!

allowaccess

Synopsis: `allowaccess = authgroup`

Description: (previously groupcfg) Defines that the specified authgroup members are authorized to access the ARC-CE via this interface. A related config option the `denyaccess` (see below) can be used to reject access. Multiple `allowaccess` and `denyaccess` authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

Default: `undefined`

This option in **multivalued**.

Example:

```
allowaccess=biousers
allowaccess=atlasusers
```

denyaccess

Synopsis: `denyaccess = authgroup`

Description: Defines that the specified authgroup members are REJECTED, not authorized to access the ARC-CE via this interface. Note that a related config option the `allowaccess` (see above) can be used to grant access. Multiple `denyaccess` and `allowaccess` authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

Default: `undefined`

This option in **multivalued**.

Example:

```
denyaccess=blacklisted-users
```

maxjobdesc

Synopsis: `maxjobdesc = size`

Description: specifies maximal allowed size of job description in bytes. Default value is 5MB. Use 0 to set unlimited size.

Default: `5242880`

Example:

```
maxjobdesc=0
```

Warning: CHANGE: NEW in 6.0.0.

[arex/ws/publicinfo] block

AREX allows access to public informaton for non-authorized users. Presence of this block enables such feature.

Warning: CHANGE: NEW block in 6.7.0.

allowaccess

Synopsis: `allowaccess = authgroup`

Description: Defines that the specified authgroup members are authorized to access public information. For more information see similar configuration option in `[arex/ws/jobs]` block.

Default: `undefined`

This option in **multivalued**.

Example:

```
allowaccess=monitors
```

Warning: CHANGE: NEW in 6.7.0.

denyaccess

Synopsis: `denyaccess = authgroup`

Description: Defines that the specified authgroup members are REJECTED, not authorized to access public information. For more information see similar configuration option in `[arex/ws/jobs]` block.

Default: `undefined`

This option in **multivalued**.

Example:

```
denyaccess=badactors
```

Warning: CHANGE: NEW in 6.7.0.

[arex/ws/cache] block

The content of the A-REX cache can be accessed via a WS-interface. Configuring this block will allow reading cache files through a special URL. For example, if the remote file gsiftp://remotehost/file1 is stored in the cache and the WS interfaces (configured above) are available via wsurl of https://hostname:443/arex/, then the cached copy of the file can be access via the following special URL: https://hostname:443/arex/cache/gsiftp://remotehost/file1 Comment out this block if you don't want to expose the cache content via WS-interface.

Warning: CHANGE: NEW block in 6.0.0.

cacheaccess

Synopsis: `cacheaccess = rule`

Description: This parameter defines the access control rules for the cache wsinterface, the rules for allowing access to files in the cache remotely through the A-REX web interface. If not set, then noone can access anything. The default is not set that means complete denial. A rule has three parts:

1. Regular expression defining a URL pattern
2. Credential attribute to match against a client's credential
3. Regular expression defining a credential value to match against a client's credential

A client is allowed to access the cached file if a URL pattern matches the cached file URL and the client's credential has the attribute and matches the value required for that pattern. Possible values for credential attribute are dn, voms:vo, voms:role and voms:group.

This option in **multivalued**.

Default: `undefined`

Example:

```
cacheaccess=gsiftp://host.org/private/data/.* voms:vo myvo:production
cacheaccess=gsiftp://host.org/private/data/bob/.* dn /O=Grid/O=NorduGrid/.*
```

[arex/ws/candypond] block

The CandyPond (Cache and deliver your pilot on-demand data) A-REX Web Service (previously called Cache Service) exposes various useful data-staging related operations for the pilot job model where input data for jobs is not known until the job is running on the worker node. This service is intended to be used by A-REX managed jobs. This service requires the `[arex/data-staging]` functionality.

The CandyPond service is available via the wsurl/candypond URL (e.g. https://hostname:443/arex/candypond)

> **Warning:** CHANGE: NEW block in 6.0.0, and RENAMED service.

[arex/ws/argus] block

The Web Service components of A-REX may directly use the Argus service (https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework) for requesting authorization decisions and performing client mapping to a local user account. This block turns on and configures the A-REX WS - Argus integration. When this block is enabled, A-REX will communicate to Argus PEP or PDP service for every WS interface operation! Comment out this block if you don't intend to use any external Argus service with A-REX WS interfaces. Using Argus with gridftp interface is possible only via LCMAPS callout.

> **Warning:** CHANGE: NEW block in 6.0.0.

arguspep_endpoint

Synopsis: `*arguspep_endpoint = url`

Description: Specifies URL of Argus PEPD service to use for authorization and user mapping. It is worth to mention that `requireClientCertAuthentication` (default is false) item of pepd.ini (configuration of Argus PEPD service) is set to be 'true', then https should be used, otherwise http is proper.

> **Note:** Argus will be contacted for every WS interface operation requested!

Default: `undefined`

Example:

```
arguspep_endpoint=https://somehost.somedomain:8154/authz
```

arguspep_profile

Synopsis: `arguspep_profile = profile_name`

Description: Defines which communication profile to use while communicating with Argus PEPD service. Possible values for profile_name are:

direct - pass all authorization attributes (only for debugging) subject - pass only subject name of client
emi - ARC native profile developed in EMI project. This is default option.

Allowed values: `direct`, `subject`, `emi`

Default: `emi`

Example:

```
arguspep_profile=emi
```

Warning: CHANGE: MODIFIED parameter values in 6.0.0: removed CREAM.

arguspep_usermap

Synopsis: `arguspep_usermap = yes/no`

Description: Specifies whether response from Argus service may define mapping of client to local account. Default is 'no'. Note that Argus is contacted after all the other user mapping is performed. Hence it can overwrite all other decisions.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
arguspep_usermap=no
```

arguspdp_endpoint

Synopsis: `arguspdp_endpoint = url`

Description: Specifies URL of Argus PDP service to use for authorization and user mapping. It is worth to mention that `requireClientCertAuthentication` (default is false) item of pdp.ini (configuration of Argus PDP service) is set to be 'true', then https should be used, otherwise http is proper.

Note: Argus will be contacted for every WS interface operation requested!

Default: `undefined`

Example:

```
arguspdp_endpoint=https://somehost.somedomain:8152/authz
```

arguspdp_profile

Synopsis: `arguspdp_profile = profile_name`

Description: Defines which communication profile to use while communicating with Argus PDP service. Possible values for profile_name are:

> subject - pass only subject name of client emi - ARC native profile developed in EMI project. This is default option.

Allowed values: `subject`, `emi`

Default: `emi`

Example:

```
arguspdp_profile=emi
```

Warning: CHANGE: MODIFIED parameter values in 6.0.0: remove CREAM.

arguspdp_accpetnotapplicable

Synopsis: `arguspdp_accpetnotapplicable = yes/no`

Description: Specify if the `NotApplicable` decision returned by Argus PDP service is treated as reason to deny request. Default is 'no', which treats `NotApplicable` as reason to deny request.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
arguspdp_acceptnotapplicable=no
```

[arex/jura] block

A-REX is responsible for collecting accounting measurements from various ARC subsystems, including batch system backends and DTR data staging.

Staring from 6.4 release A-REX writes all accounting data into the local accounting database that can be queries with `arcctl accounting`.

JURA is the accounting record generating and reporting ARC CE module. A-REX periodically executes JURA to create usage records based on the accounting target configuration and using accounting data.

Note: in the ARC releases before 6.4 the accounting measurements were collected in the job log files that are only used for publishing records. This functionality was supported for backward compatibility in 6.4-6.7 releases and was removed in ARC 6.8.0

Enable and configure this block if you want to send accounting records to accounting services.

Note that a dedicated `accounting target` subblock is needed for every accounting destination. The target subblocks are either of a type `apel` or `sgas`: `[arex/jura/apel:targetname]` or `[arex/jura/sgas:targetname]`.

logfile

Synopsis: `logfile = path`

Description: (previously jobreport_logfile) The name of the jura logfile.

Default: `/var/log/arc/jura.log`

Example:

```
logfile=/var/log/arc/jura.log
```

loglevel

Synopsis: `loglevel = number`

Description: Log level for the JURA accounting module.

Allowed values: `0`, `1`, `2`, `3`, `4`, `5`, `FATAL`, `ERROR`, `WARNING`, `INFO`, `VERBOSE`, `DEBUG`

Default: `3`

Example:

```
loglevel=3
```

vomsless_vo

Synopsis: `vomsless_vo = voname[#voissuer]`

Description: This parameter allows the sysadmin to manually assign VOs during pubishing to jobs that were submitted with `VOMS-less grid proxies`. `voname` is the VO name to be used in the generated records (the same as expected in voms-proxy) optional `voissuer` (relevant to SGAS only) value is the VOMS server identity (certificate DN).

Default: `undefined`

Example:

```
vomsless_vo=atlas
vomsless_vo=atlas#/DC=ch/DC=cern/OU=computers/CN=lcg-voms.cern.ch
```

vo_group

Synopsis: `vo_group = group`

Description: Adds an additional VO group attribute(s) to the usage records.

This option in **multivalued**.

Default: `undefined`

Example:

```
vo_group=/atlas/production
```

urdelivery_frequency

Synopsis: `urdelivery_frequency = seconds`

Description: (previously jobreport_period) Specifies the frequency of JURA process regular execution by the A-REX. The actual treshold of records reporting frequency can be defined on per-target basis.

Default: `3600`

Example:

```
urdelivery_frequency=3600
```

Warning: CHANGE: RENAMED in 6.0.0.

x509_host_key

Synopsis: `x509_host_key = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_key}`

Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

Warning: CHANGE: NEW in 6.0.0.

x509_host_cert

Synopsis: `x509_host_cert = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_cert}`

Example:

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

Warning: CHANGE: NEW in 6.0.0.

x509_cert_dir

Synopsis: `x509_cert_dir = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_cert_dir}`

Example:

```
x509_cert_dir=/etc/grid-security/certificates
```

Warning: CHANGE: NEW in 6.0.0.

[arex/jura/sgas:targetname] block

An SGAS sub-block of `[arex/jura]` enables and configures an SGAS accounting server as a target destination to which JURA will send properly formatted usage records. You need to define a separate block with a unique targetname for every SGAS target server.

Note that the block name will be used by JURA to track that latest records sent to this targed. Be aware that ff you rename the block, target will be handled as a new one. However `targeturl` change will not trigger a new target handling.

targeturl

Synopsis: `*targeturl = url`

Description: The service endpoint URL of SGAS server.

Default: `undefined`

Example:

```
targeturl=https://grid.uio.no:8001/logger
```

Warning: CHANGE: NEW in 6.0.0.

localid_prefix

Synopsis: `localid_prefix = prefix_string`

Description: Sets a prefix value for the LocalJobID ur parameter for the SGAS usage records.

Default: `undefined`

Example:

```
localid_prefix=some_text_for_SGAS
```

Warning: CHANGE: NEW in 6.0.0.

vofilter

Synopsis: `vofilter = vo`

Description: Configures a job record filtering mechanism based on the VO attribute of the jobs. Only the matching job records, which was one of VO that you set here, will be sent to the target accounting service.

This option in **multivalued**.

Default: `undefined`

Example:

```
vofilter=atlas
vofilter=fgi.csc.fi
```

Warning: CHANGE: NEW in 6.0.0.

urbatchsize

Synopsis: `urbatchsize = number`

Description: JURA sends usage records not one-by-one, but in batches. This options sets the size of a batch. Zero value means unlimited batch size.

Default: `50`

Example:

```
urbatchsize=80
```

urdelivery_frequency

Synopsis: `urdelivery_frequency = seconds`

Description: (introduced in 6.4.0) Add optional minimal treshold of the interval between subsequent records publishing to this target.

Note: the actual delivery interval is the value divisible by `urdelivery_frequency` defined in `[arex/jura]` block that define the entire JURA process invocation frequency.

Default: `undefined`

Example:

```
urdelivery_frequency=3600
```

Warning: CHANGE: NEW in 6.4.0.

[arex/jura/apel:targetname] block

An APEL sub-block of `[arex/jura]` enables and configures an APEL accounting server as a target destination to which JURA will send properly formatted usage records. You need to define a separate block with a unique targetname for every APEL target server.

Note that the block name will be used by JURA to track that latest records sent to this targed. Be aware that ff you rename the block, target will be handled as a new one. However `targeturl` change will not trigger a new target handling.

targeturl

Synopsis: `*targeturl = url`

Description: The service endpoint URL of the APEL accounting server.

Default: `undefined`

Example:

```
targeturl=https://msg.argo.grnet.gr
```

topic

Synopsis: `topic = topic_name`

Description: Sets the name of the APEL topic to which JURA will publish the accounting records. AMS destination topic for compute element is 'gLite-APEL'

Default: `gLite-APEL`

Example:

```
topic=/queue/global.accounting.test.cpu.central
```

project

Synopsis: `project = project_name`

Description: Sets the name of the APEL project to use.

Default: `accounting`

Example:

```
project=accounting-nl
```

Warning: CHANGE: NEW in 6.19.0

gocdb_name

Synopsis: `*gocdb_name = name`

Description: Can be used to specify the GOCDB name of the resource. This value would be seen as Site attribute in the generated APEL records.

Default: `undefined`

Example:

```
gocdb_name=GRID_UIO_NO
```

apel_messages

Synopsis: `apel_messages = type`

Description: (introduced in 6.4.0) Define what kind of records JURA will send to APEL services during regular publishing process. Possible cases are: per-job EMI CAR records (`urs`), APEL summary records (`summaries`) or APEL summary records v0.4 (`summaries-v04`) APEL Sync messages are always generated.

Note: on the heavy loaded sites with 10k+ jobs weekly `summaries` generation has a performance penalty in ARC < 6.8. It is advised to use `urs` in this case.

Allowed values: `urs`, `summaries`, `summaries-v04`

Default: `summaries`

Example:

```
apel_messages=urs
```

Warning: CHANGE: NEW in 6.4.0.

vofilter

Synopsis: `vofilter = vo`

Description: Configures a job record filtering mechanism based on the VO attribute of the jobs. Only the matching job records, which was one of VO that you set here, will be sent to the target accounting service.

This option in **multivalued**.

Default: `undefined`

Example:

```
vofilter=atlas
vofilter=fgi.csc.fi
```

urbatchsize

Synopsis: `urbatchsize = number`

Description: JURA sends usage records not one-by-one, but in batches. This options sets the size of a batch. Zero value means unlimited batch size. 500 is recommended to avoid too large messages using AMS

Default: `500`

Example:

```
urbatchsize=500
```

urdelivery_frequency

Synopsis: `urdelivery_frequency = seconds`

Description: (introduced in 6.4.0) Add optional minimal treshold of the interval between subsequent records publishing to this target.

Note: the actual delivery interval is the value divisible by `urdelivery_frequency` defined in `[arex/jura]` block that define the entire JURA process invocation frequency.

APEL recommended value is once per day for `summaries`. Use smaller values for `urs`.

Default: `86000`

Example:

```
urdelivery_frequency=14000
```

Warning: CHANGE: NEW in 6.4.0.

[arex/ganglia] block

This block enables the monitoring of ARC-specific metrics. Earlier versions (ARC < 6.0) relied only on the standalone tool gangliarc, ganglia is now instead integrated into ARC, and gangliarc is obsolete. Note that AREX ganglia (as gangliarc did) depends on an existing ganglia installation, as it sends its metrics to a running gmond process.

> **Warning:** CHANGE: RENAMED block in 6.0.0.

gmetric_bin_path

Synopsis: `gmetric_bin_path = path`

Description: (previously gmetric_exec) The path to gmetric executable.

Default: `/usr/bin/gmetric`

Example:

```
gmetric_bin_path=/usr/local/bin/gmetric
```

> **Warning:** CHANGE: MOVED and RENAMED in 6.0.0 from deleted `[gangliarc]` block.

metrics

Synopsis: `metrics = name_of_the_metrics`

Description: the metrics to be monitored. metrics takes a comma-separated list of one or more of the following metrics: - staging – number of tasks in different data staging states - not yet implemented - cache – free cache space - session – free session directory space - heartbeat – last modification time of A-REX heartbeat - processingjobs – the number of jobs currently being processed by ARC (jobs

between PREPARING and FINISHING states) - not yet implemented

- failedjobs – the number of failed jobs per last 100 finished
- jobstates – number of jobs in different A-REX stages
- all – all of the above metrics

Default: `all`

Allowed values: `staging`, `cache`, `session`, `heartbeat`, `processingjobs`, `failedjobs`, `jobstates`, `all`

Example:

```
metrics=all
```

frequency

Synopsis: `frequency = seconds`

Description: The period between each information gathering cycle, in seconds.

Default: `60`

Example:

```
frequency=300
```

> **Warning:** CHANGE: MODIFIED in 6.0.0. Default increased from 20s to one minute.

[gridftpd] block

This block enables and configures the gridftp server. The usage of the gridftp is twofold in connection with ARC: 1) The server together with its custom jobplugin can be used as a job submission and management interface for an ARC CE. 2) The server with the filedirplugin can be used as a very simplistic storage element. This block configures the common server capabilities. To make the gridftp service functional, you need to enable at least one of the plugin subblocks as well.

user

Synopsis: `user = user[:group]`

Description: Switch to a non root user/group after startup WARNING: Make sure that the certificate files are owned by the user/group specified by this option.

Default: `root:root`

Example:

```
user=grid
```

loglevel

Synopsis: `loglevel = level`

Description: (previously debug) Set log level of the gridftpd daemon, between 0 (FATAL) and 5 (DEBUG). Default is 3 (INFO).

Allowed values: `0`, `1`, `2`, `3`, `4`, `5`, `FATAL`, `ERROR`, `WARNING`, `INFO`, `VERBOSE`, `DEBUG`

Default: `3`

Example:

```
loglevel=2
```

> **Warning:** CHANGE: RENAMED in 6.0.0.

logfile

Synopsis: `logfile = path`

Description: Set logfile location of the gridftp server.

Default: `/var/log/arc/gridftpd.log`

Example:

```
logfile=/var/log/arc/gridftpd.log
```

pidfile

Synopsis: `pidfile = path`

Description: Specify location of file containing PID of daemon process.

Default: `/run/gridftpd.pid`

Example:

```
pidfile=/run/gridftpd.pid
```

port

Synopsis: `port = bindport`

Description: Port to listen on. For gridftp-based job submission strongly adviced to use the default 2811 port because 3rd party clients assume ARC CE using that port.

Default: `2811`

Example:

```
port=2811
```

allowencryption

Synopsis: `allowencryption = yes/no`

Description: (previously encryption) should data encryption be allowed on client request. Encryption is very heavy, therefore the default is no.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
allowencryption=no
```

Warning: CHANGE: RENAMED in 6.0.0.

allowactivedata

Synopsis: `allowactivedata = yes/no`

Description: if no, only passive data transfer is allowed. By default both passive and active data transfers are allowed.

Default: `yes`

Example:

```
allowactivedata=yes
```

maxconnections

Synopsis: `maxconnections = number`

Description: The maximum number of connections accepted by a gridftpd server.

Default: `100`

Example:

```
maxconnections=200
```

defaultbuffer

Synopsis: `defaultbuffer = size`

Description: Defines size of every buffer for data reading/writing. The actual value may decrease if the cumulative size of all buffers exceeds value specified by maxbuffer.

Default: `65536`

Example:

```
defaultbuffer=65536
```

maxbuffer

Synopsis: `maxbuffer = size`

Description: Defines maximal amount of memory in bytes to be allocated for all data reading/writing buffers. Default is 640kB. The number of buffers is $(\max\{3, \min\{41, 2P + 1\}\})$, where P is the parallelism level requested by the client. Hence, even without parallel streams enabled number of buffers will be 3.

Default: `655360`

Example:

```
maxbuffer=655360
```

globus_tcp_port_range

Synopsis: `globus_tcp_port_range = port_range`

Description: In a firewalled environment the software which uses GSI needs to know what ports are available. If not set a random port is selected. These variable are similar to the Globus enviroment variables: GLOBUS_TCP_PORT_RANGE and GLOBUS_UDP_PORT_RANGE.

Default: `9000,9300`

Example:

```
globus_tcp_port_range=9000,12000
```

globus_udp_port_range

Synopsis: `globus_udp_port_range = port_range`

Description: In a firewalled environment the software which uses GSI needs to know what ports are available. If not set a random port is selected. These variable are similar to the Globus enviroment variables: GLOBUS_TCP_PORT_RANGE and GLOBUS_UDP_PORT_RANGE.

Default: `9000,9300`

Example:

```
globus_udp_port_range=9000,12000
```

firewall

Synopsis: `firewall = hostname`

Description: The hostname or IP address to use in response to PASV command instead of the IP address of a network interface of computer.

Default: `undefined`

Example:

```
firewall=my.host.org
```

x509_host_key

Synopsis: `x509_host_key = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_key}`

Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

Warning: CHANGE: RENAMED in 6.0.0.

x509_host_cert

Synopsis: `x509_host_cert = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_cert}`

Example:

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

> **Warning:** CHANGE: RENAMED in 6.0.0.

x509_cert_dir

Synopsis: `x509_cert_dir = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_cert_dir}`

Example:

```
x509_cert_dir=/etc/grid-security/certificates
```

[gridftpd/jobs] block

The jobplugin of the gridftp server implements a custom job management and submission interface of ARC CE. This subblock enables and configures that interface. Consult the Technical Reference within the ARC sysadmin guide for the interface specification. Comment out this subblock if you don't want a gridftp-based jobinterface.

allownew

Synopsis: `allownew = yes/no`

Description: This parameter sets if the ARC CE accepts submission of new jobs via the gridftp interface. This parameter can be used to close down the ARC CE.

Allowed values: `yes`, `no`

Default: `yes`

Example:

```
allownew=yes
```

allownew_override

Synopsis: `allownew_override = [authgroup ...]`

Description: (previously allowsubmit) Defines which authorization groups are allowed to submit new jobs via the gridftp interface when the CE is closed with allownew=no. Note that it requires the allownew=no to be set.

This option in **multivalued**.

Default: `undefined`

Example:

```
allownew_override=biousers atlasusers
allownew_override=yourauthgroup
```

> **Warning:** CHANGE: RENAMED and MODIFIED in 6.0.0. Possible to specify several groups on a single line.

allowaccess

Synopsis: `allowaccess = authgroup`

Description: (previously groupcfg) Defines that the specified authgroup members are authorized to access the ARC-CE via this interface. A related config option the `denyaccess` (see below) can be used to reject access. Multiple `allowaccess` and `denyaccess` authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

Default: `undefined`

This option in **multivalued**.

Example:

```
allowaccess=biousers
allowaccess=atlasusers
```

denyaccess

Synopsis: `denyaccess = authgroup`

Description: Defines that the specified authgroup members are REJECTED, not authorized to access the ARC-CE via this interface. Note that a related config option the `allowaccess` (see above) can be used to grant access. Multiple `denyaccess` and `allowaccess` authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

Default: `undefined`

This option in **multivalued**.

Example:

```
denyaccess=blacklisted-users
```

maxjobdesc

Synopsis: `maxjobdesc = size`

Description: specifies maximal allowed size of job description in bytes. Default value is 5MB. Use 0 to set unlimited size.

Default: `5242880`

Example:

```
maxjobdesc=0
```

[gridftpd/filedir] block

The filedirplugin module of the gridftp server can be used to set up a simplistic grid storage element (SE). This subblock enables and configures such an SE by `exporting` a directory using the gridftpd's filedirplugin. Comment out this block if you don't need a SE.

path

Synopsis: `*path = virtdir`

Description: The name of the virtual directory served by the gridftp server. The exported storage area is accessible as `gsiftp://my_server/virtdir`. Even `/` is a valid choice.

Default: `undefined`

Example:

```
path=/topdir
```

mount

Synopsis: `*mount = path`

Description: The physical directory corresponding to the virtual one: gsiftp://my_server/virtdir will give access to this location.

Default: `undefined`

Example:

```
mount=/scratch/grid
```

allowaccess

Synopsis: `allowaccess = authgroup`

Description: (previously groupcfg) Defines that the specified authgroup members are authorized to access the gridftp file service. A related config option the `denyaccess` (see below) can be used to reject access. Multiple `allowaccess` and `denyaccess` authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

Default: `undefined`

This option in **multivalued**.

Example:

```
allowaccess=biousers
allowaccess=atlasusers
```

denyaccess

Synopsis: `denyaccess = authgroup`

Description: Defines that the specified authgroup members are REJECTED, not authorized to access the gridftp file service. Note that a related config option the `allowaccess` (see above) can be used to grant access. Multiple `denyaccess` and `allowaccess` authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

Default: `undefined`

This option in **multivalued**.

Example:

```
denyaccess=blacklisted-users
```

dir

Synopsis: `dir = path options`

Description: Specifies access rules for accessing files in `path` (relative to virtual and real path) and all the files and directories below. Available permisisons check `options` are:

nouser
: do not use local file system rights, only use those specifies in this line

owner
: check only file owner access rights

group
: check only group access rights

other
: check only `others` access rights

If none of the above specified usual unix access rights are applied. Avaiable permissions enforecement `options` are:

read
: allow reading files

delete
: allow deleting files

append
: allow appending files (does not allow creation)

overwrite
: allow overwriting already existing files (does not allow creation, file attributes are not changed)

dirlist
: allow obtaining list of the files

cd
allow to make this directory current

create owner:group permissions_or:permissions_and
allow creating new files. File will be owned by `owner` and owning group will be `group`. If `*` is used, the user/group to which connected user is mapped will be used. The permissions will be set to `permissions_or & permissions_and`. (second number is reserved for the future usage).

mkdir owner:group permissions_or:permissions_and
allow creating new directories.

Example shows setting permissions on mounted `/` directory and adjusting permissions on `/section1` and `/section2` subdirectories.

This option in **multivalued**.

Default: `undefined`

Example:

```
dir=/ nouser read cd dirlist delete create *:* 664:664 mkdir *:* 775:775
dir=/section1 nouser read mkdir *:* 700:700 cd dirlist
dir=/section2 nouser read mkdir *:* 700:700 cd dirlist
```

[infosys] block

This block enables and configures the core part of the information system. Enables the information collection to be used by other ARC components, including interfaces. Parameters in this block applies to all the infosys subsystems.

logfile

Synopsis: `logfile = path`

Description: (previously providerlog) Specifies log file location for the information provider scripts.

Default: `/var/log/arc/infoprovider.log`

Example:

```
logfile=/var/log/arc/infoprovider.log
```

Warning: CHANGE: RENAMED in 6.0.0.

loglevel

Synopsis: `loglevel = number`

Description: (previously provider_loglevel) The loglevel for the infoprovider scripts (0-5). Each value corresponds to the following verbosity levels: FATAL => 0, ERROR => 1 , WARNING => 2, INFO => 3, VERBOSE => 4, DEBUG => 5

Allowed values: `0`, `1`, `2`, `3`, `4`, `5`, `FATAL`, `ERROR`, `WARNING`, `INFO`, `VERBOSE`, `DEBUG`

Default: `3`

Example:

```
loglevel=3
```

Warning: CHANGE: RENAMED in 6.0.0.

validity_ttl

Synopsis: `validity_ttl = seconds`

Description: The published infosys records advertise their validity e.g. how long the info should be considered up-to-date by the clients. Use this parameter to set the published validity value.

Note: different schemas may render this information differently.

Default: `10800`

Example:

```
validity_ttl=10800
```

[infosys/ldap] block

This infosys subblock enables and configures the ldap hosting service for the infosys functionality. Using an LDAP server with some schema is one way to publish information about your Computing Element. Comment out this block if you don't want to run an LDAP-based information system.

Warning: CHANGE: NEW block in 6.0.0.

hostname

Synopsis: `hostname = FQDN`

Description: the hostname of the machine running the slapd service will be the bind for slapd. If not present, will be taken from the `[common]`

Default: `$VAR{[common]hostname}`

Example:

```
hostname=my.testbox
```

slapd_hostnamebind

Synopsis: `slapd_hostnamebind = string`

Description: May be used to set the hostname part of the network interface to which the slapd process will bind. Most of the cases no need to set since the hostname parameter is already sufficient. The example below will bind the slapd process to all the network interfaces available on the server.

Default: `undefined`

Example:

```
slapd_hostnamebind=*
```

port

Synopsis: `port = port_number`

Description: The port on which the slapd service runs. The default infosys port is assumed to be 2135 by many clients, therefore think twice before you change it because 3rd party clients assume 2135 to be the ldap infosys port.

Default: `2135`

Example:

```
port=2135
```

user

Synopsis: `user = unix_user`

Description: overwrites the unix user running the slapd. By default the startup scripts search for well-known ldap-users like `ldap` or `openldap` than fall-back to `root` if not found.

Default: `undefined`

Example:

```
user=slapd
```

Warning: CHANGE: MOVED in 6.0.0 from `[infosys]`.

slapd

Synopsis: `slapd = path`

Description: explicitly define the path to slapd command. By default the startup scripts search for `slapd` binary in the system PATH.

Default: `undefined`

Example:

```
slapd=/usr/sbin/slapd
```

slapd_loglevel

Synopsis: `slapd_loglevel = number`

Description: Sets the native slapd loglevel (see man slapd). Slapd logs via syslog. The default is set to no-logging (0) and it is RECOMMENDED not to be changed in a production environment. Non-zero slap_loglevel value causes serious performance decrease.

Default: `0`

Example:

```
slapd_loglevel=0
```

threads

Synopsis: `threads = number`

Description: The native slapd threads parameter, default is 32.

Default: `32`

Example:

```
threads=128
```

timelimit

Synopsis: `timelimit = seconds`

Description: The native slapd timelimit parameter. Maximum number of seconds the slapd server will spend answering a search request. Default is 3600. You probably want a much lower value.

Default: `3600`

Example:

```
timelimit=1800
```

idletimeout

Synopsis: `idletimeout = seconds`

Description: The native slapd idletimeout parameter. Maximum number of seconds the slapd server will wait before forcibly closing idle client connections. It's value must be larger than the value of `timelimit` option. If not set, it defaults to timelimit + 1.

Default: `$EVAL{$VAR{timelimit} + 1}`

Example:

```
idletimeout=1801
```

infosys_ldap_run_dir

Synopsis: `infosys_ldap_run_dir = path`

Description: The location where NorduGrid/GLUE2 LDAP ldif file will be generated, and where the fifo to sync between infoproviders and BDII will be generated.

Default: `/run/arc/infosys`

Example:

```
infosys_ldap_run_dir=/run/arc/infosys
```

ldap_schema_dir

Synopsis: `ldap_schema_dir = path`

Description: Allows to explicitly specify an additional path to the schema files. Note that this doesn't override standard location, but adds the specified path to the standard locations /etc/ldap and /etc/openldap. Normally it is sufficient to use only standard schema file locations, therefore not to set this parameter.

Default: `undefined`

Example:

```
ldap_schema_dir=/nfs/ldap/schema/
```

The following options configure the third-party bdii ldap parameters. In 99% of cases no need to change anything and use the defaults. These variables are usually automatically set by ARC, and are here mostly for debug purposes and to tweak exotic BDII installations.

bdii_debug_level

Synopsis: `bdii_debug_level = level`

Description: Set this parameter to DEBUG to check bdii errors in bdii-update.log At the same time don't enable slapd logs this way reducing performance issues.

Default: `WARNING`

Example:

```
bdii_debug_level=ERROR
```

bdii_provider_timeout

Synopsis: `bdii_provider_timeout = seconds`

Description: (previously provider_timeout in bdii block) This variable allows a system administrator to modify the behaviour of bdii-update. This is the time BDII waits for the bdii provider scripts generated by A-REX infosys to produce their output.

Default: `10800`

Example:

```
bdii_provider_timeout=10800
```

> **Warning:** CHANGE: RENAMED parameter in 6.0.0.

BDII5 uses these variables. These might change depending on BDII version. ARC sets them by inspecting distributed bdii configuration files. DO NOT

> **Warning:** CHANGE UNLESS YOU KNOW WHAT YOU'RE DOING

bdii_location

Synopsis: `bdii_location = path`

Description: The installation directory for the BDII.

Default: `/usr`

Example:

```
bdii_location=/usr
```

bdii_run_dir

Synopsis: `bdii_run_dir = path`

Description: Contains BDII pid files and slapd pid files

Default: `/run/arc/bdii`

Example:

```
bdii_run_dir=/run/arc/bdii
```

bdii_log_dir

Synopsis: `bdii_log_dir = path`

Description: Contains infosys logs

Default: `/var/log/arc/bdii`

Example:

```
bdii_log_dir=/var/log/arc/bdii
```

bdii_tmp_dir

Synopsis: `bdii_tmp_dir = path`

Description: Contains provider scripts

Default: `/var/tmp/arc/bdii`

Example:

```
bdii_tmp_dir=/var/tmp/arc/bdii
```

bdii_var_dir

Synopsis: `bdii_var_dir = path`

Description: Contains slapd databases

Default: `/var/lib/arc/bdii`

Example:

```
bdii_var_dir=/var/lib/arc/bdii
```

bdii_update_pid_file

Synopsis: `bdii_update_pid_file = path`

Description: Allows to change bdii-update pidfiles filename and location

Default: `$VAR{bdii_run_dir}/bdii-update.pid`

Example:

```
bdii_update_pid_file=/run/arc/bdii/bdii-update.pid
```

bdii_database

Synopsis: `bdii_database = backend_type`

Description: Configure what ldap database backend should be used. If left `undefined` it will default to `hdb` for openldap versions up to 2.4 and to `mdb` for openldap versions 2.5 and later.

Default: `undefined`

Example:

```
bdii_database=hdb
```

bdii_conf

Synopsis: `bdii_conf = path`

Description: Location of the bdii config file generated by ARC.

Default: `$VAR{[infosys/ldap]infosys_ldap_run_dir}/bdii.conf`

Example:

```
bdii_conf=/run/arc/infosys/bdii.conf
```

bdii_update_cmd

Synopsis: `bdii_update_cmd = path`

Description: path to bdii-update script

Default: `$VAR{bdii_location}/sbin/bdii-update`

Example:

```
bdii_update_cmd=/usr/sbin/bdii-update
```

bdii_db_config

Synopsis: `bdii_db_config = path`

Description: path to slapd database configuration file

Default: `/etc/bdii/DB_CONFIG`

Example:

```
bdii_db_config=/etc/bdii/DB_CONFIG
```

bdii_archive_size

Synopsis: `bdii_archive_size = number`

Description: Sets BDII_ARCHIVE_SIZE in bdii configuration file

Default: `0`

Example:

```
bdii_archive_size=0
```

bdii_breathe_time

Synopsis: `bdii_breathe_time = number`

Description: Sets BDII_BREATHE_TIME in bdii configuration file

Default: `10`

Example:

```
bdii_breathe_time=10
```

bdii_delete_delay

Synopsis: `bdii_delete_delay = number`

Description: Sets BDII_DELETE_DELAY in bdii configuration file

Default: `0`

Example:

```
bdii_delete_delay=0
```

bdii_read_timeout

Synopsis: `bdii_read_timeout = number`

Description: Sets BDII_READ_TIMEOUT in bdii configuration file

Default: `$EVAL{$VAR{bdii_provider_timeout} + $VAR{[arex]infoproviders_timelimit} + $VAR{[arex]wakeupperiod}}`

Example:

```
bdii_read_timeout=300
```

Infosys Schema sub-blocks: The following infosys sub-blocks enable information publishing according to various information schema. In order to publish information in a certain schema, the corresponding sub-block must be defined in addition to the schema-neutral `[infosys/cluster]` and `[queue:name]` blocks! Comment out a specific schema block if you don't want to publish a specific information schema representation. Currently available information model (schema) sub-blocks:

> `[infosys/nordugrid]` - The native ARC info representation of a cluster and its queues `[infosys/glue2]` - The GLUE2 information model, both LDAP and XML (the latter is for WS-interface) `[infosys/glue2/ldap]` - The LDAP rendering of the GLUE2 model `[infosys/glue1]` - The legacy GLUE1 model (only LDAP) `[infosys/glue1/site-bdii]` - The site BDII element of the GLUE1 legacy model

[infosys/nordugrid] block

Enables the publication of the NorduGrid information model in the LDAP-based infosys. See the NORDUGRID-TECH-4 for schema definition. The configuration block does not contain any parameter. The information tree is populated based on the contents of the schema-neutral `[infosys/cluster]` and `[queue:name]` blocks.

> **Warning:** CHANGE: NEW block in 6.0.0.

[infosys/glue2] block

Enables the publication of the GLUE2 information model both in the LDAP and XML rendering. The information tree is populated based on the contents of the schema-neutral `[infosys/cluster]` and `[queue:name]` blocks and the GLUE2 specific schema sub-blocks.

> **Warning:** CHANGE: NEW block in 6.0.0.

AdminDomain entity parameters: admindomain_name = string - The Name attribute for the admindomain. This will show in top-BDII to group the resources belonging to this cluster. To group a bunch of clusters under the same AdminDomain, just use the same name. If not specified, will default to UNDEFINEDVALUE.

Default: `UNDEFINEDVALUE`

Example:

```
admindomain_name=ARC-TESTDOMAIN
```

admindomain_description

Synopsis: `admindomain_description = text`

Description: The free-form description of this domain.

Default: `undefined`

Example:

```
admindomain_description=ARC test Domain
```

admindomain_www

Synopsis: `admindomain_www = url`

Description: The URL pointing at a site holding information about the AdminDomain.

Default: `undefined`

Example:

```
admindomain_www=http://www.nordugrid.org/
```

admindomain_distributed

Synopsis: `admindomain_distributed = yes/no`

Description: Set this to yes if the domain is distributed that means, if the resources belonging to the domain are considered geographically distributed.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
admindomain_distributed=yes
```

admindomain_owner

Synopsis: `admindomain_owner = email`

Description: The contact email of a responsible person for the domain

Default: `undefined`

Example:

```
admindomain_owner=admin@nordugrid.org
```

admindomain_otherinfo

Synopsis: `admindomain_otherinfo = text`

Description: Free-form text that fills the OtherInfo GLUE2 field. no need to set, used only for future development.

Default: `undefined`

Example:

```
admindomain_otherinfo=Test Other info
```

ComputingService entity parameters: computingservice_qualitylevel = qlevel - (previously infosys_glue2_service_qualitylevel) Allows a sysadmin to define different GLUE2 QualityLevel values for A-REX. Refer to GLUE2 documentation for the qualitylevel definitions.

Allowed values: `production`, `pre-production`, `testing`, `development`

Default: `production`

Example:

```
computingservice_qualitylevel=production
```

Warning: CHANGE: RENAMED in 6.0.0.s

[infosys/glue2/ldap] block

Enables the publication of the LDAP-rendering of the GLUE2 infomodel. The information tree is populated based on the contents of the schema-neutral `[infosys/cluster]` and `[queue:name]` blocks and the GLUE2 specific schema sub-blocks.

Warning: CHANGE: NEW block in 6.0.0.

showactivities

Synopsis: `showactivities = yes/no`

Description: (previously infosys_glue2_ldap_showactivities) Enables GLUE2 ComputingActivities to appear in the LDAP rendering

Allowed values: `yes`, `no`

Default: `no`

Example:

```
showactivities=no
```

Warning: CHANGE: RENAMED in 6.0.0.

[infosys/glue1] block

This block enables the publication of GLUE1 LDAP representation of a CE. The information tree is populated based on the contents of the schema-neutral `[infosys/cluster]` and `[queue:name]` blocks and the GLUE1 specific schema sub-blocks. This block holds information that is needed by the glue1, in addition to the schema-neutral blocks.

Warning: CHANGE: RENAMED block in 6.0.0.

resource_location

Synopsis: `resource_location = string`

Description: GLUE1 location attribute. IMPORTANT: no slashes or backslashes here!

Default: `undefined`

Example:

```
resource_location=Kastrup, Denmark
```

resource_latitude

Synopsis: `resource_latitude = latitude`

Description: GLUE1 latitude.

Default: `undefined`

Example:

```
resource_latitude=55.75000
```

resource_longitude

Synopsis: `resource_longitude = longitude`

Description: GLUE1 longitude.

Default: `undefined`

Example:

```
resource_longitude=12.41670
```

cpu_scaling_reference_si00

Synopsis: `cpu_scaling_reference_si00 = number`

Description: GLUE1 CPU_scaling

Default: `undefined`

Example:

```
cpu_scaling_reference_si00=2400
```

processor_other_description

Synopsis: `processor_other_description = string`

Description: GLUE1 proc description The text entered under this option is copied `as in` in the glue1 rendering of GlueHostProcessorOtherDescription. Cores is used to calculate the number of physical processors for

GlueSubClusterPhysicalCPUs

The Benchmark value entered here is ignored by the rest of ARC configuration. It will NOT be used for accounting. Note that in ARC > 6.3 the proper way of configuring benchmark for both infosys and accounting is by adding it to the `[queue:name]` blocks. This generic string is kept here for backward compatibility, and most likely will not work as expected (or as used by CREAM).

Default: `undefined`

Example:

```
processor_other_description=Cores=3,Benchmark=9.8-HEP-SPEC06
```

glue_site_web

Synopsis: `glue_site_web = url`

Description: GLUE1 site web url

Default: `undefined`

Example:

```
glue_site_web=http://www.ndgf.org
```

glue_site_unique_id

Synopsis: `glue_site_unique_id = siteid`

Description: GLUE1 site id

Default: `undefined`

Example:

```
glue_site_unique_id=NDGF-T1
```

[infosys/glue1/site-bdii] block

Enable this block ONLY if you want to publish a semi-fake GLUE1 site-bdii as part of the LDAP server. This block is used to configure ARC to generate a semi-fake site-bdii that can be registered in GOCDB. The default will generate an object that you can reference in GOCDB GIIS field as such:

> ldap://archost.domain:2135/mds-vo-name=<unique-site-id>,o=grid Example: ldap://archost.domain:2135/mds-vo-name=NDGF-T1,o=grid

Since the original site-bdii listens on port 2170, you may want to port forward 2170 to 2135 in your firewall, but it is not a requirement. This fake site bdii is only good if you have a single ARC as a service in your organization. It cannot aggregate any information like the real site-bdii does. It just presents a partial rendering of a site-bdii that is good enough for top-bdii to pick up the information. GOCDB checks against this fake-site-bdii should be disabled as it is not fully compliant with the site-bdii specification. They will fail if enabled.

> **Warning:** CHANGE: RENAMED block in 6.0.0.

unique_id

Synopsis: `*unique_id = id`

Description: The unique id used to identify this site, eg `NDGF-T1`

Default: `undefined`

Example:

```
unique_id=NDGF-T1
```

url

Synopsis: `*url = url`

Description: This url is the insertion point in the arc Glue1 ldif tree for the semi-fake site-bdii ldap object. The content of the ldap object is mainly generated from the data in the `[infosys/glue1]` block. The url format is ldap://host.domain:ldapport/basedn

> Example: ldap://host.domain:2135/o=grid

The generated site-bdii information will be attached under the basedn,

> in the example for o=grid: mds-vo-name=unique-id,o=grid

In most cases you want to leave the default untouched, unless you know what you're doing.

Default: `ldap://localhost:2135/o=grid`

Example:

```
url=ldap://host.domain:2135/o=grid
```

[infosys/cluster] block

Information schema-neutral blocks `[infosys/cluster]` and `[queue:NAME]` contain attributes that describe the computing cluster together with its queues. The parameters are available for every information model/schema representation.

This block describes the cluster characteristics of a Computing Element. The information specified here is mostly used by the Infosys ARC component.

alias

Synopsis: `alias = text`

Description: An arbitrary alias name of the cluster, optional.

Default: `undefined`

Example:

```
alias=Big Blue Cluster in Nowhere
```

hostname

Synopsis: `hostname = fqdn`

Description: Set the FQDN of the frontend.

Default: `$VAR{[common]hostname}`

Example:

```
hostname=myhost.org
```

interactive_contactstring

Synopsis: `interactive_contactstring = url`

Description: the contact URL for interactive logins, set this if the cluster supports some sort of grid-enabled interactive login (gsi-ssh),

This option in **multivalued**.

Default: `undefined`

Example:

```
interactive_contactstring=gsissh://frontend.cluster:2200
```

comment

Synopsis: `comment = text`

Description: Free text field for additional comments on the cluster in a single line, no newline character is allowed!

Default: `undefined`

Example:

```
comment=This cluster is specially designed for XYZ applications: www.xyz.org
```

cluster_location

Synopsis: `cluster_location = formatted_string`

Description: The geographical location of the cluster, preferably specified as a postal code with a two letter country prefix

Default: `undefined`

Example:

```
cluster_location=DK-2100
```

cluster_owner

Synopsis: `cluster_owner = text`

Description: It can be used to indicate the owner of a resource, multiple entries can be used

This option in **multivalued**.

Default: `undefined`

Example:

```
cluster_owner=World Grid Project
cluster_owner=University of NeverLand
```

advertisedvo

Synopsis: `advertisedvo = vo_name`

Description: (previously authorizedvo) This attribute is used to advertise which VOs are authorized on the cluster. Add only one VO for each advertisedvo entry. Multiple VOs in the same line will cause errors. These entries will be shown in all GLUE2 AccessPolicy and MappingPolicy objects, that is, they will apply for all Endpoints(Interfaces) and all Shares(currently queues). You can override the advertisedvos per queue. The information is also published in the NorduGrid schema.

Note: it is IMPORTANT to understand that this parameter is NOT enforcing any access control, it is just for information publishing!

This option in **multivalued**.

Default: `undefined`

Example:

```
advertisedvo=atlas
advertisedvo=community.nordugrid.org
```

Warning: CHANGE: RENAMED in 6.0.0.

clustersupport

Synopsis: `clustersupport = email`

Description: This is the support email address of the resource.

This option in **multivalued**.

Default: `undefined`

Example:

```
clustersupport=arc.support@mysite.org
clustersupport=arc.support@myproject.org
```

homogeneity

Synopsis: `homogeneity = True/False`

Description: Determines whether the cluster consists of identical NODES with respect to cputype, memory, installed software (opsys). The frontend is NOT needed to be homogeneous with the nodes. In case of inhomogeneous nodes, try to arrange the nodes into homogeneous groups assigned to a queue and use queue-level attributes. False may trigger multiple GLUE2 ExecutionEnvironments to be published if applicable.

Allowed values: `True`, `False`

Default: `True`

Example:

```
homogeneity=True
```

architecture

Synopsis: `architecture = string`

Description: Sets the hardware architecture of the NODES. The `architecture` is defined as the output of the `uname -m` (e.g. i686). Use this cluster attribute if only the NODES are homogeneous with respect to the architecture. Otherwise the queue-level attribute may be used for inhomogeneous nodes. If the frontend's architecture agrees to the nodes, the `adotf` (Automatically Determine On The Frontend) can be used to request automatic determination.

Default: `adotf`

Example:

```
architecture=adotf
```

opsys

Synopsis: `opsys = formatted_string`

Description: This multivalued attribute is meant to describe the operating system of the computing NODES. Set it to the opsys distribution of the NODES and not the frontend! opsys can also be used to describe the kernel or libc version in case those differ from the originally shipped ones. The distribution name should be given as distroname-version.number, where spaces are not allowed. Kernel version should come in the form kernelname-version.number. If the NODES are inhomogeneous with respect to this attribute do NOT set it on cluster level, arrange your nodes into homogeneous groups assigned to a queue and use queue-level attributes. If opsys=adotf, will result in Automatic Determination of the Operating System On The Frontend, which should only be used if the frontend has the same OS as the nodes. The adotf discovered values will be used to fill GLUE2 OSName, OSVersion and OSFamily unless these values are explicitly defined for each queue. See the `[queue:queuename]` block for their usage. Note: Any custom value other than 'adotf' does NOT affect values in the GLUE2 schema.

This option in **multivalued**.

Default: `adotf`

Example:

```
opsys=Linux-2.6.18
opsys=glibc-2.5.58
opsys=CentOS-5.6
```

nodecpu

Synopsis: `nodecpu = formatted_string`

Description: This is the cputype of the homogeneous nodes. The string is constructed from the /proc/cpuinfo as the value of `model name` and `@` and value of `cpu MHz`. Do NOT set this attribute on cluster level if the NODES are inhomogeneous with respect to cputype, instead arrange the nodes into homogeneous groups assigned to a queue and use queue-level attributes. Setting the nodecpu=adotf will result in Automatic Determination On The Frontend, which should only be used if the frontend has the same cputype as the homogeneous nodes.

Default: `adotf`

Example:

```
nodecpu=AMD Duron(tm) Processor @ 700 MHz
```

nodememory

Synopsis: `nodememory = number`

Description: This is the amount of memory (specified in MB) on the node which can be guaranteed to be available for the application. Please note in most cases it is less than the physical memory installed in the nodes. Do NOT set this attribute on cluster level if the NODES are inhomogeneous with respect to their memories, instead arrange the nodes into homogeneous groups assigned to a queue and use queue-level attributes.

Default: `undefined`

Example:

```
nodememory=64000
```

middleware

Synopsis: `middleware = string`

Description: The multivalued attribute shows the installed grid software on the cluster. Nordugrid-ARC is automatically set, no need to specify

This option in **multivalued**.

Default: `undefined`

Example:

```
middleware=my software
```

nodeaccess

Synopsis: `nodeaccess = inbound/outbound`

Description: Determines how the nodes can connect to the internet. Not setting anything means the nodes are sitting on a private isolated network. `outbound` access means the nodes can connect to the outside world while `inbound` access means the nodes can be connected from outside. inbound & outbound access together means the nodes are sitting on a fully open network.

This option in **multivalued**.

Default: `undefined`

Allowed values: `inbound`, `outbound`

Example:

```
nodeaccess=inbound
nodeaccess=outbound
```

localse

Synopsis: `localse = url`

Description: This multivalued parameter tells the BROKER that certain URLs (and locations below that) should be considered `locally` available to the cluster.

This option in **multivalued**.

Default: `undefined`

Example:

```
localse=gsiftp://my.storage/data1/
localse=gsiftp://my.storage/data2/
```

cpudistribution

Synopsis: `cpudistribution = formatted_string`

Description: This is the CPU distribution over nodes given in the form `ncpu:m` where:

`n` is the number of CPUs per machine `m` is the number of such machines

Example: `1cpu:3,2cpu:4,4cpu:1` represents a cluster with 3 single CPU machines, 4 dual CPU machines and one machine with 4 CPUs.

Default: `undefined`

Example:

```
cpudistribution=1cpu:3,2cpu:4,4cpu:1
```

maxcputime

Synopsis: `maxcputime = number`

Description: This is the maximum CPU time specified in seconds that the LRMS can allocate for the job. The default if not defined is that infoproviders get this value automatically from the LRMS. The purpose of this option is to tweak and override discovered value, or publish this value in case the LRMS module do not support automatic detection.

Default: `undefined`

Example:

```
maxcputime=300000
```

mincputime

Synopsis: `mincputime = number`

Description: This is the minimum CPU time specified in seconds that the LRMS can allocate for the job. The default if not defined is that infoproviders get this value automatically from the LRMS. The purpose of this option is to tweak and override discovered value, or publish this value in case the LRMS module do not support automatic detection.

Default: `undefined`

Example:

```
mincputime=1200
```

maxwalltime

Synopsis: `maxwalltime = number`

Description: This is the maximum Wall time specified in seconds that the LRMS can allocate for the job. The default if not defined is that infoproviders get this value automatically from the LRMS. The purpose of this option is to tweak and override discovered value, or publish this value in case the LRMS module do not support automatic detection.

Default: `undefined`

Example:

```
maxwalltime=600000
```

minwalltime

Synopsis: `minwalltime = number`

Description: This is the minimum Wall time specified in seconds that the LRMS can allocate for the job. The default if not defined is that infoproviders get this value automatically from the LRMS. The purpose of this option is to tweak and override discovered value, or publish this value in case the LRMS module do not support automatic detection.

Default: `undefined`

Example:

```
maxwalltime=1800
```

[queue:name] block

Each grid-enabled queue on the cluster should be represented and described by a separate queue block. The queue_name should be used as a label in the block name. In case of fork, or other LRMSes with no queue names, just use any unique string. A queue can represent a PBS/LSF/SGE/SLURM/LL queue, a SGE pool, a Condor pool or a single machine in case 'fork' type of LRMS. This block describes the queue characteristics.

homogeneity

Synopsis: `homogeneity = True/False`

Description: determines whether the queue consists of identical NODES with respect to cputype, memory, installed software (opsys). In case of inhomogeneous nodes, try to arrange the nodes into homogeneous groups and assigned them to a queue. Possible values: True,False, the default is True.

Allowed values: `True`, `False`

Default: `$VAR{[infosys/cluster]homogeneity}`

Example:

```
homogeneity=True
```

comment

Synopsis: `comment = text`

Description: A free-form text field for additional comments on the queue in a single line, no newline character is allowed!

Default: `undefined`

Example:

```
comment=This queue is nothing more than a condor pool
```

pbs_queue_node

Synopsis: `pbs_queue_node = string`

Description: (previously queue_node_string) In PBS you can assign nodes to a queue (or a queue to nodes) by using the `node property` mark in PBS config.

Essentially, `pbs_queue_node` value is used to construct `nodes=` string in PBS script, such as `nodes=count:pbs_queue_node` where `count` is taken from the job description (1 if not specified).

This corresponds to setting the following parameter in PBS for this queue:

```
resources_default.neednodes = cpu_topology[:pbs_queue_node]
```

Setting the `pbs_queue_node` changes how the queue-totalcpus, user freecpus are determined for this queue.

You shouldn't use this option unless you are sure that your PBS configuration makes use of the above configuration. Read NorduGrid PBS instructions for more information: http://www.nordugrid.org/documents/pbs-config.html

Default: `undefined`

Example:

```
pbs_queue_node=gridlong_nodes
pbs_queue_node=ppn=4:ib
```

> **Warning:** CHANGE: RENAMED in 6.0.0.

sge_jobopts

Synopsis: `sge_jobopts = string`

Description: Per-queue override of additional SGE options to be used when submitting jobs to SGE to this queue

Default: `undefined`

Example:

```
sge_jobopts=-P atlas -r yes
```

condor_requirements

Synopsis: `condor_requirements = string`

Description: It may be defined for each Condor queue. Use this option to determine which nodes belong to the current queue. The value of `condor_requirements` must be a valid constraints string which is recognized by a `condor_status -constraint ...` command. It can reference pre-defined ClassAd attributes (like Memory, Opsys, Arch, HasJava, etc) but also custom ClassAd attributes. To define a custom attribute on a condor node, just add two lines like the ones below in the `$(hostname).local` config file on the node:

```
NORDUGRID_RESOURCE=TRUE
STARTD_EXPRS = NORDUGRID_RESOURCE, $(STARTD_EXPRS)
```

A job submitted to this queue is allowed to run on any node which satisfies the `condor_requirements` constraint. If `condor_requirements` is not set, jobs will be allowed to run on any of the nodes in the pool. When configuring multiple queues, you can differentiate them based on memory size or disk space, for example.

Default: `$VAR{[lrms]condor_requirements}`

Example:

```
condor_requirements=(OpSys == "linux" && NORDUGRID_RESOURCE && Memory >= 1000 &&␣
↪Memory < 2000)
```

slurm_requirements

Synopsis: `slurm_requirements = string`

Description: Use this option to specify extra SLURM-specific parameters.

Default: `undefined`

Example:

```
slurm_requirements=memory on node >> 200
```

Warning: CHANGE: NEW in 6.0.0.

totalcpus

Synopsis: `totalcpus = number`

Description: Manually sets the number of cpus assigned to the queue. No need to specify the parameter in case the queue_node_string method was used to assign nodes to the queue (this case it is dynamically calculated and the static value is overwritten) or when the queue have access to the entire cluster (this case the cluster level totalcpus is the relevant parameter).

Default: `undefined`

Example:

```
totalcpus=32
```

queue-level configuration parameters: nodecpu, nodememory, architecture, opsys should be set if they are homogeneous over the nodes assigned to the queue AND they are different from the cluster-level value. Their meanings are described in the `[infosys/cluster]` block. Usage: this queue collects nodes with `nodememory=512` while another queue has nodes with `nodememory=256` -> don't set the cluster attributes but use the queue-level attributes. When the frontend's architecture or cputype agrees with the queue nodes, the `adotf` (Automatically Determine On The Frontend) can be used to request automatic determination of architecture or nodecpu. For GLUE2, fine tune configuration of ExecutionEnvironments' OSName, OSVersion, OSFamily is allowed with dedicated options osname,osversion,osfamily.

nodecpu

Synopsis: `nodecpu = formatted_string`

Description: see description at `[infosys/cluster]` block

Default: `$VAR{[infosys/cluster]nodecpu}`

Example:

```
nodecpu=AMD Duron(tm) Processor @ 700 MHz
```

nodememory

Synopsis: `nodememory = number`

Description: see description at `[infosys/cluster]` block

Default: `$VAR{[infosys/cluster]nodememory}`

Example:

```
nodememory=512
```

defaultmemory

Synopsis: `defaultmemory = number`

Description: The LRMS memory request of job to be set by the LRMS backend scripts, if a user submits a job without specifying how much memory should be used. The order of precedence is: job description -> `[lrms-defaultmemory]` -> `[queue-defaultmemory]`. This is the amount of memory (specified in MB) that a job will request.

Default: `undefined`

Example:

```
defaultmemory=512
```

Warning: CHANGE: NEW in 6.0.0.

architecture

Synopsis: `architecture = string`

Description: see description at `[infosys/cluster]` block

Default: `$VAR{[infosys/cluster]architecture}`

Example:

```
architecture=adotf
```

opsys

Synopsis: `opsys = formatted_string`

Description: see description at `[infosys/cluster]` block If osname, osversion are present, the values in opsys are ignored.

This option in **multivalued**.

Default: `$VAR{[infosys/cluster]opsys}`

Example:

```
opsys=Linux-2.6.18
opsys=glibc-2.5.58
```

osname

Synopsis: `osname = string`

Description: Only for GLUE2 overrides values defined in opsys for a single ExecutionEnvironment. Configuration of multiple ExecutionEnvironment for the same queue is not supported. Create a different queue for that.

Default: `undefined`

Example:

```
osname=Ubuntu
```

osversion

Synopsis: `osversion = string`

Description: Only for GLUE2 overrides values defined in opsys for a single ExecutionEnvironment. Configuration of multiple ExecutionEnvironment for the same queue is not supported. Create a different queue for that.

Default: `undefined`

Example:

```
osversion=12.04
```

osfamily

Synopsis: `osfamily = string`

Description: Only for GLUE2 overrides values defined in opsys for a single ExecutionEnvironment. Configuration of multiple ExecutionEnvironment for the same queue is not supported. Create a different queue for that.

Default: `undefined`

Example:

```
osfamily=linux
```

benchmark

Synopsis: `benchmark = name value`

Description: Defines resource benchmark results for accounting and information publishing. The nodes in the same queue are assumed to be homogeneous with respect to the benchmark performance.

Note: in ARC < 6.4 this parameter is only used for information publishing.

In case of multiple benchmarks are specified:

- Accounting subsystem will use ONLY THE FIRST defined benchmark.
- Infosys will publish all defined benchmark values.

The values represent per-core CPU performance. Note that APEL accounting services are accept `HEPscore23`, `HEPSPEC` or `Si2k` benchmark types only.

This option in **multivalued**.

Default: `HEPSPEC 1.0`

Example:

```
benchmark=HEPscore23 16.5
benchmark=HEPSPEC 12.26
benchmark=Si2k 3065
```

Warning: CHANGE: MODIFIED IN 6.4.0.

allowaccess

Synopsis: `allowaccess = authgroup`

Description: (previously groupcfg) Defines that the specified authgroup members are authorized to submit jobs to this queue of ARC-CE after the user already granted access to the CE via one of the interfaces. A related config option the `denyaccess` (see below) can be used to deny submission to the queue. Multiple `allowaccess` and `denyaccess` authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then the queue is accessible by everyone already authorized.

Default: `undefined`

This option in **multivalued**.

Example:

```
allowaccess=biousers
allowaccess=atlasusers
```

denyaccess

Synopsis: `denyaccess = authgroup`

Description: Defines that the specified authgroup members are NOT allowed to submit jobs to this queue of ARC-CE after despite the user is already granted access to the CE via one of the interfaces. A related config option the `allowaccess` (see below) can be used to grant job submission to the queue. Multiple `allowaccess` and `denyaccess` authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then the queue is accessible by everyone already authorized.

Default: `undefined`

This option in **multivalued**.

Example:

```
denyaccess=blacklisted-for-the-queue
```

advertisedvo

Synopsis: `advertisedvo = vo_name`

Description: (previously authorizedvo) This attribute is used to advertise which VOs are authorized on the `[queue:name]` of the cluster. Add only one VO for each advertiseddvo entry. Multiple VOs in the same line will cause errors. These entries will be shown in the MappingPolicy objects, that is, they will apply for the Shares that corresponds to the queue. The information is also published in the NorduGrid schema.

Note: if you have also configured `advertisedvo` in the `[infosys/cluster]` block, the result advertised VOs per queue will override whatever is defined in `[infosys/cluster]` block!

Note: it is IMPORTANT to understand that this parameter is NOT enforcing any access control, it is just for information publishing!

This option in **multivalued**.

Default: `$VAR{[infosys/cluster]advertisedvo}`

Example:

```
advertisedvo=atlas
advertisedvo=community.nordugrid.org
```

Warning: CHANGE: RENAMED in 6.0.0.

maxslotsperjob

Synopsis: `maxslotsperjob = number`

Description: This GLUE2 specific parameter configures the MaxSlotsPerJob value on a particular queue. This value is usually generated by LRMS infocollectors, but there are cases in which a system administrator might like to tweak it. Default is to publish what is returned by the LRMS, and if nothing is returned, NOT to publish the MaxSlotsPerJob attribute. If a system administrator sets the value here, that value will be published instead, regardless of what the LRMS returns. Each LRMS might have a different meaning for this value.

Default: `undefined`

Example:

```
maxslotsperjob=5
```

forcedefaultvoms

Synopsis: `forcedefaultvoms = VOMS_FQAN`

Description: specify VOMS FQAN which user will be assigned if his/her credentials contain no VOMS attributes.

Default: `$VAR{[arex]forcedefaultvoms}`

Example:

```
forcedefaultvoms=/vo/group/subgroup
```

Warning: CHANGE: documented in `[queue]` as described in `[arex]`.

maxcputime = number - This value overrides the one defined in the `[infosys/cluster]` block. See description in that block.

Default: `undefined`

maxcputime=300000

mincputime

Synopsis: `mincputime = number`

Description: This value overrides the one defined in the `[infosys/cluster]` block. See description in that block.

Default: `undefined`

Example:

```
mincputime=1200
```

maxwalltime

Synopsis: `maxwalltime = number`

Description: This value overrides the one defined in the `[infosys/cluster]` block. See description in that block.

Default: `undefined`

Example:

```
maxwalltime=600000
```

minwalltime

Synopsis: `minwalltime = number`

Description: This value overrides the one defined in the `[infosys/cluster]` block. See description in that block.

Default: `undefined`

Example:

```
minwalltime=1800
```

[datadelivery-service] block

This block configures and enables the data delivery service. This service is intended to off-load data-staging from A-REX and usually deployed on one or more separate machines.

This service can also act as an independent data transfers service that case it would require an inteligent data manager that could replace A-REX's intelligence.

> **Warning:** CHANGE: NEW block in 6.0.0.

transfer_dir

Synopsis: `*transfer_dir = path`

Description: (previously allowed_dir) The directori(es) on the DDS host in which the service is allowed to read and write. When DDS is used as a remote transfer service assisting A-REX then this is usually one or more cache and/or session directories shared as a common mount with A-REX.

This option in **multivalued**.

Default: `undefined`

Example:

```
transfer_dir=/shared/arc/cache
transfer_dir=/shared/arc/session
```

> **Warning:** CHANGE: NEW in 6.0.0.

hostname

Synopsis: `hostname = FQDN`

Description: The hostname of the machine on which DDS service runs.

Default: `$EXEC{hostname -f}`

Example:

```
hostname=localhost
```

Warning: CHANGE: NEW in 6.0.0.

port

Synopsis: `port = port`

Description: Port on which service listens

Default: `443`

Example:

```
port=8443
```

Warning: CHANGE: NEW in 6.0.0.

pidfile

Synopsis: `pidfile = path`

Description: pid file of the daemon

Default: `/run/arched-datadelivery-service.pid`

Example:

```
pidfile=/run/arched-datadelivery-service.pid
```

Warning: CHANGE: NEW in 6.0.0.

logfile

Synopsis: `logfile = path`

Description: log file of the daemon

Default: `/var/log/arc/datadelivery-service.log`

Example:

```
logfile=/tmp/delivery.log
```

Warning: CHANGE: NEW in 6.0.0.

loglevel

Synopsis: `loglevel = level`

Description: set loglevel of the data delivery service between 0 (FATAL) and 5 (DEBUG). Defaults to 3 (INFO).

Allowed values: `0`, `1`, `2`, `3`, `4`, `5`

Default: `3`

Example:

```
loglevel=4
```

Warning: CHANGE: NEW in 6.0.0.

user

Synopsis: `user = username`

Description: Overwrites the user under which the service runs. The default is the user starting the service. DDS is very limited if not run as root.

Default: `undefined`

Example:

```
user=ddsuser
```

Warning: CHANGE: NEW in 6.0.0.

secure

Synopsis: `secure = yes/no`

Description: Set to `no` if the service should run without a host certificate. In this case the corresponding deliveryservice option in the `[arex/data-staging]` A-REX configuration block should use http rather than https URLs.

Allowed values: `yes`, `no`

Default: `yes`

Example:

```
secure=no
```

Warning: CHANGE: NEW in 6.0.0.

allowed_ip

Synopsis: `*allowed_ip = ip`

Description: IP address authorized to access service. Normally this is the A-REX host IP. By default the delivery service listens on all available interfaces, so if both IPv4 and IPv6 are enabled on this and the A-REX host, remember to add both A-REX host IPs here.

This option in **multivalued**.

Default: `undefined`

Example:

```
allowed_ip=192.0.2.1
allowed_ip=2001:db8:85a3::8a2e:370:7334
```

Warning: CHANGE: NEW in 6.0.0.

allowed_dn

Synopsis: `allowed_dn = DN`

Description: DN authorized to access service. This option restricts access to specified DNs (of the users who submit jobs to A-REX). It is only effective if secure=yes.

This option in **multivalued**.

Default: `undefined`

Example:

```
allowed_dn=/O=Grid/O=Big VO/CN=Main Boss
```

Warning: CHANGE: NEW in 6.0.0.

x509_host_key

Synopsis: `x509_host_key = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_key}`

Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

Warning: CHANGE: NEW in this block in 6.0.0.

x509_host_cert

Synopsis: `x509_host_cert = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_cert}`

Example:

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

> **Warning:** CHANGE: NEW in this block in 6.0.0.

x509_cert_dir

Synopsis: `x509_cert_dir = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_cert_dir}`

Example:

```
x509_cert_dir=/etc/grid-security/certificates
```

> **Warning:** CHANGE: NEW in this block in 6.0.0.

[acix-scanner] block

The ARC Cache Index (ACIX) is a distributed system that maintains a catalog of locations of cached files stored in various A-REX caches. ACIX consists of two components, the Cache Scanner (on the CEs), and the Index Server.

This config block enables and configures the cache scanner component of ACIX. The scanning component of ACIX is a separate service that runs alongside A-REX/DDS and all it needs from A-REX/DDS service is the location of the cache.

The acix-scanner periodically scans the cache directories and composes a Bloom filter of A-REX cache content which can be pulled via its public interface. In the current deployment the ACIX index services are the main consumers of the collected information.

> **Warning:** CHANGE: RENAMED block, RENAMED component in 6.0.0 (cache scanner, instead of cacheserver)

cachedir

Synopsis: `cachedir = cache_path`

Description: Specifies the cache directory to be scanned in case not set in `[arex/cache]` block (e.g. the scanner is running on a different machine then A-REX)

This option in **multivalued**.

Default: `$VAR{[arex/cache]cachedir}`

Example:

```
cachedir=/scratch/cache
cachedir=/fs1/cache drain
```

logfile

Synopsis: `logfile = path`

Description:
Log file location for the acix-scanner.

Default: `/var/log/arc/arc-acix-scanner.log`

Example:

```
logfile=/tmp/arc-acix-scanner.log
```

Warning: CHANGE: MODIFIED default for the logfile.

hostname

Synopsis: `hostname = string`

Description: Hostname on which the acix-scanner listens

Default: `$EXEC{hostname -f}`

Example:

```
hostname=myhost.org
```

port

Synopsis: `port = port`

Description: Port on which the acix-scanner service listens

Default: `5443`

Example:

```
port=6000
```

cachedump

Synopsis: `cachedump = yes/no`

Description: Whether to make a dump of the list of files on the cache at $TMP/ARC-ACIX/timestamp each time the acix-scanner runs.

Default: `no`

Allowed values: `yes`, `no`

Example:

```
cachedump=yes
```

x509_host_key

Synopsis: `x509_host_key = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_key}`

Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

> **Warning:** CHANGE: NEW in this block from 6.0.0.

x509_host_cert

Synopsis: `x509_host_cert = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_cert}`

Example:

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

> **Warning:** CHANGE: NEW in this block from 6.0.0.

x509_cert_dir

Synopsis: `x509_cert_dir = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_cert_dir}`

Example:

```
x509_cert_dir=/etc/grid-security/certificates
```

> **Warning:** CHANGE: NEW in this block from 6.0.0.

[acix-index] block

The ARC Cache Index (ACIX) is a distributed system that maintains a catalog of locations of cached files stored in various A-REX caches. ACIX consists of two components, the Cache Scanner (on the CEs), and the Index Server. This config block enables and configures the index server component of ACIX.

The index server component of ACIX collects cache content filters generated by a set of acix-scanners and maintains an aggregated view of distributed cache contents.

The acix-index server is deployed separately and can be queried for the location of cached files. The service endpoint is https://hostname:6443/data/index and a query is performed via giving the URLs to check as comma-separated values to the `url` option, e.g.: `index_service_endpoint?url=http://www.nordugrid.org:80/data/echo.sh,http://my.host/data1`

cachescanner

Synopsis: `*cachescanner = url`

Description: (previously cacheserver) ACIX cache scanners from which to pull information

This option in **multivalued**.

Default: `undefined`

Example:

```
cachescanner=https://some.host:5443/data/cache
cachescanner=https://another.host:5443/data/cache
```

> **Warning:** CHANGE: RENAMED in 6.0.0.

x509_host_key

Synopsis: `x509_host_key = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_key}`

Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

> **Warning:** CHANGE: NEW in this block from 6.0.0.

x509_host_cert

Synopsis: `x509_host_cert = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_cert}`

Example:

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

> **Warning:** CHANGE: NEW in this block from 6.0.0.

x509_cert_dir

Synopsis: `x509_cert_dir = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_cert_dir}`

Example:

```
x509_cert_dir=/etc/grid-security/certificates
```

> **Warning:** CHANGE: NEW in this block from 6.0.0.

[userlist:name] block

The `[userlist:name]` blocks are used to define userlists and configure how those are generated by the nordugridmap utility, including the optional user mapping information. The userlist is identified with the generated file that is stored in the `outfile`. Please note that behaviour of the nordugridmap external utility can be further modified by the optional `[nordugridmap]` block (see next block). Note that the `[userlist:name]` block by itself does not affect authorization. In order to define auth rules userlists can be referenced within the `[authgroup]` blocks by the userlist=name parameter. Also, the generated `outfile` can be used via the 'file' auth rule of the `[authgroup]` block. The order of this block in `arc.conf` may influence authorization decision. This block must appear before the `[arex]` block.

> **Warning:** CHANGE: RENAMED block in 6.0.0: blockname now better reflects the purpose of the config block. This block is not defining any VO!

outfile

Synopsis: `outfile = path`

Description: (previously file) The full path of the GENERATED file that contains the userlist (with optional mapping info). If the same file specified as output for different `[userlist:name]` blocks then nordugridmap will automatically merge entries following the order of the blocks.

Default: `/etc/grid-security/grid-mapfile`

Example:

```
outfile=/etc/grid-security/lists/atlas-users
```

> **Warning:** CHANGE: RENAMED in 6.0.0.

source

Synopsis: `*source = url`

Description: the URL of the VO database which is used to generate the userlist. The nordugridmap will use this URL to automatically generate and keep up-to-date userlist (mapfile) specified by the 'outfile' attribute.

url is a multivalued attribute, several sources can be specified and all the users from those sources will be merged into the same file. The source URLs are processed in the given order.

Currently supported URL types are:

`http(s)://`
: URL to plain text file. File should contain a list of DNs.

`voms(s)://`
: URL to VOMS-Admin interface

`file://`
: local file (stand-alone or dynamicaly generated by `nordugridmap`). File should contain a list of DNs with optional mapped unixid: `user DN [mapped user ID]` Result of optional mapped unixid processing depend on `mapuser_processing` option settings.

`userlist://`
: reference to another `[userlist/name]` configuration block

`nordugrid`
: add NorduGrid VO members

You can use either `userlist://` or `file://` entries to specify dependencies between `[userlist/name]` blocks, but using `userlist://` is a recommended way.

For each separate source URL it is possible to override some parameters value. You can use the following syntax to perform this:

```
"source=URL < parameter1=value1 parameter2=value2"
```

You can override the following parameters:

`mapped_unixid`
: for http(s),voms(s),ldap and file URLs

`cache_enable`
: for http(s),voms(s),ldap and file URLs

`voms_method`
: for voms(s) URLs

`mapuser_processing`
: for file URLs with mapped_unixid='<unixid>' overrided (control `mapped_unixid` overriding behaviour for URL)

This option in **multivalued**.

Default: `undefined`

Example:

```
source=vomss://voms.ndgf.org:8443/voms/nordugrid.org
source=vomss://lcg-voms.cern.ch:8443/voms/atlas?/atlas/Role=VO-Admin < mapped_unixid=atlasadmin
source=vomss://kuiken.nikhef.nl:8443/voms/gin.ggf.org < voms_method=get
source=http://www.nordugrid.org/developers.dn
source=file:///etc/grid-security/priviliged_users.dn
source=userlist://biousers
source=nordugrid
```

Warning: CHANGE: MODIFIED options in 6.0.0: edg-mkgridmap source option is removed (obsolete technology); vo source option is renamed as userlist.

mapped_unixid

Synopsis: `mapped_unixid = unixid`

Description: The local UNIXID which is optionally used in the generated outfile by the nordugridmap utility.

If any of the sources have already provided mapping information (file:// or userlist://) behavior depends on 'mapuser_processing' from the `[nordugridmap]` block:

`mapuser_processing = overwrite`
: ignore already provided mapping and apply `mapped_unixid` for all sources

`mapuser_processing = keep`
: apply mapped_unixid only for sources that does not already has mapping information

If the `mapped_unixid` config parameter is not specified or has empty value, then behavior depends on the value of `allow_empty_unixid` from the `[nordugridmap]` block:

`allow_empty_unixid = yes`
: empty value will be used for `mapped_unixid` which means that nordugridmap will generate only the list of DNs without mapping (consider using `mapuser_processing = overwrite` along with this option or sources that does not provide previously defined mapping information)

`allow_empty_unixid = no`
: then nordugridmap will skip users without mapping information (if no mapping information provided by sources)

Default: `nobody`

Example:

```
mapped_unixid=
mapped_unixid=gridtest
```

Warning: CHANGE: MODIFIED in 6.0.0. The empty value should work the same way as missing parameter. set the default to `nobody`

voms_fqan_map

Synopsis: `voms_fqan_map = fqan unixid`

Description: The local UNIXID which is used to map voms(s) sources with specific FQAN given.

Several voms_fqan_map can be specified for a `[userlist/name]` block. For each voms(s) sources in `[userlist/name]` block and every voms_fqan_map record separate source record will be automatically generated with mapped_unixid overwritten to specified one.

Sources are generated in a given voms_fqan_map order. Original voms(s) source URL are processed LAST.

This allows to simplify configuration, especially in redundancy cases when several VOMS servers are used for the same VO.

This option in **multivalued**.

Default: `undefined`

Example:

```
voms_fqan_map=/atlas/Role=VO-Admin atlasadmin
voms_fqan_map=/atlas/Role=production atlasprod
```

filter

Synopsis: `filter = ACL string`

Description: An ACL filter for the nordugridmap utility. Multiple allow/deny statements are possible. The fetched DNs are filtered against the specified rules before they are added to the generated outfile.

* can be used as a wildcard. You may run the nordugridmap with the `--test` command line option to see how the filters you specified work.

If at least one allow filter is specified implicit deny is used at the end of ACL. If only deny filters are present - implicit allow used at the end.

This option in **multivalued**.

Default: `undefined`

Example:

```
filter=deny *infn*
filter=allow *NorduGrid*
```

[nordugridmap] block

This optional block is used to fine-tune the behavior of the nordugridmap utility - an ARC tool used to generate grid-mapfiles. Normal setups don't need to configure this block. Please refer to `[userlist/name]` block above to find information how to specify sources for userlist generation. This section setup general source-independent parameters.

x509_host_key

Synopsis: `x509_host_key = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_key}`

Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

Warning: CHANGE: RENAMED in 6.0.0.

x509_host_cert

Synopsis: `x509_host_cert = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_host_cert}`

Example:

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

Warning: CHANGE: RENAMED in 6.0.0.

x509_cert_dir

Synopsis: `x509_cert_dir = path`

Description: Optional parameter to overwrite `[common]` block values.

Default: `$VAR{[common]x509_cert_dir}`

Example:

```
x509_cert_dir=/etc/grid-security/certificates
```

gridmap_owner

Synopsis: `gridmap_owner = username`

Description: The owner of the generated mapfiles.

Default: `root`

Example:

```
gridmap_owner=root
```

gridmap_group

Synopsis: `gridmap_group = groupname`

Description: The group of generated gridmapfiles.

Default: `root`

Example:

```
gridmap_group=root
```

gridmap_permissions

Synopsis: `gridmap_permissions = filemode`

Description: The permissions of generated gridmapfiles.

Default: `0600`

Example:

```
gridmap_permissions=0600
```

log_to_file

Synopsis: `log_to_file = yes/no`

Description: control whether logging output of nordugridmap will be saved to file. If the value is 'no' nordugridmap will write all information to STDERR.

Allowed values: `yes`, `no`

Default: `yes`

Example:

```
log_to_file=no
```

logfile

Synopsis: `logfile = path`

Description: specify the nordugridmap log file location when log_to_file is set yes.

Default: `/var/log/arc/nordugridmap.log`

Example:

```
logfile=/var/log/arc/nordugridmap.log
```

cache_enable

Synopsis: `cache_enable = yes/no`

Description: Controls whether caching of external sources will be used.

Allowed values: `yes`, `no`

Default: `yes`

Example:

```
cache_enable=yes
```

cachedir

Synopsis: `cachedir = path`

Description: Specifies the path where cached sources will be stored.

Default: `/var/spool/arc/gridmapcache/`

Example:

```
cachedir=/var/spool/arc/gridmapcache/
```

cachetime

Synopsis: `cachetime = seconds`

Description: Controls how long (in seconds) the cached information remains valid. Default is 259200 (3 days).

Default: `259200`

Example:

```
cachetime=259200
```

mapuser_processing

Synopsis: `mapuser_processing = owerwrite/keep`

Description: Controls the behavior of `[userlist:name]` block's mapped_unixid parameter usage. Please see 'mapped_unixid' description in `[userlist:name]` block for details.

Allowed values: `keep`, `overwrite`

Default: `keep`

Example:

```
mapuser_processing=keep
```

allow_empty_unixid

Synopsis: `allow_empty_unixid = yes/no`

Description: Controls whether empty (or unspecified) 'mapped_unixid' `[userlist:name]` block option is allowed to be used. Please see 'mapped_unixid' description for details.

Allowed values: `yes`, `no`

Default: `yes`

Example:

```
allow_empty_unixid=no
```

Warning: CHANGE: MODIFIED in 6.0.0.

voms_method

Synopsis: `voms_method = soap/get`

Description: Controls how to get information from VOMS(S) sources. Valid values are:

> soap - call SOAP method directly using SOAP::Lite get - use old implementation that manually parses XML response

Allowed values: `soap`, `get`

Default: `soap`

Example:

```
voms_method=soap
```

loglevel

Synopsis: `loglevel = level`

Description: (previously debug) Controls the verbosity of nordugridmap output. Valid values are:

> 0 - FATAL - only critical fatal error shown 1 - ERROR - errors, including non-critical are shown 2 - WARNING (default) - configuration errors that can be ignored 3 - INFO - processing information 4 - VERBOSE - a bit more processing information 5 - DEBUG - lot of processing information

When test run is requested (–test command line option of the nordugridmap) loglevel is automatically set to 5 (DEBUG).

Allowed values: `0`, `1`, `2`, `3`, `4`, `5`

Default: `2`

Example:

```
loglevel=4
```

Warning: CHANGE: RENAMED in 6.0.0.

fetch_timeout

Synopsis: `fetch_timeout = seconds`

Description: Controls how many time (in seconds) nordugridmap will wait for external sources retrieval.

Default: `15`

Example:

```
fetch_timeout=15
```

[custom:name] block

This optional block is for those who wish to include non-ARC configuration in `arc.conf`. Custom blocks will be ignored by ARC components including the configuration validator. Any non-ARC configuration which is not in a custom block will be flagged as an error by the validator and A-REX will not start.

4.1.3 Configuration options removed in ARC 6.8.0

[arex/jura] block

urdelivery_keepfailed

Synopsis: `urdelivery_keepfailed = days`

Description: (deprecated in 6.4.0) Specifies for how many days JURA will try to send a record to the destination accounting service before it gives up. Records not successfully sent by after the number of days expired will be deleted from the controldir/logs directory. The deleted records are nevertheless archived if archiving was turned on. Scatring from 6.4 all records are stored in the accounting database without expiration time independently of publishing process.

Default: `30`

Example:

```
urdelivery_keepfailed=30
```

Warning: CHANGE: DEPRECATED in 6.4.0

[arex/jura/sgas:targetname] block

legacy_fallback

Synopsis: `legacy_fallback = yes/no`

Description: (introduced in 6.4.0) If set to `yes` then accounting records publishing to this target will be handled with the previous version of code (6.0-6.3) that relies on A-REX job log files. Use this option only if you experience problems with the new Jura.

Default: `no`

Example:

```
legacy_fallback=no
```

Warning: CHANGE: NEW in 6.4.0.

[arex/jura/apel:targetname] block

use_ssl

Synopsis: `use_ssl = yes/no`

Description: (deprecated in 6.4.0) Turns on/off ssl for the SSM communication with APEL server. Use only with ARC version < 6.4.0. In ARC 6.4+ `targeturl` protocol defines the usage of SSL.

Allowed values: `yes`, `no`

Default: `no`

Example:

```
use_ssl=yes
```

> **Warning:** CHANGE: DEPRECATED in 6.4.0.

benchmark_type

Synopsis: `benchmark_type = type`

Description: (deprecated in 6.4.0) Type of benchmark (Si2k, HEPSPEC) to be reported in every UR. Starting from 6.4 no need to specify the benchmark values here as those will be taken from the `[queue]` block `benchmark` option.

Default: `undefined`

Example:

```
benchmark_type=HEPSPEC
```

> **Warning:** CHANGE: DEPRECATED in 6.4.0.

benchmark_value

Synopsis: `benchmark_value = number`

Description: (deprecated in 6.4.0) The value of the of benchmark to be reported in every UR. Starting from 6.4 no need to specify the benchmark values here as those will be taken from the `[queue]` block `benchmark` option.

Default: `undefined`

Example:

```
benchmark_value=2.4
```

> **Warning:** CHANGE: DEPRECATED in 6.4.0.

benchmark_description

Synopsis: `benchmark_description = string`

Description: (deprecated in 6.4.0) Additional description for the benchmark to be reported in every UR.

Default: `undefined`

Example:

```
benchmark_description=some description for benchmark
```

Warning: CHANGE: DEPRECATED in 6.4.0.

legacy_fallback

Synopsis: `legacy_fallback = yes/no`

Description: (introduced in 6.4.0) If set to `yes` then accounting records publishing to this target will be handled with the previous version of code (6.0-6.3) that relies on A-REX job log files. Use this option only if you experience problems with the new Jura.

Default: `no`

Example:

```
legacy_fallback=no
```

Warning: CHANGE: NEW in 6.4.0.

[arex/jura/archiving] block

This block enables the archiving functionality of pre 6.4 jura implementaton

Note: starting from ARC 6.4.0, with the next generation accounting subsystem introduction, this block only takes action if the `legacy_fallback` option is set to `yes` for a target. In 6.4 all accounting data is stored in the database eliminating the need of any additional records archiving.

When enabled the usage records generated by JURA are stored in a the specified `archivedir` directory on the disk.

Stored records are than processed by accounting archive manager that process the records, store data into SQLite accounting archive database, create new archive structure and maintains the accounting database aging.

By default the archiving is turned off.

Warning: CHANGE: DEPRECATED in 6.4.0.

archivedir

Synopsis: `archivedir = path`

Description: Sets the directory path for the jura archived usage records. Usage records generated by jura reporter are stored in this directory with the following file naming convention used:: `usagerecord<Type>.<jobid>.<random>`. Here Type is one of the supported record formats such as OGFUR or CAR.

Default: `/var/spool/arc/jura/archive`

Example:

```
archivedir=/var/spool/arc/jura/archive
```

logfile

Synopsis: `logfile = path`

Description: The name of the accounting database manager logfile.

Default: `/var/log/arc/jura-archive-manager.log`

Example:

```
logfile=/var/log/arc/jura-archive-manager.log
```

archivettl

Synopsis: `archivettl = days`

Description: The records time to live (ttl) parameter sets the number of days to keep the usage record files in the accounting database If not specified the files are kept forever.

Default: `undefined`

Example:

```
archivettl=365
```

4.2 ARC CE Deployment and Operation

4.2.1 Try ARC6: towards distributed computing in a few minutes

Scared of distributed computing complexities?

With ARC6 you can setup a *Computing Element* and try common distributed computing workflows in just a few minutes!

ARC6 comes with so-called *zero configuration* included and works out of the box without any configuration at all.

https://www.youtu.be/wrE36NQM67c

Step 1. Enable NorduGrid ARC6 repos

Prepare your system to install via the *NorduGrid Repositories*.

Note: Alpha and release-candidate packages are in *testing* repository, so please make sure it is enabled, e.g. on RHEL-based systems you can use `yum --enablerepo=nordugrid-testing` to enable it for one transaction or `yum-config-manager --enable nordugrid-testing` to enable permanently.

If you want to test ARC6 including all latest developments, set up your repository to include the nightly builds following *Using ARC packages from nightly builds* instructions.

Or for the latest development of the not yet released ARC 7: Nightly Next Builds

Step 2. Install A-REX

ARC Resource-coupled EXecution service (A-REX) is a core component that manages authentication, authorization and job life cycle. It is enough to have A-REX installed to have a minimal computing element:

```
[root ~]# yum -y install nordugrid-arc-arex
```

Step 3. Run A-REX

To start ARC services just run:

```
[root ~]# arcctl service start --as-configured
```

You can check if A-REX is running with:

```
[root ~]# arcctl service list
arc-acix-index                    (Not installed, Disabled, Stopped)
arc-acix-scanner                  (Not installed, Disabled, Stopped)
arc-arex                          (Installed, Enabled, Running)
arc-datadelivery-service          (Not installed, Disabled, Stopped)
arc-gridftpd                      (Not installed, Disabled, Stopped)
arc-infosys-ldap                  (Not installed, Disabled, Stopped)
```

Note: `arcctl` tool automates many ARC CE operations and is designed with bash-completion in mind. If you would like to use ARC in production it is advised to have completion enabled:

```
[root ~]# yum install bash-completion python-argcomplete
[root ~]# activate-global-python-argcomplete
```

Step 4. Generate user certificate and key for testing

Grid services and users authentication heavily relies on cryptography and uses certificates/keys for each entity. ARC6 comes with Test Certificate Authority on board that can issue the test user certificates easily.

The ARC6 zero configuration implements a *default closed* approach defining the special authorization object called *authgroup*.

During the test-user certificate generation, `arcctl test-ca` will automatically add the issued certificate subject to the `testCA.allowed-subjects` file, opening the job submission possiblity to the test-user transparently. the testCA.allowed-subjects can be found in your `/etc/grid-security` folder.

No other subject will be able to submit to your system before you change the `authgroup` settings in `arc.conf`.

You can test submission from the host running A-REX or from any other host in the network.

Testing from the host running A-REX

It is technically possible to submit jobs from the `root` account, however it is advised to use a dedicated regular user. Here we assume that you use a dedicated regular user.

To generate test certificate/key and install it to standard location inside local user's home directory run:

```
[root ~]# arcctl test-ca usercert --install-user user01
User certificate and key are installed to default /home/user01/.globus location for␣
↪user user01.
```

Testing from any other host

In order to submit jobs from any other host (not the one running A-REX) you need to transfer the (test) user certificate and the CA-files to this other host.

On the **A-REX host** generate a user certificate/key:

```
[root ~]# arcctl test-ca usercert --export-tar
User certificate and key are exported to testcert-09160712.tar.gz.
To use it with arc* tools on the other machine, copy the tarball and run the␣
↪following commands:
  tar xzf testcert-09160712.tar.gz
  source arc-test-certs/setenv.sh
```

Transfer the tarball to the client host and on the client host execute the commands suggested in the `arcctl` output:

```
[user ~]$ tar xzf /tmp/testcert-09160712.tar.gz
[user ~]$ source arc-test-certs/setenv.sh
```

Note: The zero configured A-REX comes with EMI-ES and REST interfaces enabled. It runs on port `443`, so make sure it is not firewalled to be able to be used from another client host.

Step 5. Submit job and check it is running

Install ARC client tools on the client host:

```
[root ~]# yum -y install nordugrid-arc-client
```

To submit a job, or perform any other action towards the ARC server you need a so-called *proxy-certificate* which is a Single Sign-On token for distributed grid-infrastructure. It is generated in the following way:

```
[user ~]$ arcproxy
Your identity: /DC=org/DC=nordugrid/DC=ARC/O=TestCA/CN=Test Cert 08272152
Proxy generation succeeded
Your proxy is valid until: 2018-08-28 09:54:24
```

You can start with the information query about your newly installed ARC computing element[1]:

[1] Examples uses `arc.example.org` as a domain name for A-REX host

```
[user ~]$ arcinfo -c https://arc.example.org/arex
Computing service:
  Information endpoint: https://arc.example.org:443/arex
  Information endpoint: https://arc.example.org:443/arex
  Information endpoint: https://arc.example.org:443/arex
  Submission endpoint: https://arc.example.org:443/arex (status: ok, interface: org.
↪ogf.glue.emies.activitycreation)
  Submission endpoint: https://arc.example.org:443/arex (status: ok, interface: org.
↪ogf.bes)
  Submission endpoint: https://arc.example.org:443/arex (status: ok, interface: org.
↪nordugrid.arcrest)
```

A simple job can be submitted with the `arctest` tool:

```
[user ~]$ arctest -J 2 -c https://arc.example.org/arex
Submitting test-job 2:
&( executable = "/usr/bin/env" )( stdout = "stdout" )( stderr = "stdout" )( gmlog =
↪"gmlog" )( jobname = "arctest2" )( clientxrsl = "&( executable = ""/usr/bin/env""␣
↪)( jobname = ""arctest2"" )( stdout = ""stdout"" )( join = ""yes"" )( gmlog = "
↪"gmlog"" )" )
Client version: nordugrid-arc-20180822231219
Test submitted with jobid: https://arc.example.org:443/arex/
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
```

The job status can be checked with the `arcstat` tool:

```
[user ~]$ arcstat https://arc.example.org:443/arex/
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
Job: https://arc.example.org:443/arex/
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
 Name: arctest2
 State: Running

Status of 1 jobs was queried, 1 jobs returned information
```

To fetch the job's stdout run `arccat` tool:

```
[user ~]$ arccat https://arc.example.org:443/arex/
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
GRIDMAP=/dev/null
HOSTNAME=arc.zero
TMPDIR=/tmp
GLOBUS_LOCATION=/usr
<output omitted>
```

Step 6. Play more with the ARC Computing Element

As an admin you might frequently need to extract information from the logs and directories that ARC computing element uses. The brief list of the relevant paths can be obtained from:

```
[root ~]# arcctl config brief
ARC Storage Areas:
    Control directory:
        /var/spool/arc/jobstatus
    Session directories:
        /var/spool/arc/sessiondir
```

(continues on next page)

(continued from previous page)

```
    Scratch directory on Worker Node:
        Not configured
    Additional user-defined RTE directories:
        Not configured
ARC Log Files:
    A-REX Service log:
        /var/log/arc/arex.log
    A-REX Jobs log:
        /var/log/arc/arex-jobs.log
    A-REX Helpers log:
        /var/log/arc/job.helper.errors
    A-REX WS Interface log:
        /var/log/arc/ws-interface.log
    Infosys Infoproviders log:
        /var/log/arc/infoprovider.log
```

To get information and manage jobs on A-REX server, the `arcctl job` is useful. Operations include but is not limited to:

- Listing jobs:

```
[root ~]# arcctl job list
NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
zb0LDm7RfEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmDBFKDme1CYXm
<output omitted>

[root ~]# arcctl job list --long
NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm      FINISHED arctest2        ␣
↪                 /DC=org/DC=nordugrid/DC=ARC/O=TestCA/CN=Test Cert 08272152
zb0LDm7RfEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmDBFKDme1CYXm      FINISHED arctest2        ␣
↪                 /DC=org/DC=nordugrid/DC=ARC/O=TestCA/CN=Test Cert 08272234
<output omitted>
```

- Job general information:

```
[root ~]# arcctl job info NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
Name          : arctest2
Owner         : /DC=org/DC=nordugrid/DC=ARC/O=TestCA/CN=Test Cert 08272152
State         : FINISHED
LRMS ID       : 16890
Modified      : 2018-08-27 22:06:34
```

- Job log:

```
[root ~]# arcctl job log NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
2018-08-27T22:03:34Z Job state change UNDEFINED -> ACCEPTED   Reason: (Re)Accepting␣
↪new job
2018-08-27T22:03:34Z Job state change ACCEPTED -> PREPARING   Reason: Starting job␣
↪processing
2018-08-27T22:03:34Z Job state change PREPARING -> SUBMIT   Reason: Pre-staging␣
↪finished, passing job to LRMS
2018-08-27T22:03:36Z Job state change SUBMIT -> INLRMS   Reason: Job is passed to LRMS
2018-08-27T22:06:34Z Job state change INLRMS -> FINISHING   Reason: Job finished␣
↪executing in LRMS
2018-08-27T22:06:34Z Job state change FINISHING -> FINISHED   Reason: Stage-out␣
↪finished.
```

- A-REX logs that mentions the job:

```
[root ~]# arcctl job log NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm --
↪service
### /var/log/arc/arex.log:
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: ACCEPTED: parsing␣
↪job description
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: ACCEPTED: moving to␣
↪PREPARING
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: PREPARING from␣
↪ACCEPTED
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: SUBMIT from PREPARING
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: state SUBMIT: starting␣
↪child: /usr/share/arc/submit-fork-job
[2018-08-27 22:03:36] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: state SUBMIT: child exited␣
↪with code 0
[2018-08-27 22:03:36] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: INLRMS from SUBMIT
[2018-08-27 22:06:34] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: Job finished
[2018-08-27 22:06:34] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: FINISHING from INLRMS
[2018-08-27 22:06:34] [Arc] [INFO] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: FINISHED from␣
↪FINISHING
[2018-08-27 22:06:34] [Arc] [ERROR] [16298/4]␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: Job monitoring stop␣
↪requested with 1 active references
### /var/log/arc/ws-interface.log:
[2018-08-27 22:05:32] [Arc.A-REX] [INFO] [16298/42] GET: id␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm path stdout
[2018-08-27 22:05:49] [Arc.A-REX] [INFO] [16298/43] GET: id␣
↪NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm path stdout
```

- Getting job attributes:

```
[root ~]# arcctl job attr NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm␣
↪jobname
arctest2
```

Now you are ready to *Install production ARC6 Computing Element*!

4.2.2 ARC Computing Element Installation and Configuration Guide

Prerequisites

Choosing the host

It is assumed that ARC CE is installed on top of an existing Linux computing cluster. Many Linux distributions are supported. ARC works well also on a complete virtual computing cluster environment in a cloud.

ARC is non-intrusive towards existing systems. We suggest to deploy ARC CE on a dedicated (virtual) machine connected to the cluster network and filesystem.

ARC software is very lightweight and does not require powerful machines to run, however if ARC CE will perform data transfers the requirements are higher. As a minimum, a production CE with 4 cores and 8GB of RAM should be capable of handling up to 10,000 concurrent jobs without problems. One CE can easily handle the load of a single cluster, however multiple CEs may be deployed in parallel for redundancy.

Plan for storage areas

Several storage areas are necessary for job submission, execution and data storing. You should mount/export following directories:

- *session directory*
- *data staging cache directory* (if planned)
- decide to what extent to use NOT cross-mounted *scratch directory* on the worker nodes

Session directory (and the cache directory if used) is typically cross-mounted NFS share. Please note, that in the typical setup when A-REX is running as root NFS share need to be exported with `no_root_squash`.

Local resource management system (LRMS)

Install and configure your LRMS (batch system). ARC supports a variety of LRMS back-ends:

- fork - fork jobs on the ARC CE host node, not a cluster. Targeted for testing and development but not for real production workloads.
- condor - uses HTCondor-powered HTC resource
- slurm - for SLURM clusters
- pbs - any flavor of PBS batch system, including Torque and PBSPro
- pbspro - dedicated Altair PBS Professional backend (from 6.1 release)
- ll - Load Leveler batch system
- lsf - Load Sharing Facility batch system
- sge - Oragle Grid Engine (formely Sun Grid Engine)
- boinc - works as a gateway to BOINC volunteer computing resources

Start by checking if you are able to submit jobs to the chosen LRMS from the ARC CE host.

You may consider setting up dedicated queues to use with ARC CE (e.g. per-VO queues).

Please also NOTICE that in some cases (*depending on LRMS*) you need to share the batch system log directories with ARC CE.

Configure OS accounts

Plan for local account(s) (or account pools) that will be used to execute jobs on the worker nodes.

These accounts should be also available on the ARC CE node.

Please note that ARC services are ran as root on the ARC CE node and switch to an appropriate local account when processing job data staging and job execution. This process is called *mapping*.

Installation

This section assumes you have already enabled the *NorduGrid repositories* for your package utility (yum/dnf/apt).

Note: If you are using RHEL-based operating systems, ARC can be directly installed from the EPEL repository.

Please note that in EPEL-7 `nordugrid-arc-*` packages delivers ARC 5. Use `nordugrid-arc6-*` to install ARC 6 from EPEL-7.

Install ARC CE core packages from repositories:

```
[root ~]# yum -y install nordugrid-arc-arex
or
[root ~]# apt-get install nordugrid-arc-arex
```

Any extra packages will be installed based on the ARC configuration file with *ARC Control Tool* as described *below*. Full list of packages to install manually (especially additional plugins) can be found *here*.

Grid security heavily relies on PKI and all actions requires certificates/keys for ARC CE as a service and users:

- for testing purposes, the ARC Test-CA and host certificate signed by the Test-CA are generated during A-REX installation.
- for production use please obtain a certificate signed by one of the IGTF accredited CAs and remove Test-CA files with `arcctl test-ca cleanup`.

In production ARC CE needs IGTF CA certificates deployed to authenticate users and other services, such as storage elements. To deploy IGTF CA certificates to ARC CE host, run[1]:

```
[root ~]# arcctl deploy igtf-ca classic
```

Warning: If you are on an EL9 type server (CentOS-Stream 9, AlmaLinux 9, Rocky 9, Fedora 9) you need to allow legacy crypto policies to be compatible with IGTF. On the command line of the ARC-CE server, issue:

```
update-crypto-policies --set LEGACY
```

Configuration

Configuration of ARC CE can be done by means of modifying the pre-shipped *zero configuration* available at `/etc/arc.conf`.

The purpose of this *zero configuration* is to offer a minimalistic working computing element out-of-the box right after package installation with zero additional configuration needed.

For production deployment you will need to customize the configuration in accordance to your actual setup and operations mode.

Note: ARC services must be restarted when changes have been made to arc.conf.

The ultimate information about available configuration options can be found in the *ARC Configuration Reference Document* which is also available locally as `/usr/share/doc/nordugrid-arc-*/arc.conf.reference`.

The most common configuration steps are explained below.

[1] Use `--installrepo` argument to enable repositories with IGTF CA certificates if ARC is not installed from the NorduGrid repos.

Configure authorization and mapping rules

Authorization rules define who can access the computing element (execute jobs, query info, etc). *Mapping rules* define which grid-users are mapped to which system accounts.

Both authorization and mapping rules in ARC6 rely on the concept of **authgroups**. Each authgroup represents a set of users, whose identities are matched to configured rules.

Once defined, authgroups can be applied to filter access to the CE per interface (*[arex/ws/jobs]*, *[gridftpd/jobs]*) and/or per-*queue*.

The `allowaccess` and/or `denyaccess` options in the corresponding block define which authgroups are allowed to access the interface or submit to the queue.

The *[mapping] block* used to configure the rules that defines how the particular authgroup members are mapped to OS accounts.

In the shipped *zero configuration* the `[authgroup:  zero]` is defined and applied to A-REX WS interface, the effect of which is to deny any access unless user is listed in the `testCA.allowed-subjects` file. The mapping is configured with `map_to_user` rule that assign the same `nobody` account to everyone in `zero` authgroup.

The typical configuration looks like this:

```
[authgroup: atlas]
voms = atlas * * *

[mapping]
map_to_pool = atlas /etc/grid-security/pool/atlas

[gridftpd/jobs]
allowaccess = atlas

[queue: qatlas]
allowacces = atlas
```

Please read the *Authorization, Mapping and Queue selection rules* document to get familiar with all aspects of this important configuration step.

Provide LRMS-specific information

One more critical configuration step is to supply ARC CE with relevant information regarding you LRMS specifics.

Specify you LRMS type

In the `arc.conf` there is a dedicated *[lrms]* block that defines the type of your LRMS, as well as several options related to the LRMS behaviour. For example, to instruct ARC to use SLURM, use the following configuration:

```
[lrms]
lrms = slurm
slurm_use_sacct = yes
```

Specify queues

In addition to specifying LRMS itself, it is necesssary to list all the queues that will be exposed via the ARC CE, by using *[queue: name]* blocks.

```
[queue: atlas]
comment = Queue for ATLAS jobs
```

More information about configuring particular LRMS to work with ARC can be found in *Batch systems support* document.

Configure A-REX Subsystems

The ARC Resource-coupled EXecution service (A-REX) is a core service handling execution and entire life cycle of compute jobs.

Enable job management interfaces

A-REX has several job management interfaces avaliable. One can control which of them are enabled and exposed by configuring the corresponding blocks

WS Interfaces (EMI-ES and ARC REST)
[arex/ws/jobs]

Gridftp
[gridftpd/jobs]

Internal
Install `nordugrid-arc-plugins-internal` package to use this interface.

Enable data services

ARC has a built-in data transfer framework called DTR. It was designed to be used in environments in which data transfer was not possible or not desirable on the worker nodes such as HPC centres or sites without local storage.

DTR relies on users submitting jobs with pre-defined input and output files. When A-REX receives a job, it takes care of downloading the specified input files to the job's session directory, then submits the job to the batch system. After the batch job finishes, A-REX takes care of uploading any output files to grid storage.

Define the *[arex/data-staging]* block to enable data-staging capabilities. Data transfers can be scaled out using *multi-host data-staging*.

DTR also includes a cacheing capability. If cacheing is enabled then A-REX will download all input files to the cache, and create symlinks from the session directory for each file. If a job requests a file that is already cached, A-REX will not download it again, but simply link from the existing cache file. Define the *[arex/cache]* block to enable cacheing.

More detailed technical documentation on ARC data features and advanced features such as ACIX and CandyPoind can be found in the *data overview pages*.

RunTime Environments

RunTime Environments can modify the job execution cycle and are used for advertising software or features offered by the computing facility.

ARC ships several RTEs that are ready to be used and classified as system-defined.

One can add ones own directories with so-called user-defined RTEs using the *runtimedir* configuration option in the `[arex]` block.

In ARC6, both system- and user-defined directories are local to the ARC CE node and *SHOULD NOT* be shared to worker nodes (unlike in ARC 5).

To use an installed RTE, one should additionally **enable** this RTE with *ARC Control Tool*. For example, to enable the system-defined `ENV/PROXY` RTE, run:

```
[root ~]# arcctl rte enable ENV/PROXY
```

More details on operating RunTime Environments can be found in *RunTime Environments in ARC6*.

Information system

ARC CE information system aims to collect and publish informaion to be used by special clients for matchmaking and/or monitoring the state and stats of the resource.

It is mandatory to configure the information system for production cases, like those of the WLCG computing infrastructure.

Defining general information

There are many information schemas and renderings of data available to comply to existing standards. There are several blocks that are used to define published information depending on schemas:

`[infosys]`
: The most common block that enables internal information collection from ARC CE host and LRMS.

`[infosys/cluster]`
: The common information about the whole cluster, including e.g. its capacity.

`[queue: name]`
: For heterogeneous clusters, most of the information in the `[infosys/cluster]` block can be re-defined on per-queue basis.

`[infosys/glue2]`
: Configures the GLUE2-specific values and enables internal glue2 rendering.

`[infosys/ldap]`
: Enables LDAP/BDII dedicated services to publish information via LDAP protocol.

`[infosys/glue2/ldap]`
: Enables GLUE2-schema LDAP rendering of the information.

`[infosys/nordugrid]`
: Enables LDAP rendering of the information according to the NorduGrid schema.

`[infosys/glue1]`
: Configures the GLUE1.x-schema specific values and enables LDAP rendering of GLUE1.x.

`[infosys/glue1/site-bdii]`
: Enables and configures GLUE1.x site-bdii functionality.

Accounting

ARC CE has built-in functionality to measure job's resource usage metrics that can be used for analyses and publishing to the SGAS and APEL centralized accounting services.

New in version 6.4: ARC 6.4 introduced the next generation accounting subsystem: A-REX store a complete job accounting data permanently in the local SQLite accounting database. Local accounting database is used as a powerful analyses instrument as a part of *ARC Control Tool* functionality and to generate standard-complient usage records to publish data to SGAS and APEL.

Deprecated since version 6.4: In 6.0-6.3 releases the *Job Usage Reporter of ARC* (JURA) tool creates standard-complient usage records from job usage information provided by the A-REX *Job Log* files, send the records to remote accounting services and optionally archive the records for future analyses and republishing.

If you need to configure accounting follow the *accounting guide*.

Configure Firewall

Different ARC CE services open a set of ports that should be allowed in the firewall configuration.

To generate iptables configuration based on `arc.conf`, run:

```
[root ~]# arcctl deploy iptables-config
```

Enable and Run Services

To enable and run all services as configured in `arc.conf`, run:

```
[root ~]# arcctl service enable --as-configured --now
```

Instead of using *ARC Control Tool* to manage *ARC services*, you can always use your OS native tools.

Test Basic Functionality

To test some basic job submission to the configured ARC CE, follow the instructions provided in the *Try ARC6: towards distributed computing in a few minutes*.

4.2.3 ARC5 to ARC6 Migration Guide

ARC 6 comes with major re-engineering in the code base leading to improvements in the deployment, configuration, maintenance and scalability of ARC CE. It introduces new features while some legacy interfaces, components and features have been obsoleted and removed.

Despite all the code changes the supported ARC 6 CE interfaces are unchanged as well as the structure of the control directory, session directory, accounting logs, etc. This makes it possible to upgrade ARC5 to ARC6 without a complete re-installation "from scratch".

Note: Despite the fact that we haven't observed any issues during normal package upgrades it is recommended to perform a clean install "from scratch" wherever possible in order to eliminate hidden bugs and ARC5 leftovers.

Following sections provides you with the necessary info about the major steps of an ARC5 to ARC6 migration.

Quick reference

Here is a quick **6 step** checklist for what to do:

1. Rewrite your `arc.conf`: you can *send your configuration to us* and we will help to rework it for ARC6.
2. Learn the ARC6 *packages* and *services* names.
3. Get familiar with *ARC Control Tool* that will simplify your life as an ARC6 administrator.
4. Don't forget to enable *RunTime Environments (RTE)* with `arcctl rte enable`.
5. Be aware of *accounting subsystem* changes.
6. Say "Good Bye!" to EGIIS, the ARC community is migrating over to a DNS-based ARCHERY system.

The details:

Configuration file

Because of the complete overhaul of the server-side ARC CE configuration, an existing ARC5 configuration can not be used with an ARC6 installation, it will immediately fail during the configuration validation process of the ARC service startup. Therefore you must create a new ARC6 configuration from scratch.

Note: There is no automatic converter from an old ARC5 configuration file to the new ARC6 syntax. The main reason for this is that there are many conceptual changes rather than simple option and block renaming changes.

We offer support in configuration rework via nordugrid-discuss mailing list or ARC6 Support Skype Channel.

Just send your configuration to us!

When reworking the configuration take notice of these **syntax changes**:

- no more quotes in option values
- blocks are now not only grouping options, but have a enable/disable functionality
- in block names keywords are separated by `/` to represent subblocks (if such exist), while block identifiers are separated by `:` from keywords

Note: It is advised to read the *General configuration structure* section of ARC configuration reference document before starting any rewrite.

There are also **changes in the configuration approach** in several areas. The major highlights are:

- The way *authorization and user mapping* is handled has been greatly reworked and improved. In particular mapping via *gridmap files* is NO longer required nor recommended. If you still have *gridmap files* in your configuration consider whether you really want to keep this approach in ARC6.
- Jura *accounting configuration* is now configured within dedicated blocks.
- Infosys schemas are now also enabled/disabled with blocks. In particular the default ARC5 NorduGrid LDAP schema should be explicitly enabled with *[infosys/nordugrid] block*.

Warning: ARC6 configuration also comes with consistent defaults for everything. If a block is enabled all configuration options in this block that has no specific value in `arc.conf` get their default values.

Configuration with all defaults included can be printed with `arcctl config dump`.

Notice that any changes in configuration requires restart off all affected ARC services.

Note: Config validation is enforced during A-REX start or can be run manually with `arcctl config validate`. So don't worry if you had misspelled something, the validator will catch it!

Operating ARC services

ARC6 also comes with the a rewised consistent *packages* and *services* naming scheme. We made the service and package names more clear and consistent with configuration block names in `arc.conf`.

Log file locations and internal format have been reviewed to provide consistency across the ARC subsystems. Log file locations can be printed with `arcctl config brief`.

ARC Control Tool

To simplify common ARC CE operations, ARC6 introduces *ARC Control Tool* - the single entry point to administrate and work with various ARC and third-party components. The `arcctl` tool helps ARC site admins by wrapping typical complex operations and unifying ARC CE administration under a single utility.

The `arcctl` tool was designed with BASH-completion in mind. It completes subsystem names, arguments, jobIDs, certificate DNs, RTE names, etc.

Note: To get familiar with some of the `arcctl` functionality it is advised to follow *Try ARC6: towards distributed computing in a few minutes* tutorial. Using the `arcctl` tool to administrate an ARC6 site is recommended even for the most experienced ARC5 admins.

RunTime Environments

The new framework for *RunTime Environments (RTE)* has gotten a redesigned architecture and a largely extended functionality (default, installed, enabled RTEs, introduction of RTE parameters).

The major conceptual and behaviour differences are:

- in ARC6 RTE directories are local to the ARC CE head node and *SHOULD NOT* be shared to worker nodes
- in ARC6 the RTE concept got extended by the introduction of RTE types such as default, installed, enabled, dummy, etc.. An ordinary ARC5 RTE would correspond to an installed user-defined ARC6 RTE. Please read the documentation.
- All installed ARC5 RTEs (available in a directory) should be enabled explicitly (e.g. `arcctl rte enable ENV/PROXY`) to be used on an ARC6 CE.

Note: You can use wildcards to enable all RTEs you have in one shot: `arcctl rte enable '*'`

Accounting

In ARC 6.0 *Jura accounting* had redesigned archive structure that introduces an archived records database and comes with a redesigned record republishing to accounting services, in addition to the already mentioned configuration changes.

To make ARC 6.0-6.3 accounting compatible with ARC5 `jura`, an automatic archive conversion is done by an additional `jura-archive-manager` process run by A-REX.

Warning: The JURA archive conversion consumes CPU and I/O resources of your ARC CE. If you are upgrading and have a lot of records in archive already expect an additional load during conversion.

In ARC 6.4 the *next generation accounting subsystem* had been introduces. Instead of publihsed records archiving it introduced local SQLite database that stores A-REX accounting records for each job for both on-site CE auditing and publishing.

Warning: If you want to keep old archived usage records from ARC 5 release it is advised to upgrade from ARC 5 to ARC 6.3 first, wait for records conversion completion, than update to ARC 6.4 or higher versions.

Retiring the EGIIS

All configuration blocks and services in ARC5 that correspond to registration to EGIIS are now gone in ARC6.

ARCHERY, the new DNS-based service endpoint catalogue for ARC is replacing EGIIS and it works *top-to-bottom* without explicit registration, i.e. nothing registration related needs to be configured in the `arc.conf` of an ARC6 CE.

If your CE had been previously registered in the country-level NorduGrid EGIIS your ARC CE registration is already migrated to the nordugrid.org ARCHERY instance.

If you find your CE missing (e.g. in the nordugrid top-level monitor) please contact the country-level ARCHERY instance admins as described in *this document*.

4.2.4 ARC CE Deployment Scenarios

To be written/WIP deployment guides:

WLCG Deployment with Data Capabilities

Note: WIP document!

Prerequisites

First follow the installation and configuration guide: *ARC 6 installation guide*.

Install necessary packages

CA certifcates

Install necessary CA certificates needed for WLCG (you have already installed the igtf-ca classic from the prerequisite step)

```
[root ~]# arcctl deploy igtf-ca mics slcs
```

Install and set up fetch-crl

To keep your CA's revocation lists up-to-date you need to install and set up `fetch-crl` tool. Required for a WLCG site.

Install the package:

```
[root ~]# yum install fetch-crl
```

Enable and start `fetch-crl`:

```
[root ~]# systemctl enable fetch-crl-boot
[root ~]# systemctl enable fetch-crl-cron --now
[root ~]# systemctl start fetch-crl-boot
[root ~]# systemctl start fetch-crl-cron
```

For older pre-systemd distributions (e.g. RHEL 6) use:

```
[root ~]# chkconfig fetch-crl-boot on
[root ~]# chkconfig fetch-crl-cron on
[root ~]# service fetch-crl-cron start
```

If you want to edit the times that `fetc-crl-cron` runs, have a look in the CRON configuration file: `/etc/cron.d/fetch-crl` The default settings should be good to get you going. For more advanced options, please see `fetch-crl` man pages.

Prepare for voms service signature authentication

If your arc.conf contains authgroups using voms like

```
[authgroup:atlas-jobs]
voms = atlas * * * *
```

Then you must install the corresponding voms directory in the following way:

```
arcctl deploy voms-lsc -e atlas
```

arcctl will search in the EGI database for the VO in question, and install the necessary files in the vomsdir, where the default is /etc/grid-security/vomsdir, and the vo folder will be /etc/grid-security/vomsdir/atlas in this example.

Configure ARC datastaging and cache

The datastaging service is enabled by adding the *[arex/data-staging]* block to `arc.conf`

A minimal configuration with logfile enabled could look like:

```
[arex/data-staging]
logfile=/var/log/arc/datastaging.log
```

See *[arex/data-staging]* for other options. Especially the `preferredpattern` and `deliveryservice` in case you have one or more remote delivery service machine(s) set up.

Example configuration:

```
[arex/cache]
cachedir=/grid/cache01
cachedir=/grid/cache02
```

(continues on next page)

(continued from previous page)

```
[arex/cache/cleaner]
cachesize=90 80
cachelifetime=50d
calculatesize=filesystem
```

Runtime environments

```
[root ~]# arcctl rte enable ENV/PROXY
```

ATLAS requires a dummy ATLAS-SITE RTE, create one and enable it like this:

```
[root ~]# arcctl rte enable --dummy APPS/HEP/ATLAS-SITE
```

Singularity from cvmfs on compute nodes

For singularity to work from cvmfs on the compute nodes (which it must) you need to run the following on all compute nodes

```
echo "user.max_user_namespaces = 15000" > /etc/sysctl.d/90-max_user_namespaces.conf;␣
↪sysctl -p /etc/sysctl.d/90-max_user_namespaces.conf
```

Configure the site to work with ARC Control Tower (aCT)

For the ARC data-staging mode to work, the site needs to receive jobs from aCT instead of receiving pilot jobs directly from the job-provider (e.g. PanDA).

WLCG Deployment for pilot sites

ARC Deployment beyond the WLCG scope

4.2.5 Tuning tips for ARC and FAQ

Note: Work-in-progress!

CA certificates

The `ENV/PROXY` RTE by default makes a copy of CA certificates in each job's session directory. Copying many small files to a shared file system is potentially expensive so it is preferred to have CA certificates locally installed on each worker node, and turn off the copying in `ENV/PROXY` with `arcctl rte params-set ENV/PROXY COPY_CACERT_DIR No`.

The CA certificate packages can be found in the repositories for ARC (http://www.nordugrid.org/documents/arc6/common/repos/repository.html)

You should also install the fetch-crl package and enable the services as you would on the ARC-CE.

File-system tuning

If you have your sessiondirectories and/or cache directories on NFS make sure you have increased the number of nfs kernel threads. The default is 8 which is too low. The RPCNFSDCOUNT variable should be increased to 32 or maybe even 64 or even larger depending on your system (number of worker-nodes, number of cores on the nfs servers).

How to extract backtrace of core dump file

How to extract backtrace of ARC core dump file using gdb and save the output to a file.

```
gdb  <path-to-arc-binary> <path-to-core-file>
(gdb) set logging file gdb.log
(gdb) set logging on
(gdb) thread apply all bt
```

Example: Assuming ARC is installed in default location: .. code-block:: console

> gdb /usr/sbin/arched /var/log/arc/arccore/core.30547 (gdb) set logging file gdb.log (gdb) set logging on (gdb) thread apply all bt

How to extract backtraces using a-rex-backtrace-collect

a-rex-backtrace-collect processes core file(s) collected in ARC_LOGS_DIR/arccore folder and produces their backtraces. The backtrace(s) are stored in files <core_name>.backtrace. The ARC instalation location can be adjusted using ARC_LOCATION environment variable. The location of configuration file can be specified using ARC_CONFIG environment variable.

.. code-block::console
> a-rex-backtrace-collect

4.2.6 Operating ARC CE Subsystems

ARC6 Packages

Table 4.1: List of ARC 6 binary packages

Package info	ARC 6 package name	Block name[Page 139]	ARC 5 package name
Base package holding common files	nordugrid-arc	N/A	nordugrid-arc
ARC Hosting Environment Daemon	nordugrid-arc-hed	N/A	nordugrid-arc-hed
ARC Resource-coupled EXecution service (A-REX)	nordugrid-arc-arex	[arex]	nordugrid-arc-arex
ARC Candypond Service	part of nordugrid-arc-arex	[arex/ws/candypond]	nordugrid-arc-candypond
ARC GridFTP Server	nordugrid-arc-gridft	[gridftpd]	nordugrid-arc-gridftpd
ARC LDAP-based Information Services[2]	nordugrid-arc-infosy	[infosys/ldap]	nordugrid-arc-aris, nordugrid-arc-ldap-infosys
ARC Data Delivery Service (DDS)	nordugrid-arc-datade	[datadeli	nordugrid-arc-datadelivery-service
ARC Cache Index (ACIX) - Core	nordugrid-arc-acix-c	N/A	nordugrid-arc-acix-core
ARC Cache Index (ACIX) - Scanner	nordugrid-arc-acix-s	[acix-sca	nordugrid-arc-acix-cache
ARC Cache Index (ACIX) - Index	nordugrid-arc-acix-i	[acix-ind	nordugrid-arc-acix-index
The nordugridmap tool	nordugrid-arc-nordug	[nordugri	nordugrid-arc-gridmap-utils
ARC development files	nordugrid-arc-devel	N/A	nordugrid-arc-devel
Python 2 bindings for ARC	python2-nordugrid-ar	N/A	python2-nordugrid-arc
Python 3 bindings for ARC	python3-nordugrid-ar	N/A	python3-nordugrid-arc
ARC command line clients	nordugrid-arc-client	own config	nordugrid-arc-client
ARC test tools	nordugrid-arc-test-u	N/A	nordugrid-arc-misc-utils
ARC LDAP monitor web application	nordugrid-arc-monito	N/A	nordugrid-arc-ldap-monitor
ARC base plugins (MCCs and DMCs)	nordugrid-arc-plugin	N/A	nordugrid-arc-plugins-needed
ARC Globus plugins	nordugrid-arc-plugin	N/A	nordugrid-arc-plugins-globus
ARC xrootd plugins	nordugrid-arc-plugin	N/A	nordugrid-arc-plugins-xrootd
ARC GFAL2 plugins[4]	nordugrid-arc-plugin	N/A	nordugrid-arc-plugins-gfal
ARC S3 plugins	nordugrid-arc-plugin	N/A	nordugrid-arc-plugins-s3
ARC Internal plugin	nordugrid-arc-plugin	N/A	N/A
ARCHERY administration tool	nordugrid-arc-archer	N/A	N/A
A-REX Python LRMS backends	nordugrid-arc-python	N/A	N/A
ARC optional worker nodes components [6.2]	nordugrid-arc-wn	N/A	N/A

List of packages deprecated in ARC6:

- `nordugrid-arc-ws-monitor`
- `nordugrid-arc-arcproxyalt`
- `nordugrid-arc-ca-utils`
- `nordugrid-arc-egiis`
- `nordugrid-arc-java`

[1] Block names are used by `arcctl service enable --as-configured` command to fetch necessary packages automanically based on the blocks configured in *arc.conf*

[2] Package define LDAP/BDII/Glue-Schema dependencies and contains wrappers to start all this LDAP world. Infoproviders are in the A-REX package.

[3] No longer relevant `saml_assertion_init` tool had been removed.

[4] Support for specific rotocols is provided by separate 3rd-party GFAL2 plugin packages.

ARC6 Services

Table 4.2: List of ARC 6 services

Block name[Page 140, 1]	ARC 6 service name	Main process	ARC 5 service name
[arex]	arc-arex	arched	a-rex
[arex/ws/candypond]	started by arc-arex	arched	arc-candypond
[gridftpd]	arc-gridftpd	gridftpd	gridftpd
[infosys/ldap]	arc-infosys-ldap	slapd, bdii-update	nordugrid-arc-aris
[datadelivery	arc-datadelivery-service	arched	arc-datadelivery-service
[acix-scanner	arc-acix-scanner	twistd	acix-cache
[acix-index]	arc-acix-index	twistd	acix-index

Authorization, Mapping and Queue selection rules

Overview

ARC CE authorization and mapping rules rely on the concept of **authgroups** (configured by *[authgroup]* blocks).

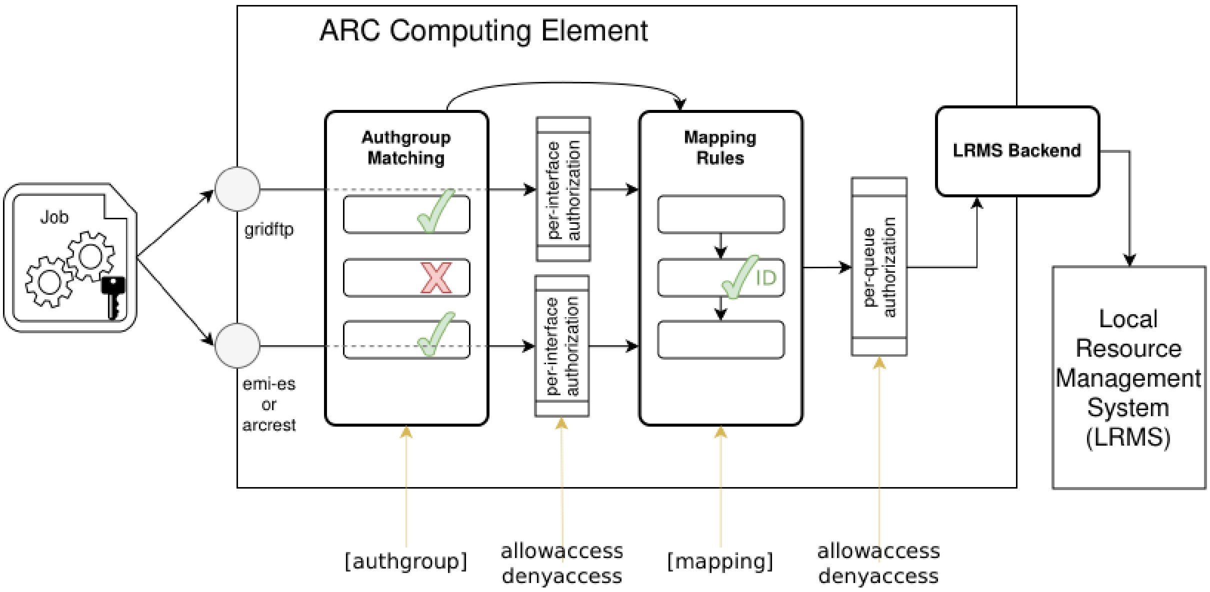


Fig. 4.1: General overview of authorization and mapping concepts in ARC6

During the connection establishment and authentication process on any of the ARC CE interfaces (e.g. job submission on the figure), the **Authgroup matching** happened. Each authgroup represents a set of users, whose identities are matched to configured rules. The same user can match (belongs to) several authgroups.

If configured by `allowaccess` and `denyaccess` options, **per-interface authorization** rules will be enforced. This rules defines the list of authgoups that granted or denied access to the particular ARC CE interface. Without per-interface authorization rules, configuration access to the interface is granted to anyone who passed authentication process.

[1] Block names are used by `arcctl service start --as-configured` command to start/stop necessary services automanically based on the blocks configured in *arc.conf*

Warning: User will be successfully authenticated only in case of certificate and proxy-certificate has been passed validation.

In case of time syncronization problems, missing or invalid CA certificates and CRLs, missing VOMS LSC files, etc - the connection *WILL NOT* be established.

Mapping rules in ARC6 also relies on *authgroup* membership. Configured *mapping rules* are processed sequentially and define the OS account **ID** that will be assigned to authgroup members. Several mapping methods are available, depending on requirements.

Notice that in ARC6 authorization rules can also be **enforced per-queue**. It uses the same `allowaccess` and `denyaccess` options syntax as for *per-interface* configuration but can *additionally restrict access* to the particular *queue*. By default access to the queue granted to all users authorized on interface level.

Defining authgroups

Authgroups should be defined on the top of `arc.conf` using the *[authgroup:groupname] block* before it will be referenced in the other parts of configuration.

Each authgroup is a named object, that will be referenced by its name during authorization and mapping rules configuration. There are no special restrictions to the authgroup names except the absence of spaces, so you can even define `*` authgroup to blow the mind of other `arc.conf` readers.

Each config line in the *[authgroup:groupname] block* represent a matching rule that are processed sequentially.

When the matchig criteria of the rule has been satisfied by user identity - the processing stops within this authgroup.

Whether user belogns to this authgroup or not is defined by the type of rule that was matched: there are **positively** and **negatively** matching rules. By default all rules are positive (user IS a member of authgroup when matched) unless prefixed with `-` sign.

Matching one of the authgroups does not mean that the same user is not processed for the next authgroup. All authgroups are evaluated, even if a user already has a match with one of the earlier processed authgroups.

Note: Notice that:

- authgroup blocks should be defined before referencing!
- authgroup rules within blocks are order-dependent!
- all authgroup blocks are evaluated!

Complete list of authgroup rules can be found in the *ARC Configuration Reference* document. Some examples are:

```
[authgroup: norduguest]
-file = /etc/grid-security/banned.dns
voms = nordugrid Guests * *

[authgroup: admin]
subject = /O=Grid/O=Big VO/CN=Main Admin

[authgroup: lcas]
plugin = 10 /usr/libexec/arc/arc-lcas %D %P liblcas.so /usr/lib64 /etc/lcas/lcas.db

[authgroup: any]
authgroup = norduguest
authgroup = admin
authgroup = lcas
```

(continues on next page)

(continued from previous page)

```
[authgroup: *]
all = yes
```

Applying authorization rules

You can enforce authentication restrictions on every interface. This includes:

- EMI-ES and REST interfaces configured with *[arex/ws/jobs] block*
- GridFTP job submission interface configured with *[gridftpd/jobs] block*
- GridFTP trivial storage interface configured with *[gridftpd/filedir] block*

In addition to interface level authorization, queue-level authorization can be configured using the same configuration approach,

The `allowaccess` confuguration option defines that the *specified authgroup* members are authorized to access the ARC-CE via this interface or access particular queue. A related config option `denyaccess` can in turns be used to reject access.

Multiple `allowaccess` and `denyaccess` authorization statements are allowed within a configuration block. These statements are processed *sequentially, in the order they are specified* in the config block.

The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface or queue is open to everybody authenticated.

Note: *Default deny* authorization approach can be configured using the empty authgroup.

Example: ARC6 zero configuration

In the shipped *zero configuration* the `[authgroup: zero]` is defined and applied to A-REX WS interface. The effect of this configuration is to allow access to CE only to the subjects stored in the `testCA.allowed-subjects` file. This file is empty by default and close down CE access until subjects are added by `arcctl test-ca usercert`.

```
[authgroup:zero]
file = /etc/grid-security/testCA.allowed-subjects

[arex/ws/jobs]
allowaccess = zero
```

Example: subject-based authorization

To authorize users based on certificate subject the *subject* or *file* rules can be used.

The *file* option support both:

- plain list of subjects (each line contains only a subject name),
- grid-mapfile format, when subject name followed by mapped account ID.

In both cases subject name should be enquoted if it contains spaces.

```
[authgroup: banana]
subject = /O=Grid/O=Bad Users/CN=The Worst

[authgroup: boss]
subject = /O=Grid/O=Big VO/CN=Main Boss

[authgroup: dnfromfile]
file = /etc/grid-security/local_users

[gridftpd/jobs]
denyaccess = banana
allowaccess = boss
allowaccess = dnfromfile
```

Example: VOMS-based authorization

To filter access based on VOMS certificate attributes, define one or more *[authgroup]* blocks using the *voms* keyword.

To verify VO membership signatures, ARC CE needs the so-called list of certificates (LSC) files that can be installed by *arcctl*.

Example configuration for *atlas*, *alice* and *swegrid.se* VOs[1]:

1. Deploy LSC files:

```
[root ~]# arcctl deploy voms-lsc atlas --egi-vo
[root ~]# arcctl deploy voms-lsc alice --egi-vo
[root ~]# arcctl deploy voms-lsc swegrid.se --voms vomss://voms.ndgf.
↪org:8443
```

2. Create authorization group and apply access resctiction to interface and/or queue in `arc.conf`:

```
[authgroup: atlas]
voms = atlas * * *

[authgroup: alice]
voms = atlas * * *

[authgroup: swegrid]
voms = swegrid.se * * *

[authgroup: all]
authgroup = atlas
authgroup = alice
authgroup = swegrid

[gridftpd/jobs]
allowaccess = all

[arex/ws/jobs]
allowaccess = all

[queue: qlhc]
allowaccess = alice
```

(continues on next page)

[1] In this example and in what follows, a simplified configuration is shown. An actual configuration will in most cases include different authgroups for different VO groups and roles.

(continued from previous page)

```
allowaccess = atlas

[queue: qswegrid]
allowaccess = swegrid
```

Configure mapping

Any grid user should be mapped to a local account to *start processes* and *access files*.

Mapping rules configured in *[mapping] block* define which grid-users (*specified by authgroup*) are mapped to which system accounts (several mapping methods available).

Rules in the *[mapping] block* are processed *in a sequence* in line order of the configuration file (from top to bottom).

There are two kind of rules avaiable:

- mapping rules (started with `map_`) that defines how the particular authgroup members are mapped,
- policy rules (started with `policy_`) that modifies the mapping rules sequence processing.

Default policy for mapping rules processing is:

- processing *CONTINUES* to the next rule if identity of user *DO NOT* match authgroup specified in the rule (can be redefined with *policy_on_nogroup* option)
- processing *STOPS* if identity of user matched the authgroup specified in the mapping rule. Depend on whether this mapping rule returns *valid UNIX identity* the processing can be redefined with *policy_on_map* and *policy_on_nomap* options.

Policy can be redefined at the any point of configuration sequence and affects all mapping rules defined *after* the polcy rule.

> **Warning:** If mapping process STOPS and there is still no local UNIX identity identified, the user running A-REX will be used (typically `root` unless redefined by *user* option for specific deployment case).
>
> When grid-identity is mapped to `root` account - request processing fails implicitely!

Example: mapping to the same account

The *map_to_user* option allows to map all authgroup members to the same account specified as an argument.

For example in shipped *zero configuration* all users that are matched to authgroup `zero` are mapped to the same `nobody` account (and `nobody` group) that will work with local job forking:

```
[mapping]
map_to_user = zero nobody:nobody
```

Example: mapping to the accounts pool

The most secure and flexible way is to map authgroup members to account pools (so-called *map_to_pool* method). It is recommended to use pools mapping when the resource is under the use of different communities.

In this approach, every member of specified *authgroup* will be dynamically mapped to one of the available accounts in the configured pool.

Available pool account names are stored one per line in the `pool` file inside the dedicated directory. Accounts from pool are assigned by means of leasing approach. All leased accounts are stored in the other files placed in the same directory. They can be reassigned to other users after 10 days of inactivity.

Example configuration for atlas:

1. Create necessary number of accounts to be used on ARC CE and Worked Nodes of the cluster.
2. Define ARC accounts pool:

```
[root ~]# mkdir -p /etc/grid-security/pool/atlas
[root ~]# for u in atlas{001..100}; do echo $u >> /etc/grid-security/pool/
↪atlas/pool; done
```

2. Configure mapping in `arc.conf`[2]:

```
[mapping]
map_to_pool = atlas /etc/grid-security/pool/atlas
```

Example: Legacy grid-mapfile based mapping

Warning: Legacy grid-mapfile based mapping is **NOT recommended** for the typical production loads.

In the grid-mapfile approach users are mapped to local accounts based on certificate DNs only. Mapping rules are stored line-by-line in the so-called grid-mapfile that describes which user is mapped to which account, for example:

```
"/O=Grid/O=NorduGrid/OU=uio.no/CN=Aleksandr Konstantinov" user1
"/O=Grid/O=NorduGrid/OU=hep.lu.se/CN=Oxana Smirnova" user2
```

In the simplest legacy case ARC can use the grid-mapfile for both authorization and mapping decisions.

Example configuration for legacy grid-mapfile case:

```
[authgroup: legacy]
file = /etc/grid-security/grid-mapfile

[mapping]
map_with_file = legacy /etc/grid-security/grid-mapfile
```

Grid-mapfiles in `arc.conf` can be also referred as a *[userlist]* objects and be generated regularly, keeping them up-to-date (from e.g. VOMS database) with `nordugridmap` utility that can be used and configured with the *[nordugridmap] block*

Note: You can find more information about moving from grid-mapfiles in the The life without gridmapfiles presentation.

Example: mapping with external LCMAPS rules

ARC can run an external plugin to map users that can be configured with the *map_with_plugin* option.

To support several production loads, ARC ships with the built-in LCMAPS plugin included in A-REX package:

```
[authgroup:all]
all = yes

[mapping]
map_with_plugin = all 30 /usr/libexec/arc/arc-lcmaps %D %P liblcmaps.so /usr/lib64 /
↪etc/lcmaps/lcmaps-arc-argus.db arc
```

[2] `atlas` is the name used in `[authgroup: atlas]`

LCMAPS itself is a third-party tool that should be installed and configured separately, which is beyound the scope of this guide.

Queue selection rules and queue configuration

Note: Behaviour new in ARC 6.11

Once a submitter is authorized to submit to one of the submission interfaces defined in arc.conf, the next step for ARC is to select a queue and check authorisation at queue-level.

The logic is as follows

1. if (queue defined in job description xrsl)

- submit to that queue w/o modification if authorised in the queue block

2. elseif (no queue defined in job description xrsl but a default queue is defined in arc.conf)

- substitute the default queue into xrsl if authorised in the queue block

3. elseif (no queue in defined in job description xrsl and no default queue is defined in arc.conf and the VO is authorised in one of the arc.conf queues*)

- substitute into xrsl the first of the queues where the VO is authorised according to arc.conf

4. else

- reject

* any VO is authorised in a queue that has no authorisation defined at all.

Note that this means that you must ensure that the authgroups you wish to allow submission from, are authorized in the queue you intend the authgroup to submit to. An example of an arc.conf setup that previously could result in the LRMS selecting the queue, but where submission today will be rejected could be the following:

```
[authgroup:exp1]
...

[authgroup:exp2]
...

[mapping]
..

[queue:queue1]
allowaccess= exp1

[queue:queue2]
allowacces= exp1
```

Let's look at the exp2 authgroup. According to this arc.conf we assume the authorisation and mapping is ok.

Old behaviour: If there is no queue information in the xrsl, the old behaviour would be to pass the job to the LRMS directly and let LRMS select the queue. This means that any queue in the system, even a queue not defined in arc.conf could be selected, e.g. a queue3 defined by the LRMS, but not in arc.conf.

New behaviour: From ARC version 6.11 only queue1 and queue2 will be possible queues for exp2 (and any authgroups), as they are the ones defined in arc.conf. Therefore in the example, none of the queues are allowed for exp2 and the job will be rejected.

A possible rewrite of this arc.conf example-snippet, to allow exp2 to submit to queue2 could be:

```
[queue:queue1]
allowaccess= exp1

[queue:queue2]
<empty>
```

This is an example of rule 3: No queue is defined in the job description xrsl and no default queue is defined in arc.conf and the VO is authorised in (at least) one of the arc.conf queues. In this example all VO's are authorised for using queue2.

Batch systems support

Overview

The A-REX has to interfaced to the LRMS in order to be able to submit jobs and query their information. The A-REX supports several Local Resource Management Systems (LRMS), with which it interacts by several **backend scripts**.

The A-REX assumes that the LRMS has one or more **queues**, which is a couple of (usually homogeneous) worker nodes grouped together. The different LRMSes have different concepts of queues (or have no queues at all).

Nevertheless, in the A-REX configuration, the machines of the LRMS should be mapped to A-REX queues. The client side job submission tools query the information system for possible places to submit the jobs, where each queue on a CE is represented as an **execution target**, and treated separately.

Configring A-REX to use one of these LRMS backends typically involves the following steps:

1. Sharing directories between A-REX, the LRMS frontend and its working nodes. It might involve setup of shared filesystems such as NFS or similar.
2. Configuring *[lrms] block* and *[queue]* blocks in `arc.conf` in respect to LRMS setup.
3. Configuring the A-REX in respect to the *shared scratch* directories configuration.

General LRMS configuration

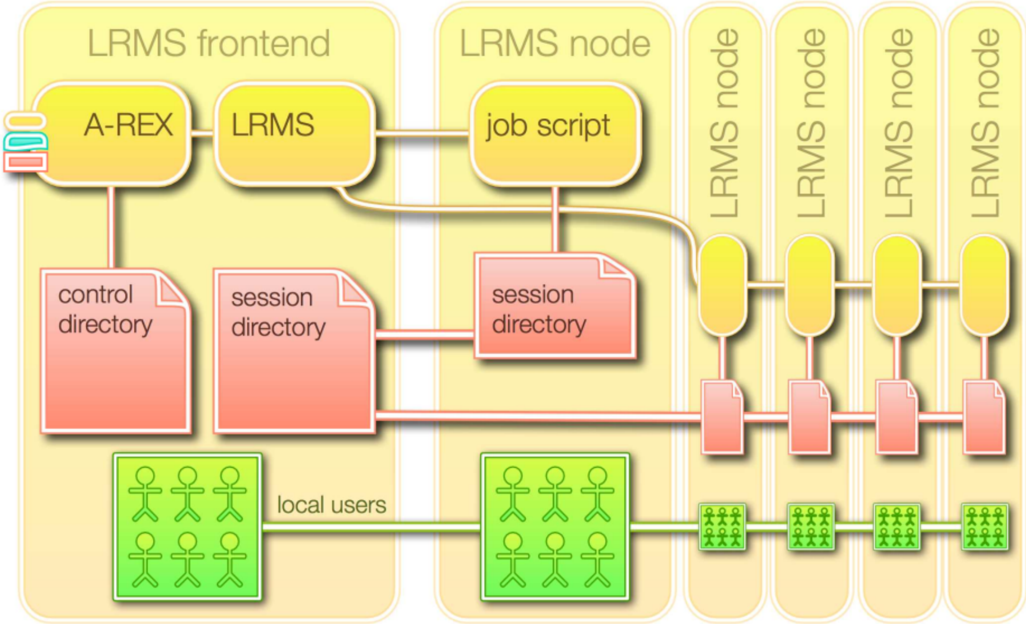
In the *[lrms] block* he name of the LRMS has to be specified with the *lrms* option.

The supported LRMS are:

- fork - fork jobs on the ARC CE host node, not a cluster. Targeted for testing and development but not for real production workloads.
- condor - uses HTCondor-powered HTC resource
- slurm - for SLURM clusters
- pbs - any flavor of PBS batch system, including Torque and PBSPro
- ll - Load Leveler batch system
- lsf - Load Sharing Facility batch system
- sge - Oragle Grid Engine (formely Sun Grid Engine)
- boinc - works as a gateway to BOINC volunteer computing rfesources
- slurmpy - new experimental SLURM backend written in Python (requires *nordugrid-arc-python-lrms* package instsalled).

Each LRMS has its own specific configuration options that are prefixed with LRMS name in *[lrms] block*.

Besided this specific options, the behaviour of LRMS backend is affected by storage areas and limits setup, in particular:



LRMS nodes

Fig. 4.2: The LRMS frontend and the nodes sharing the session directory and the local users

- *tmpdir* - defines the path to directory for temporary files on the worker nodes
- *shared_filesystem*, *scratchdir* and *shared_scratch* - changes the way how A-REX will store the jobs' data during the processing. More details can be found in *Job scratch area* document.
- *defaultmemory* and *req-queue defaultmemory* - set the memory limit values for jobs that has no explcit requirements in the job description.

Accounting considerations

A-REX has several approaches to collect accounting information:

- using cgroups measurements of memory and CPU on the WNs (starting from 6.2 release)
- using measurement from the GNU Time utility that wraps the job executable invokation inside the job script
- using data provided by LRMS

Depending on LRMS type in use there are different kind of information availble and/or missing in the LRMS accounting subsystem.

It is recommended to use cgroups or GNU Time methods to have reliable resources measurements in all cases. You can find more details in the *Measuring accounting metrics of the job* document.

Fork Backend

The Fork back-end is a simple back-end that interfaces to the local machine, i.e.: there is no batch system underneath. It simply forks the job, hence the name. The back-end then uses standard posix commands (e.g. `ps` or `kill`) to manage the job.

For is the deafult backend used in ARC pre-shipped zero configuration.

Recommended batch system configuration

Since fork is a simple back-end and does not use any batch system, there is no specific configuration needed for the underlying system.

It is still reqiuires queue definition and the queue should be named `fork`.

Example:

```
[lrms]
lrms = fork

[queue:fork]
```

Known limitations

Since Fork is not a batch system, many of the queue specific attributes or detailed job information is not available. The support for the *Fork batch system* was introduced so that quick deployments and testing of the middleware can be possible without dealing with deployment of a real batch system since fork is available on every UNIX box.

The *Fork back-end* is not recommended to be used in production. The back-end by its nature, has lots of limitations, for example it does not support parallel jobs.

Portable Batch System (PBS)

The Portable Batch System (PBS) is one of the most popular batch systems for small clusters. PBS comes in many flavours such as OpenPBS (unsupported), Terascale Open-Source Resource and QUEue Manager (TORQUE) and PBSPro (currently owned by Altair Engineering). ARC supports all the flavours and versions of PBS.

Recommended batch system configuration

PBS is a very powerful LRMS with dozens of configurable options. Server, queue and node attributes can be used to configure the cluster's behaviour. In order to correctly interface PBS to ARC (mainly the information provider scripts) there are a couple of configuration *REQUIREMENTS* asked to be implemented by the local system administrator:

1. The computing nodes MUST be declared as cluster nodes (job-exclusive), at the moment time-shared nodes are not supported by the ARC setup. If you intend to run more than one job on a single processor then you can use the virtual processor feature of PBS.
2. For each queue, one of the `max_user_run` or `max_running` attributes MUST be set and its value SHOULD BE IN AGREEMENT with the number of available resources (i.e. don't set the `max_running = 10` if there are only six (virtual) processors in the system). If both `max_running` and `max_user_run` are set then obviously `max_user_run` has to be less or equal to `max_running`.
3. For the time being, do NOT set server limits like `max_running`, please use queue-based limits instead.
4. Avoid using the `max_load` and the `ideal_load` directives. The Node Manager (MOM) configuration file (`<PBS home on the node>/mom_priv/config`) should not contain any `max_load` or `ideal_load` directives. PBS closes down a node (no jobs are allocated to it) when the load on the node reaches the `max_load`

value. The `max_load` value is meant for controlling time-shared nodes. In case of job-exclusive nodes there is no need for setting these directives, moreover incorrectly set values can close down a node.

5. Routing queues are now supported in a simple setup were a routing queue has a single queue behind it. This leverages MAUI work in most cases. Other setups (i.e. two or more execution queues behind a routing queue) cannot be used within ARC correctly.
6. PBS server logs SHOULD BE shared with ARC CE to allow backend scripts to check the job status and collect information needed for accounting. The path to logs on the ARC CE is defined with *pbs_log_path* option.

Additional useful configuration hints:

- If possible, please use queue-based attributes instead of server level ones.
- The `acl_user_enable = True` attribute may be used with the `acl_users = user1,user2` attribute to enable user access control for the queue.
- It is advisory to set the `max_queuable` attribute in order to avoid a painfully long dead queue.
- Node properties from the `<PBS home on the server>/server_priv/nodes` file together with the `resources_default.neednodes` can be used to assign a queue to a certain type of node.

Checking the PBS configuration:

- The node definition can be checked by `pbsnodes -a`. All the nodes MUST have `ntype=cluster`.
- The required queue attributes can be checked as `qstat -f -Q queuename`. There MUST be a `max_user_run` or a `max_running` attribute listed with a REASONABLE value.

Example:

```
[lrms]
lrms = pbs
defaultmemory = 512
pbs_log_path = /net/bs/var/log/torque/server_logs

[queue:grid_rt]
comment = Realtime queue for infrastructure testing
allowaccess = ops
advertisedvo = ops

[queue:alien]
comment = Dedicated queue for ALICE
allowaccess = alice
advertisedvo = alice
defaultmemory = 3500
```

Known limitations

Some of the limitations are already mentioned under the PBS deployment requirements. No support for routing queues, difficulty of treating overlapping queues, the complexity of node string specifications for parallel jobs are the main shortcomings.

SLURM

SLURM is an open-source (GPL) resource manager designed for Linux clusters of all sizes. It is designed to operate in a heterogeneous cluster with up to 65,536 nodes. SLURM is actively being developed, distributed and supported by Lawrence Livermore National Laboratory, Hewlett-Packard, Bull, Cluster Resources and SiCortex.

Recommended batch system configuration

The backend should work with a normal installation using only SLURM or SLURM+MOAB/MAUI. Do not keep nodes with different amount of memory in the same queue.

For production use-cases it is recommended to enable *slurm_use_sacct* option.

Example:

```
[lrms]
lrms=slurm
slurm_use_sacct=yes
defaultmemory=4096

[queue:normal]
comment=Queue for grid jobs
architecture=x86_64
totalcpus=1500
```

Using Python LRMS backend implementation

Experimental python LRMS backend can be used for SLURM after *nordugrid-arc-python-lrms* package installation. Python backed is distinguished by `slurmpy` name that should be specified in *lrms* option.

This backend respects the same options set, as a classical SLURM backend script, but additionally allows the connection over SSH when *[lrms/ssh] block* is enabled and configured.

Known limitations

If you have nodes with different amount of memory in the same queue, this will lead to miscalculations. If SLURM is stopped, jobs on the resource will get canceled, not stalled. The SLURM backend is only tested with SLURM 1.3, it should however work with 1.2 as well.

HTCondor

The HTCondor system, developed at the University of Wisconsin-Madison, was initially used to harness free cpu cycles of workstations. Over time it has evolved into a complex system with many grid-oriented features. Condor is available on a large variety of platforms.

Recommended batch system configuration

Install HTCondor on the A-REX node and configure it as a submit machine. Next, add the following to the node's Condor configuration (or define `CONDOR_IDS` as an environment variable):

```
CONDOR_IDS = 0.0
```

`CONDOR_IDS` has to be 0.0, so that Condor will be run as root and can then access the Grid job's session directories (needed to extract various information from the job log).

Make sure that no normal users are allowed to submit Condor jobs from this node. If normal user logins are not allowed on the A-REX machine, then nothing needs to be done. If for some reason users are allowed to log into the A-REX machine, simply don't allow them to execute the `condor_submit` program. This can be done by putting all local Unix users allocated to the grid in a single group, e.g. `griduser`, and then setting the file ownership and permissions on `condor_submit` like this:

```
[root ~]# chgrp griduser $condor_bin_path/condor_submit
[root ~]# chmod 750 $condor_bin_path/condor_submit
```

Example:

```
[lrms]
lrms = condor
defaultmemory = 2000

[queue:EL7]
comment = EL7 queue
defaultmemory = 3000
nodememory = 16384
condor_requirements = (Opsys == "linux") && (OpSysMajorVer == 66)
```

Known limitations

Only Vanilla universe is supported. MPI universe (for multi-CPU jobs) is not supported. Neither is Java universe (for running Java executables). ARC can only send jobs to Linux machines in the Condor pool, therefore excluding other unixes and Windows destinations.

LoadLeveler

LoadLeveler(LL), or Tivoli Workload Scheduler LoadLeveler in full, is a parallel job scheduling system developed by IBM.

Recommended batch system configuration

The back-end should work fine with a standard installation of LoadLeveler. For the back-end to report the correct memory usage and cputime spent, while running. LoadLeveler has to be set-up to show this data in the `llq` command. Normally this is turned off for performance reasons. It is up to the cluster administrator to decide whether or not to publish this information. The back-end will work whether or not this is turned on.

Known limitations

There is at the moment no support for parallel jobs on the LoadLeveler back-end.

LSF

Load Sharing Facility (or simply LSF) is a commercial computer software job scheduler sold by Platform Computing. It can be used to execute batch jobs on networked Unix and Windows systems on many different architectures.

Recommended batch system configuration

Set up one or more LSF queues dedicated for access by grid users. All nodes in these queues should have a resource type which corresponds to the one of the the frontend and which is reported to the outside. The resource type needs to be set properly in the `lsb.queues` configuration file.

Be aware that LSF distinguishes between 32 and 64 bit for Linux. For a homogeneous cluster, the `type==any` option is a convenient alternative.

In `lsb.queues` set one of the following:

```
RES_REQ = type==X86_64
RES_REQ = type==any
```

See the `-R` option of the `bsub` command man page for more explanation.

The *lsf_profile_path* option must be set to the filename of the LSF profile that the back-end should use.

Furthermore it is very important to specify the correct architecture for a given queue in `arc.conf`. Because the architecture flag is rarely set in the xRSL file the LSF back-end will automatically set the architecture to match the chosen queue.

LSF's standard behaviour is to assume the same architecture as the frontend. This will fail for instance if the frontend is a 32 bit machine and all the cluster resources are 64 bit. If this is not done the result will be jobs being rejected by LSF because LSF believes there are no useful resources available.

Known limitations

Parallel jobs have not been tested on the LSF back-end.

The back-end does not at present support reporting different number of free CPUs per user.

SGE

Sun Grid Engine (SGE, Oracle Grid Engine, Codine) is an open source batch system maintained by Sun (Oracle). It is supported on Linux, and Solaris in addition to numerous other systems.

Recommended batch system configuration

Set up one or more SGE queues for access by grid users. Queues can be shared by normal and grid users. In case it is desired to set up more than one ARC queue, make sure that the corresponding SGE queues have no shared nodes among them. Otherwise the counts of free and occupied CPUs might be wrong. Only SGE versions 6 and above are supported. You must also make sure that the ARC CE can run qacct, as this is used to supply accounting information.

Example:

```
[lrms]
lrms = sge
sge_root = /opt/n1ge6
sge_bin_path = /opt/n1ge6/bin/lx24-x86

[queue: long]
sge_jobopts= -P atlas -r yes
```

Known limitations

Multi-CPU support is not well tested. All users are shown with the same quotas in the information system, even if they are mapped to different local users. The requirement that one ARC queue maps to one SGE queue is too restrictive, as the SGE's notion of a queue differs widely from ARC's definition. The flexibility available in SGE for defining policies is difficult to accurately translate into NorduGrid's information schema. The closest equivalent of nordugrid-queue-maxqueuable is a per-cluster limit in SGE, and the value of nordugrid-queue-localqueued is not well defined if pending jobs can have multiple destination queues.

BOINC

BOINC is an open-source software platform for computing using volunteered resources. Support for BOINC in ARC is currently at the development level and to use it may require editing of the source code files to fit with each specific project.

Recommended batch system configuration

The BOINC database can be local to the ARC CE or remote. Read-access is required from the ARC CE to check for finished jobs and gather information on available resources. The ARC CE must be able to run commands in the project's `bin/` directory.

Project-specific variables can be set up in an *RTE* which must be used for each job. The following example shows the variables which must be defined to allow job submission to BOINC for the project "example" to work:

```
export PROJECT_ROOT="/home/boinc/project/example" # project directory
export BOINC_APP="example"                         # app name
export WU_TEMPLATE="templates/example_IN"          # input file template
export RESULT_TEMPLATE="templates/example_OUT"     # output file template
export RTE_LOCATION="$PROJECT_ROOT/Input/RTE.tar.gz" # RTEs, see below
```

The last variable is a tarball of runtime environments required by the job.

Known limitations

The BOINC back-end was designed around projects that use virtualisation. The prototype implementation in the current ARC version may not be generic enough to suit all BOINC projects.

When preparing a BOINC job, the ARC CE copies a tarball of the session directory to the BOINC project download area. Once the job is completed and the output uploaded to the BOINC peoject upload area, a modified assimilator daemon must be used to copy the result back to the session directory so that it can be retrieved by ARC clients or uploaded to Grid storage by the ARC CE.

ARC Information System

TBD

ARC CE Data Staging and Caching

RunTime Environments in ARC6

Understanding RunTime Environments

ARC Computing Element is a front-end to the various heterogeneous resource providers. To run jobs on the particular resource provider there are always a set of software or workflow-specific paths, tools, libraries, environmental variables or even dynamic content that should be recreated in the job content.

To provide a flexible way of job runtime environment tuning, ARC enforces the concept of the RunTime Environment (RTE).

ARC RunTime Environments (RTEs) provide two features:

Advertising
: indicate the available environment to be requested by end-users

Modifying job environment
: flexibly contextualize job execution environment

Advertising RTEs

Advertising RTEs provides user interfaces to application software and other resources in a way that is independent of the details of the local installation of the application and computing platform (OS, hardware, etc.).

It addresses setups typically required by large research groups or user bases, dealing with a common set of software. The actual implementation of a particular RTE may differ from site to site as necessary.

However, it should be designed so that resource providers with different accounting, licence or other site-specific implementation details can advertise the same application interface (RTE) for all users.

Despite possibly different parameters or implementation, the same software addressed by the same RTE name should be known by community. It is also supported to add RTE versioning at the end of the RTE name (after a dash). The RTE version will be used for resource matchmaking along with the RTE name.

For example to request `ENV/PROXY` and `APPS/HEP/ATLAS` with version `20.1.0.1` or greater in the *xRSL* job description:

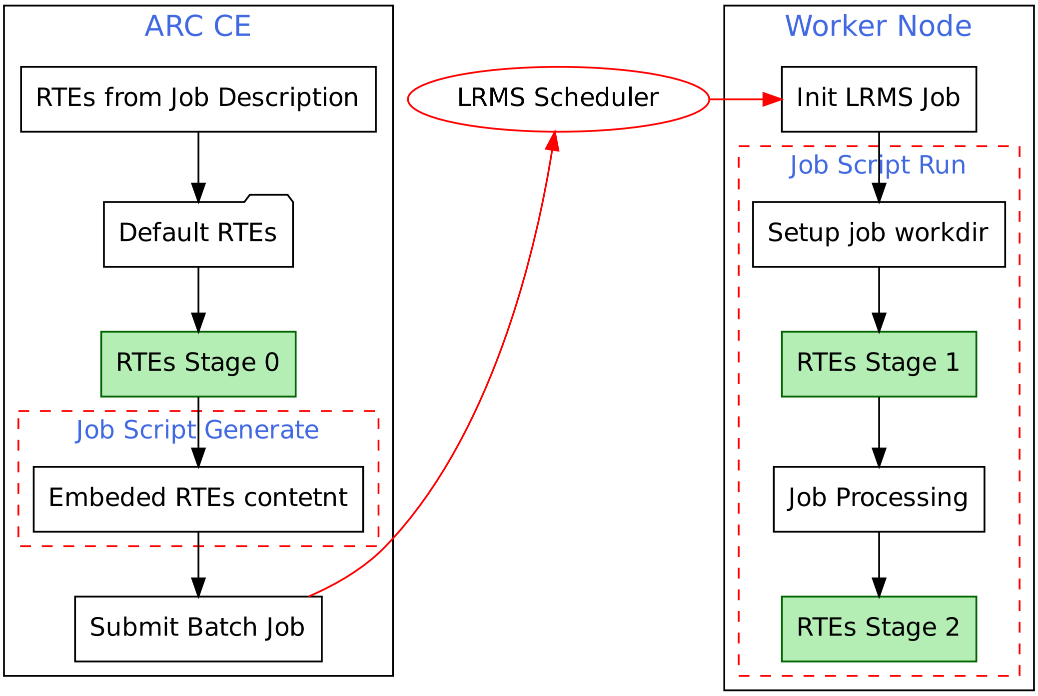
```
(runTimeEnvironment="ENV/PROXY")
(runTimeEnvironment>="APPS/HEP/ATLAS-20.1.0.1")
```

It is always up to the local system administrators to take a decision whether to *install* and *enable* a particular RTE or not.

Modifying job environment

The RTE content itself is a BASH script that is aimed to run any arbitrary code during the job life cycle.

The first argument of the RTE script indicates the so-called RTE stage. If the job description specifies additional arguments for corresponding RTE's those are appended starting at the second position.



There are 3 stages of an RTE execution:

Stage 0
: RTE script sourced before the creation of the job's LRMS submission script. In this case the scripts are run by A-REX on the frontend (ARC CE), before the job is sent to the LRMS. Some environment variables are defined in this case, and can be changed to influence the job's execution later. *TODO: list of grami attributes as a dedicated technical note*

Stage 1
: The Embedded RTE function runs before the main job processing on the Worker Node under the LRMS. Such stage can prepare the environment for some third-party software package. The current directory in this case is the one which would be used for execution of the job. The variable `$HOME` also points to this directory.

Stage 2
: The embedded RTE function runs after the main job processing on the Worker Node under the LRMS. The main purpose is to clean possible changes done by *Stage 1* (like removing temporary files).

You can use *this template* to start writing custom RTE script that fulfill your needs.

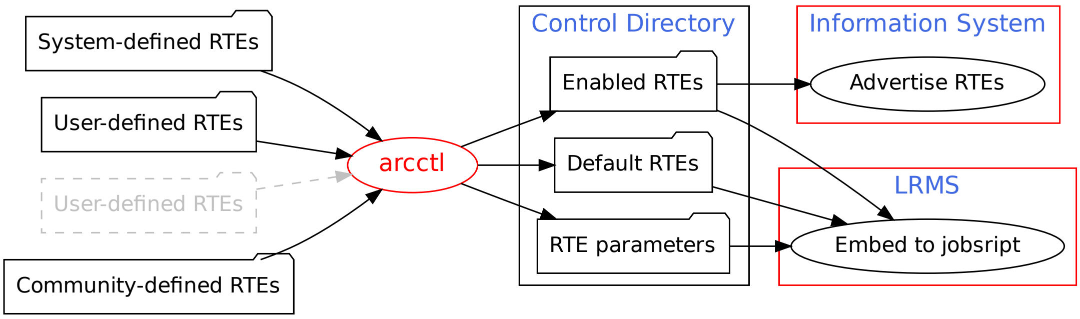
RunTime Environment script template

You can start with the following `template` to write custom RTE script:

```
#
# description: My RTE description ('# description:' is a keyword)
#
##########################
### RTE Parameters     ###
##########################
# NOTE! RTE parameters is advanced feature to keep the same code,
# but make it customizable for different clusters.
# Most probably you DO NOT need params for custom site-specific RTE.
# RTE parameters requires:
#   1. Parameters description headers ('# param:' is a keyword):
# param:PARAM_1:string:DEFAULT_VALUE:Description of parameter. Predefined 'string'␣
↪type means it can have arbitrary string value. Default is 'DEFAULT_VALUE'.
# param:PARAM_2:v1,v2,v3:v3:Description of parameter. This parameter can be set to 'v1
↪', 'v2' or 'v3' value only. Default is 'v3'.
#   These headers used by arcctl to operate parameters.
#   Should be defined within first 20 lines of RTE script.
#
#   2. Definiton of parameters default values for shell
PARAM_1="${PARAM_1:-DEFAULT_VALUE}"
PARAM_2="${PARAM_2:-v3}"

##########################
### RTE Code           ###
##########################
if [ "x$1" = "x0" ]; then
   # RTE Stage 0
   # You can do something on ARC host as this stage.
   # Here is the right place to modify joboption_* variables.
   # Note that this code runs under the mapped user account already!
   # No root priveledges!
elif [ "x$1" = "x1" ]; then
   # RTE Stage 1
   # Do something on WN before job execution.
elif [ "x$1" = "x2" ]; then
   # RTE Stage 2
   # Do something on WN after job execution.
fi
```

Operating RunTime Environments

From the release of ARC6, operating RunTime Environments is changed significantly and rely on *ARC Control Tool*:



Installing RTE scripts

There are set of *System-defined RTEs* pre-installed with the ARC CE packages that aim to fulfill common workflows.

An ARC CE administrator can add additional RTE directories (so-called *User-defined RTEs*). These additional places should be specified in `arc.conf` using the *runtimedir* configuration option in the `[arex]` block. Custom RTE scripts can be developed using *this template* as a starting point.

The *Community-defined RTEs* are additional RTEs created by research communities. These RTEs can be provioisioned to ARC CE from the trusted registries (including required software bundles) with *ARC Control Tool*.

Note: In ARC6 directories with RTE script are local to ARC CE and **SHOULD NOT** be shared with worker nodes

The RTE names used for *advertising* are implied by directory structure, e.g. in the `ENV/PROXY` the `ENV` is the directory inside the *System RTEs* location and `PROXY` is the name of the file.

Enabling RTEs

Installed RTEs should be enabled to be *advertised* and used during the job submission.

By name

To enable a particular RTE by name run the following command:

```
arcctl rte enable ENV/PROXY
```

By path

Especially if you have several RTEs with the same name installed, you can choose the exact one by specifying the filesystem path:

```
arcctl rte enable /usr/share/arc/rte/ENV/PROXY
```

Using wildcards

To enable several RTEs you can pass as many names as you want to the `arcctl` command. Additionally you can use glob (`man 7 glob`) wildcards in RTE names.

The following command will enable all `APPS/HEP/ATLAS` RTEs for `SLC6` builds:

```
arcctl rte enable APPS/HEP/ATLAS-*-X86_64-SLC6-*
```

Note: It is advised to use wildcards to enable all user-defined RTEs during ARC5 to ARC6 upgrade

Dummy RTEs

In case you need RTEs only for *advertising* but no need for script content, you can create a *Dummy RTE* for the specified name. The following command enables an `APPS/MYAPP` RTE with empty contents:

```
arcctl rte enable APPS/MYAPP --dummy
```

An example of dummy RTEs could be `APPS/ATLAS-SITE` used by the ATLAS experiment for sites to adversise that this indeed is an ATLAS-SITE.

Default RTEs

Default RTEs aim to address the workflows when *advertising* and implicit request in the job description is not needed, however *modification* of every submitted job (adjusting memory, setting LRMS scratch, etc) is required on the resource provider.

Installed RTEs can be selected for default inclusion to the job lifecycle with the following *ARC Control Tool* command:

```
arcctl rte default ENV/LRMS-SCRATCH
```

This will transparently add `ENV/LRMS-SCRATCH` to each job and will be executed the same way as *Enabled RTEs*. A default RTE does not need to be **enabled** to be executed, but it **can be enabled** if one wants to publish it in addition to executing it.

Note: You can use the same *by-name*, *by-path* and *wildcard* techniques as for *enabling*

RTE Parameters

To handle heterogeneity of resource providers, some RTEs can be parametrized.

For example, the system-defined `ENV/PROXY` RTE that transfers the delegated proxy-certificate to the worker node can optionally transfer CA certificate directories. This optional part is controlled by `COPY_CACERT_DIR` parameter.

To check if an RTE contains parameters and their default values, run the:

```
[root ~]# arcctl rte params-get ENV/PROXY
COPY_CACERT_DIR=Yes
```

You can also see the description and allowed values adding the `--long` keyword.

To set an RTE parameter value, the following command is used:

```
arcctl rte params-set ENV/PROXY COPY_CACERT_DIR No
```

List available RTEs and their status

To view the summary of all *installed*, *enabled* and *default* RTEs run:

```
[root ~]# arcctl rte list
<output omitted>
APPS/HEP/ATLAS-20.8.0-X86_64-SLC6-GCC48-OPT (user, enabled)
APPS/HEP/ATLAS-20.8.1-X86_64-SLC6-GCC48-OPT (user, enabled)
APPS/HEP/ATLAS-20.8.2-X86_64-SLC6-GCC49-OPT (user, enabled)
<output omitted>
ENV/LRMS-SCRATCH                 (system, default)
ENV/PROXY                        (system, masked, disabled)
ENV/PROXY                        (user, enabled)
ENV/RTE                          (system, disabled)
ENV/RUNTIME/ALIEN-2.17           (user, enabled)
VO-biomed-CVMFS                  (dummy, enabled)
```

The first tag describe the RTE origin (`system`, `user` or *dummy*). The following tags shows the status.

The special `masked` keyword indicates that the RTE name is used more that once and *by-name* operations will apply to another RTE script. For example `ENV/PROXY` will be enabled from the user-defined location as the system-defined is masked. However it is possible to enable a `masked` RTE *by specifying its path*.

Listing the particular kind of RTEs (e.g. *enabled*) is possible with the appropriate argument (see *ARC Control Tool* for all available options):

```
[root ~]# arcctl rte list --enabled
<output omitted>
APPS/HEP/ATLAS-20.8.2-X86_64-SLC6-GCC49-OPT
ENV/PROXY
ENV/RUNTIME/ALIEN-2.17
VO-biomed-CVMFS
```

The *long* listing allows to get the detailed pointers to RTEs locations and descriptions:

```
[root ~]# arcctl rte list --long
System pre-defined RTEs in /usr/share/arc/rte:
    ENV/PROXY                        # copy proxy certificate to the job session␣
↪directory
    ENV/RTE                          # copy RunTimeEnvironment scripts to the job␣
↪session directory
    ENV/LRMS-SCRATCH                 # enables the usage of local to WN scratch␣
↪directory defined by LRMS
User-defined RTEs in /etc/arc/rte:
    ENV/RUNTIME/ALIEN-2.17           # RTE Description is Not Available
    ENV/PROXY                        # RTE Description is Not Available
Enabled RTEs:
    ENV/RUNTIME/ALIEN-2.17           -> /etc/arc/rte/ENV/RUNTIME/ALIEN-2.17
    ENV/PROXY                        -> /etc/arc/rte/ENV/PROXY
Default RTEs:
    ENV/LRMS-SCRATCH                 -> /usr/share/arc/rte/ENV/LRMS-SCRATCH
```

View RTE content

Dumping the content of an RTE that will be embedded to job script is possible with the `cat` action:

```
[root ~]# arcctl rte cat ENV/LRMS-SCRATCH
SCRATCH_VAR="LOCALTMP"
# description: enables the usage of local to WN scratch directory defined by LRMS
# param:SCRATCH_VAR:string:WORKDIR:Variable name that holds the path to job-specific␣
↪WN scratch directory

SCRATCH_VAR="${SCRATCH_VAR:-WORKDIR}"

if [ "x$1" = "x0" ]; then
  RUNTIME_LOCAL_SCRATCH_DIR="\${${SCRATCH_VAR}}"
fi
```

Disable and Undefault RTEs

Enabled RTEs can be disabled running:

```
arcctl rte disable ENV/PROXY
```

The similar operation for *default RTEs* is called `undefault`:

```
arcctl rte undefault ENV/LRMS-SCRATCH
```

Note: You can use the same *by-name*, *by-path* and *wildcard* techniques as for *enabling*

System-defined RunTime Environments shipped with ARC

ENV/PROXY

Export delegated credentials (proxy certificate) to the job's session directory. Optionally copies CA certificates directory from ARC CE to session directory.

Sets the `X509_USER_PROXY`, `X509_USER_CERT` and `X509_CERT_DIR` to make files instantly available to client tools.

Parameters:

- `COPY_CACERT_DIR = Yes/No` - If set to Yes, CA certificate directory will be copied to the session directory along with proxy certificate. Default is `Yes`.
- `USE_DELEGATION_DB = Yes/No` - If set to Yes RTE will try to extract proxy certificate from A-REX delegation DB (works in limited number of cases). Default is `No`.

ENV/RTE

Copy RunTime Environment scripts to the job session directory for some workloads that require the files themselves instead of embedding the RTEs to the jobscript.

Designed to be used as *default* RTE.

Has no parameters.

ENV/LRMS-SCRATCH

Many resource providers use *scratchdir* to move files to the local worker node disk before running the job.

When the local scratch directory is created dynamically by LRMS (e.g. in the job prologue) and then cleaned up automatically after the job completion, the ENV/LRMS-SCRATCH is needed. The scratch place should be indicated by some environmental variable that holds a path to such LRMS-defined scratch directory.

This RTE is designed to be used as a *default* RTE to enable this optional functionality.

Parameters:

- `SCRATCH_VAR = name` - Variable name that holds the path to job-specific WN scratch directory. Default is `WORKDIR`.
- `TMPDIR_LOCATION = path` - Define the `TMPDIR` path on WN. Variable names can be used as a part of the path, e.g. `'$WORKDIR/tmp'`.

Note: The ENV/LRMS-SCRATCH is **not needed** if the scratch directory used and created on the worker node is of type `/<arc-conf-scratchdir-var>/<arc-job-id>`. It is only needed if the folder should be of type `<path defined by LRMS in SCRATCH_VAR>/<arc-job-id>`, i.e. a LRMS defined path is included.

ENV/CONDOR/DOCKER

ARC HTCondor backend supports submission to the Docker universe. This RTE enables this feature on-demand.

The RTE can be use by end-users when *enabled*. The RTE argument defines the Docker image name to be used, e.g:

```
(runtimeenvironment="ENV/DOCKER" "debian")
```

The RTE can also be used as *default* RTE to enforce Docker universe submission for any job. The Docker image should be set with the RTE parameter.

Parameters:

- `DOCKER_IMAGE = name` - Docker image to use for submitted jobs by default.

ENV/SINGULARITY

A general-purpose RTE that allows to run the submitted jobscript inside a defined singularity container image.

This RTE is designed to be used as both a *default* and *enabled* RTE.

When *enabled*, a user can specify the singularity image as an RTE argument:

```
(runtimeenvironment="ENV/SINGULARITY" "mysoftwareimage.sif")
```

Parameters:

- `SINGULARITY_IMAGE = images` - Define singulatiry images to de used with a job. It accepts comma separated `vo:path` pairs, where `vo` defines the virtual organization with image located at `path`. The special `default` value for `vo` defines the image used by default (if specific image for VO is not defined). The special `NULL` value for `path` skips singularity usage.
- `SINGULARITY_OPTIONS = options` - Define additional options for `singularity` executable. In particulat additional storage areas (e.g. CVMFS, CA certificates) can be specified to be mounted here.

ENV/CANDYPOND (experimental)

Makes available the `arccandypond` tool for use inside the job script on the Worker Nodes (including necessary environmental variables for it's operation).

Note: The CandyPond service itself should be enabled (defining the *[arex/ws/candypond] block*) on ARC CE as well.

Parameters:

- `CANDYPOND_URL = url` - Redefine the URL of CandyPond service (default is `auto` – ARC CE URL used for job submission will be used automatically)

Working with community-defined RTEs

New in version 6.5.

Community-defined RTEs are RTEs that created, managed and distributed by particular community.

ARC provides the software solution to automate Community-defined RTEs discovery and software environment provisioning for distributed computing e-Infrastructures.

Typicaly community-defined RTEs describes software packages used for computations. The cummunity itself is responsible for building, testing and verifying a particular version of software packages and the RTE scripts that will prepare such runtime enviroment to be used on the computing cluster worker nodes behind the computing element.

In general *RunTime Environments in ARC6* are very flexible, so in addition to defining software packages such RTEs can be used to transfer data or communicate with community services to fetch/register some data. It is up to community to define what they need.

This document describes how to work with community RTEs from ARC CE admin point of view.

To establish community-defined RTEs registry refer to *this guide*.

1. Enable Community-defined RTEs support

Community RTEs support is added as a technology preview in the ARC 6.5 and turned off by default.

If you need to support Community RTEs deployment on ARC CE, please install `nordugrid-arc-community-rtes` package:

```
[root ~]# yum -y install nordugrid-arc-community-rtes
```

2. Establish trust chain with community

The trust-chain between community and site-admin is based on the digital signatures. All Community-defined RTEs are supposed to be signed using OpenPGP standard for signatures. Technical implementation relies on the GNU Privacy Guard (GPG) software.

To add new community to the trusted list run:

```
[root ~]# arcctl rte community add example.org
The imported community public key data is:
  pub   2048R/AA56A775 2020-01-30 [expires: 2022-01-29]
        Key fingerprint = 3A47 F0D4 E406 D854 EDAA  ADB5 8FD6 DD57 AA56 A775
  uid                  Example Computations Lab <support@example.org>
  sub   2048R/3F914B9D 2020-01-30 [expires: 2022-01-29]

Is the community key fingerprint correct? (yes/NO): yes
```

Warning: Check the community key fingerprint matches the one provided to you by community authorities!

Note: Alternatively you can pass expected fingerprint value to `--fingerprint` option

In the provided example the `example.org` is the name of the community to add and in the same time it is interpreted as a domain name of ARCHERY community-defined RTEs registry.

If ARCHERY domain name is diffrent add `--archery <DOMAIN>` option.

It is also possible to establish trust with community using OpenPGP compatible keyserver or web-based RTEs registry[1] as an alternative to ARCHERY.

3. Discover RTEs in the registry

Note: Examples below show `APPS/EXAMPLE.ORG/SIMULATION-3.0.1` is already deployed on ARC CE. That is result of the next step execution.

You can list all available community-defined RTEs with `rte-list` command:

```
[root ~]# arcctl rte community rte-list example.org
APPS/EXAMPLE.ORG/SIMULATION-3.0.1    (deployed, registry)
APPS/EXAMPLE.ORG/ANALYSIS-1.7.0      (registry)
ENV/EXAMPLE.ORG/SENDSTATS-1.0.0      (registry)
```

RTEs optionally provide description string that can be viewer with long listing:

```
[root ~]# arcctl rte community rte-list example.org --long
Community deployed RTEs:
  APPS/EXAMPLE.ORG/SIMULATION-3.0.1    # Example Simulation Software
Additional community RTEs available in the registry:
  APPS/EXAMPLE.ORG/ANALYSIS-1.7.0      # Example Analysis Software
  ENV/EXAMPLE.ORG/SENDSTATS-1.0.0      # Send stats to central example.org services
```

Before deployment it can be usefull to look inside the RTE script. The `rte-cat` will show you the content:

[1] TODO: Document this alternative options. For now you can follow –help to find some info

```
[root ~]# arcctl rte community rte-cat example.org APPS/EXAMPLE.ORG/ANALYSIS-1.7.0
# description: Example Analysis Sowtware
# download: url:https://example.org/soft/analysis.sif␣
↪checksum:md5:63490ad38190a6f172a9020c0c5615f4

if [ "x$1" = "x1" ]; then
  mkdir ${RUNTIME_JOB_DIR}/bin
  cat > ${RUNTIME_JOB_DIR}/bin/example-analysis <<EOF
#!/bin/bash
exec singularity run ${RUNTIME_JOB_SWDIR}/analysis.sif "\$@"
EOF
  chmod +x ${RUNTIME_JOB_DIR}/bin/example-analysis
  export PATH=${RUNTIME_JOB_DIR}/bin:${PATH}
fi
```

4. Deploy community-defined RTE

Deploying community-defined RTE from the registry requires nothing more that passing RTE name to `rte-deploy`:

```
[root ~]# arcctl rte community rte-deploy example.org APPS/EXAMPLE.ORG/SIMULATION-3.0.
↪1
```

This command will:

- fetch RTE script signed by community
- verify signature using trusted community public keys (installed during step 2)
- deploy RTE script itself to be used further with `arcctl rte`
- download files specified in the community-defined RTE script to community software location
- verify checksum data of downloaded files

Note: HINT: It is usefull to increase debug level to at least `INFO` level during the deployment phase to monitor the progress.

5. Enable community-defined RTE

After deployemnt of community-defined RTEs, operating can be done *as ususal* - the same was as for other RTE types.

In particular you can list, *enable* or *default* RTEs including deployed from community registry:

```
[root ~]# arcctl rte list
ENV/CANDYPOND                     (system, disabled)
ENV/PROXY                         (system, enabled)
ENV/RTE                           (system, disabled)
ENV/SINGULARITY                   (system, disabled)
APPS/EXAMPLE.ORG/SIMULATION-3.0.1  (community, disabled)

[root ~]# arcctl rte enable APPS/EXAMPLE.ORG/SIMULATION-3.0.1
```

Additional information and hints

This section provide information how to customize the cummunity-defined RTEs operations.

Location of deployed community software

By default the location for deployed community software picked up automatically based on `arc.conf` and rely on the session directory in particular.

You can discover and change the location with `arcctl`:

```
[root ~]# arcctl rte community config-get example.org
SOFTWARE_DIR=/shared/session/_software/example.org
SOFTWARE_SHARED=Yes

[root ~]# arcctl rte community config-set example.org SOFTWARE_DIR /opt/community/
↪example.org
```

Operating without the registry

It is possible to deploy community-defined RTEs using the same `arcctl` automations without registry.

During the deployment phase you can provide URL to signed RTE file with the `--url` option:

```
[root ~]# arcctl rte community deploy example.org ENV/URLDEPLOYED-1.0.0 --url http://
↪example.org/rte.signed
```

Or it can be even RTE script wihtout signature if you trust the content:

```
[root ~]# arcctl rte community deploy example.org APPS/SIM-DEVEL --url file:///home/
↪example/dev/myrte.sh --insecure
```

Removing RTEs and communities

If you want to remove deployed RTE or entire community, there are `rte-remove` and `remove` actions respectively:

```
[root ~]# arcctl rte community rte-remove example.org APPS/EXAMPLE.ORG/SIMULATION-3.0.
↪1
[2020-02-06 18:11:03,653] [ARCCTL.RunTimeEnvironment.Community] [ERROR] [32505]␣
↪[Community
RTE APPS/EXAMPLE.ORG/SIMULATION-3.0.1 is enabled. Please disable it first or use "--
↪force"
to disable and undefault automatically]
[root ~]# arcctl rte community rte-remove example.org APPS/EXAMPLE.ORG/SIMULATION-3.0.
↪1 --force
[root ~]# arcctl rte community remove example.org
```

Measuring accounting metrics of the job

ARC CE has built-in capabilities to collect information about per-job resource consumption. This includes both ARC CE resources (e.g. data transfers, software environments) and worker nodes resources (e.g. CPU and memory usage). The full list of attributes stored in the A-REX Accounting Records (AAR) can be found in *this document*.

A-REX can use different methods (described below) to measure memory and CPU usage on the worker nodes, depending on their availability in the particular deployment case.

Measuring memory and CPU usage on the WN with cgroups

New in version 6.2.

When recent versions of GNU/Linux OS are used on the worker nodes the most precise and transparent way to measure all job workload is to rely on the `cgroups` kernel subsystem. Any `systemd`-based Linux distribution relies on `cgroups` heavily and they are already used.

Note: Some older operating systems may require mounting the cgroups tree explicitly. For example in RHEL6 it can be easily done with `libcgroup`:

```
[root ~]# yum install libcgroup
[root ~]# service cgconfig start
```

The benefit of using `cgroups` is that everyting will be accounted. Even if several payloads are executed (e.g. in pilot mode) or extra helper processes are spawned - the resource accounting will be accurate for the all workload done.

Enabling cgroups usage

To be able to use `cgroups` for accounting ARC needs an extra tool installed on the worker nodes – the `arc-job-cgroup`. Based on the tool availability, the job script will or will NOT use `cgroups` for measuring accounting metrics automatically.

The `arc-job-cgroup` tool is available for the majority of OSes as a packaged binary build as a part of ARC distribution (`nordugrid-arc-wn` package). So the easiest way to install it is to use your package manager on the worker nodes, e.g.:

```
[root ~]# yum install nordugrid-arc-wn
```

If it is not possible to install the packaged version for some reason, it is easy to compile the pure C source code with standard C library calls only.

```
[root ~]# wget https://source.coderefinery.org/nordugrid/arc/raw/master/src/wn/arc-
↪job-cgroup.c
[root ~]# cc -o arc-job-cgroup arc-job-cgroup.c
[root ~]# mv arc-job-cgroup /usr/local/bin/
[root ~]# chmod 4755 /usr/local/bin/arc-job-cgroup
```

How ARC operates cgroups

The idea behind LRMS-independent cgroup-based resource usage measurements in ARC is to:

- create child cgroups for memory and cpuacct controllers
- put the jobscript process into created cgroups (this will automatically catch all child processes)
- collect the accounting data at the end of the jobscript
- remove the child cgroup created at the beginning (moving all processes to parent cgroup)

If cgroups are used in the Kernel, the process already belongs to some cgroup. It can be either a root cgroup (used for all processes) or some dedicated cgroup created by LRMS with cgroups support, container management system, etc.

All resources used by the child cgroup are accounted in the parent cgroup. Moreover all parent-defined limits are inherited and enforced as well. So creating another child cgroup in hierarchy is safe from all points of view.

> **Warning:** Creating a child cgroup and put a task into it requires root privileges. This is the reason behind the SUID bit for arc-job-cgroup. However the code itself is as simple as the mkdir. You can review these 333 lines to reassure any possible fears.

If the arc-job-cgroup tool is not available, the cgroups tree is not mounted, or there are any other issues with cgroups creation, the job script code falls back to the GNU time measurement method.

Measuring memory and CPU usage on the WN with GNU time

The GNU time utility is capable of measuring and displaying information about the resources used by the executable it runs.

It is used as a part of the ARC-generated job script if found on the worker node.

Note: Changed in version 6.2.

In case of successful cgroups usage, GNU time will NOT be used by job script.

> **Warning:** The GNU time is a separate binary typically installed by dedicated package. Do not mix it up with built-in version of time in your shell (e.g. bash, zsh).
>
> Typically you can install it with e.g. yum install time or similar package management command.

For a non-standard location of GNU Time the *gnu_time* configuration option can be used to define it.

If the GNU time utility is not available the job will run as it is and only LRMS-provided metrics will be accounted.

Using LRMS-provided metrics

After the job has finished execution in the LRMS, the batch system backend scan-script extracts accounting information about the job from the LRMS, either executing command line clients, parsing logs or using API.

The exact data measurements and the method of these data collection completely depends on the LRMS backend implementation and differs from one backend to another.

Common metrics include LRMSStartTime and LRMSEndTime. There are also typically some memory and CPU usage metrics available.

Accounting Subsystem

New in version 6.4.

Changed in version 6.12.

> **Warning:** Information in this chapter is relevant only for 6.4+ ARC releases.
>
> Moreover ARC 6.12 get accounting changes to address the APEL move to ARGO messaging service protocol. If you are publishing to APEL you must update to 6.12+ ARC release.

Note: If you are lookng for the information about legacy accounting subsystem in 6.0-6.3 ARC releases please read *Accounting with legacy JURA* but it is highly recommended to update to recent release.

The next generation ARC Accounting Subsystem aimed to improve scalability, eliminate bottlenecks of the *legacy implementation* and provide a local on-site ARC CE accounting database to be queried and analyzed by CE admins.

Overview

Figure shows the overview of next generation accouting subsystem in ARC. More details can be found in *ARC Accounting Technical Details*.

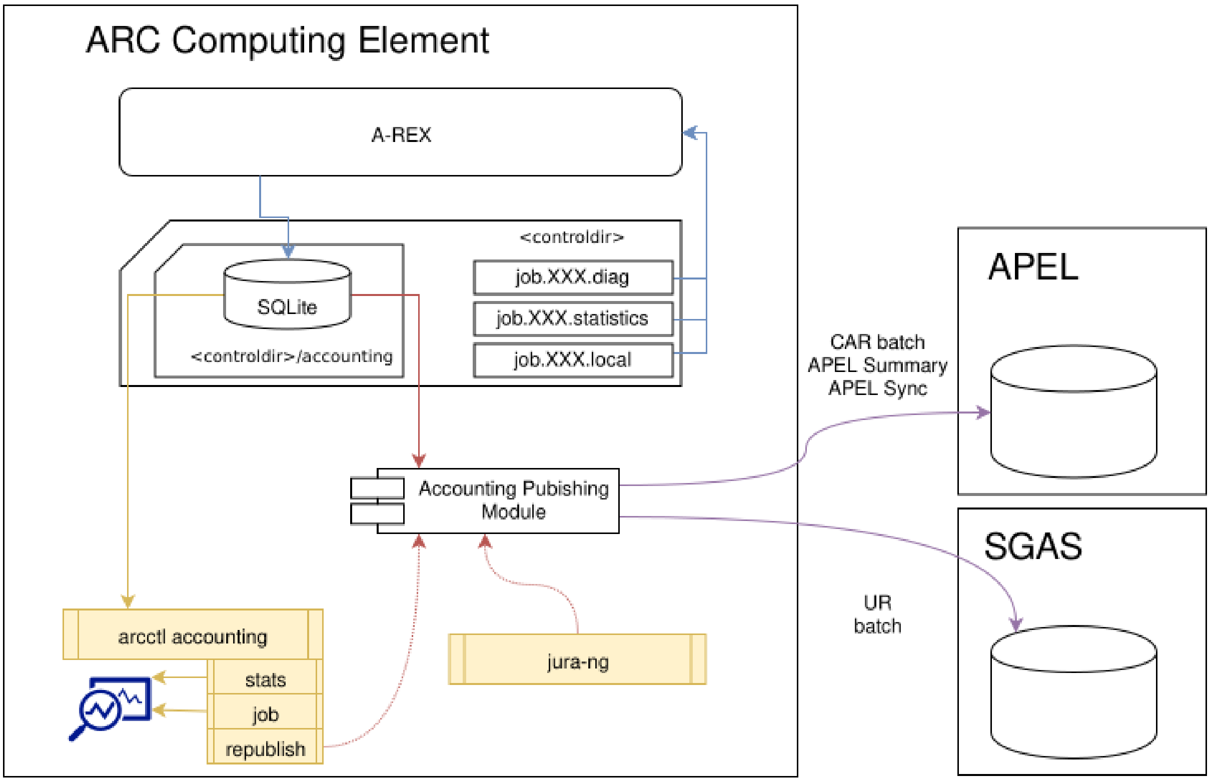


Fig. 4.3: Next generation ARC accounting subsystem overview: creating AAR records, getting access to the data and publishing to external SGAS/APEL services

The central point of ARC next generation Accounting Subsystem is a local SQLite accounting database that stores all the *A-Rex Accounting Record (AAR)* information. AAR defines all accounting information stored about a single ARC CE job.

The database is populated directly by A-REX based on per-job files inside the *control directory*. In particular the `.diag` file is a main source of *resource usage* information, `.statistics` holds data transfers measurements and

`.local` is a general datastore about the job properties like ID, owner, etc.

The record about the job is created in the accounting database as soon as the job enters ARC CE. Than each state change of the job is recorded. Once the job reached terminal state (both successfull completion or failed) all resource usage data is written becomes available via local queries (`arcctl accounting`) or for reporting to SGAS/APEL (`jura-ng`).

Accounting subsystem is integral part of the A-REX and allways enabled.

Configuration

In the typical use-case accounting subsystem just works and does not requires additional configuration.

However in some rare cases there are several things you can consider to tune.

WLCG VOs

The recommended and typically used way to authorize WLCG VOs is to use *[authgroup]* block *voms* configuration option that exactly match attributes in VOMS extesion of proxy certificate.

If you have authorization rules configured this way NOTHING should be configured.

In case you do not have at least one *voms* option in the any of the defined *[authgroup]* blocks A-REX will not trigget the VOMS attributes parsing, that consequently leads to no VO info in the accounting!

To have VO info in the accounting in this particular case you can add standalone authroup with *voms* option that have no further usage in `arc.conf`:

```
[authgroup: parsevoms]
voms = * * * *
```

There is also *forcedefaultvoms* configuration option (can be defined on per-queue basis) that can define the accounted VO for jobs that have no VOMS extension in owner's certificate.

Enabling accounting records reporing

It is possible to send resource usage reports to the centralized SGAS and/or APEL accounting databases.

Based on the local accounting database data the *Accounting Publishing Module* is capable of generating the:

- OGF Usage Record 1.0 (UR) XML format to be sent to the SGAS LUTS (Logging and Usage Tracking Service)
- EMI Compute Accounting Record (CAR) v1.2 XML format for individual job records to be sent to APEL
- APEL Summaries (to reduce traffic) and APEL Sync messages for full-featured integration with APEL services

Regular publishing sequence is handled by `jura-ng` helper tool and can be enabled with *[arex/jura] block*.

A-REX periodically runs `jura-ng` with default and minimum *hourly period*. It can be increased with *urdelivery_frequency* option. Futhremore it can be increased per-target, using the same option inside a particulat targer block.

JURA has dedicated log file defined by *logfile* option. Log rotation has been set for default `/var/log/arc/jura.log` location.

Accounting services for sending the records are configured with dedicated sub-blocks. You need to define a separate block with an unique targetname for every target server used.

Note: The target block name will be used by `jura-ng` to track that latest records sent to this targed. Be aware that if you rename the block, target will be handled as a new one. However `targeturl` change will not trigger a new target handling and records will continue publishing using the latest recorded timestamp

The AARs data will be reported to all of the defined destinations, unless `vofiler` option configured for some of them to filter records by VO name.

Warning: There were sevaral issues in the codebase and misunderstanging from the operational point of view how the `benchmarks` are propagated to the central accounting services with ARC accounting. Please read *About benchmarks and accounting publishing* for more details.

Configuring reporting to SGAS

The *SGAS sub-block* enables and configures an SGAS accounting server as a target destination to which ARC CE will send properly formatted OGF.98 Usage Record 1.0 (UR) XML records.

The *targeturl* option is the only mandatory parameter to configure SGAS target. In the specific setup cases you can also apply *VO filtering* and *set prefix* for local job IDs.

Example:

```
[arex/jura/sgas: NeIC]
targeturl = https://grid.uio.no:8001/logger
urbatchsize = 80
```

Configuring reporting to APEL

The *APEL sub-block* enables and configures an APEL accounting server as a target destination to which ARC will send the data.

The *targeturl* option defines the APEL broker URL to send records to. The currently known APEL AMS endpoint is provided in the *targeturl example* but you should refer to APEL for up to date information.

The *apel_messages* option allows you to choose between per-job EMI Compute Accounting Record (CAR) XML records publishing and APEL Summaries publishing. Sending summaries is a default and recommended by APEL behaviour that allows to save resources and traffic.

APEL Sync records that syncronize the total job counters per-month are always sent.

You also need *GOCDB name* of the resource. Since move to AMS publishing the *APEL topic* is always `gLite-APEL` that is a default value.

For correct production accounting setup it is recommended to specify resource *benchmarking results* in the *[queue:name] block*. ARC assumes that the nodes in the same queue are homogeneous with respect to the benchmark performance and benchmark values are specified per-queue.

Example:

```
[arex/jura/apel: EGI]
targeturl = https://msg.argo.grnet.gr
gocdb_name = RAL-LCG2

[queue: grid]
benchmark = HEPSPEC 8.73
```

Lookup local accounting data

Data in ARC CE accounting database can be viwed with *ARC Control Tool*. Timeframe of interest and many other filters can be specified, e.g.:

```
[root ~]# arcctl accounting stats --filter-vo ops --start-from 2019-07-10
A-REX Accounting Statistics:
  Number of Jobs: 1317
  Execution timeframe: 2019-07-10 00:01:45 - 2019-07-26 12:48:13
  Total WallTime: 4:40:28
  Total CPUTime: 0:22:32 (including 0:00:00 of kernel time)
  Data staged in: 2.3M
  Data staged out: 9.7K

[root ~]# arcctl accounting job events␣
↪UufLDmmmS5unf5481mks8bjnABFKDmABFKDmmSMKDmNBFKDm9nRnVo
2019-07-10 17:30:00   ACCEPTED
2019-07-10 17:30:00   PREPARING
2019-07-10 17:30:00   DTRDOWNLOADSTART
2019-07-10 17:30:08   SUBMIT
2019-07-10 17:30:08   DTRDOWNLOADEND
2019-07-10 17:30:11   INLRMS
2019-07-10 17:30:16   LRMSSTART
2019-07-10 17:30:26   LRMSEND
2019-07-10 17:30:47   FINISHING
2019-07-10 17:30:47   FINISHED
2019-07-14 15:01:16   DELETED

[root ~]# arcctl accounting job transfers␣
↪UufLDmmmS5unf5481mks8bjnABFKDmABFKDmmSMKDmNBFKDm9nRnVo
Data transfers (downloads) performed during A-REX stage-in:
  http://download.nordugrid.org:80/packages/nordugrid-arc/releases/6.1.0/src/
↪nordugrid-arc-6.1.0.tar.gz:
    Size: 5.2M
    Download timeframe: 2019-07-10 17:30:00 - 2019-07-10 17:30:04
No stage-out data transfers (uploads) performed by A-REX.
```

More queries examples can be found in *this document*.

Republishing records

When something goes wrong with accounting services, network, etc there is possible need of republishing local records again.

In the current implementation of accounting subsystem, there is no difference between publishing and re-publishing. The same *Accounting Publishing Module* will be used to generate the records to be sent to target service for defined timeframe.

Rebuplishing is triggered by ARC CE administrator using *ARC Control Tool*.

The most streamlined way is to republish data to the target that is already configured in `arc.conf` for regular publishing:

```
[root ~]#  arcctl accounting republish -b 2019-06-01 -e 2019-07-01 -t EGI
```

However it is also possible to define all target options from the command line, without the defined target in `arc.conf`:

```
[root ~]#  arcctl accounting republish --end-from 2019-06-01 --end-till 2019-07-01 \
> --apel-url https://msg.argo.grnet.gr --gocdb-name "UA-KNU" \
> --apel-messages summaries --apel-topic gLite-APEL
```

Clean up of the <controldir>/logs folder

With the accounting subsystem change in ARC 6.4 legacy archive files are no longer used or written to the <controldir>/logs directory. It is recommended to manually wipe this directory.

Job scratch area

ARC allows to configure different approaches to manage the job scratch area during the job life cycle. In most cases this is achieved by generating a wrapper submission script to the batch system that carries on tasks relative to directory creation and data movement.

The key elements are:

job session directory
directory on the ARC CE where the job files are located.

job scratch directory
directory on LRMS-managed worker nodes (WNs) where all I/O during computation is performed.

<arc-job-id>

a unique identifier for the job, assigned by ARC.

Example: `10YLDmDRgrynVALY5mGJwcyoABFKDmABFKDmUvKKDmABFKDmG588Rm`

job files

- input files, those coming from the client and data-staging framework that are needed for the actual job processing, copied inside a folder named after *<arc-job-id>*, that will also contain the job *stdout* and *stderr* files.
- some metadata files used by ARC, mainly *<arc-job-id>.comment* and *<arc-job-id>.diag*

Job session directory is configured with *sessiondir* configuration option. It is possible to configure several session directory root paths. A-REX will then select one of the available directories and append the *<arc-job-id>* to the path.

Example:

```
/nfs/sessiondir5/
├── 10YLDmDRgrynVALY5mGJwcyoABFKDmABFKDmUvKKDmABFKDmG588Rm
│   ├── script.sh
│   ├── outfile
│   ├── stderr
│   └── stdout
├── 10YLDmDRgrynVALY5mGJwcyoABFKDmABFKDmUvKKDmABFKDmG588Rm.comment
├── 10YLDmDRgrynVALY5mGJwcyoABFKDmABFKDmUvKKDmABFKDmG588Rm.diag
```

There are several configuration options that affect the selection of *job scratch directory*:

shared_filesystem
defines if the *job session directory* is shared between ARC CE and WNs (by means of e.g. NFS). Sets the environment variable `RUNTIME_NODE_SEES_FRONTEND`.

scratchdir
defines the path to *job scratch directory* on the WN. Sets the environment variable `RUNTIME_LOCAL_SCRATCH_DIR`.

movetool
defines what tool the job wrapper will use to move data from the session directory to the local WNs scratch directory. Sets the environment variable `RUNTIME_LOCAL_SCRATCH_MOVE_TOOL`.

shared_scratch
defines that the WNs scratch directory can be accessible from ARC CE (by means of e.g. NFS) using the configured path. Sets the environment variable `RUNTIME_FRONTEND_SEES_NODE`.

Currently this legacy option is not used much, as it is uncommon that nodes share a single scratch directory. If you require such functionality please contact the ARC team.

Note: Described environment variables can be redefined by *RunTime Environments* dynamically. For example *ENV/LRMS-SCRATCH* can be used to utilize local scratch that created dynamically by LRMS.

Compute inside shared session directory

In the simplest case, where the *job session directory* is **shared** between ARC CE and WNs (e.g. via NFS) and accessible on the same path - the *job scratch directory* is the *job session directory*.

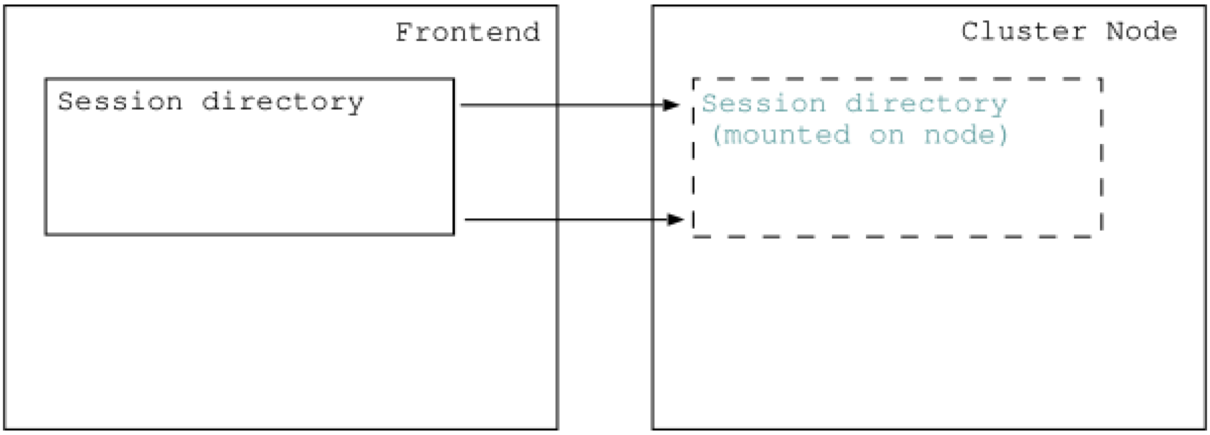


Fig. 4.4: Sessiondir is shared between ARC CE and WNs. No local scratchdir defined.

Configuration:

```
[arex]
shared_filesystem = yes
```

Variables set in the wrapper script:

```
RUNTIME_NODE_SEES_FRONTEND='yes'
```

Compute inside a local WN scratch directory

Session directory is shared, WN scratch directory is not shared

For I/O performance reasons it is possible to perform computations inside a local directory on the WN.

In this case *job scratch directory* is created in the configured local directory and files from the *job session directory* are **moved** to job scratch before execution starts (a).

Only files representing the job's *stdout* and *stderr* are placed in the original *job session directory* and soft-linked in scratch (b).

After the execution has completed all output files are moved to the *job session directory* (b) and are then available for clients to download (c).

The move is performed by default using the *mv* command. The admin can change this behaviour using the *movetool* option.

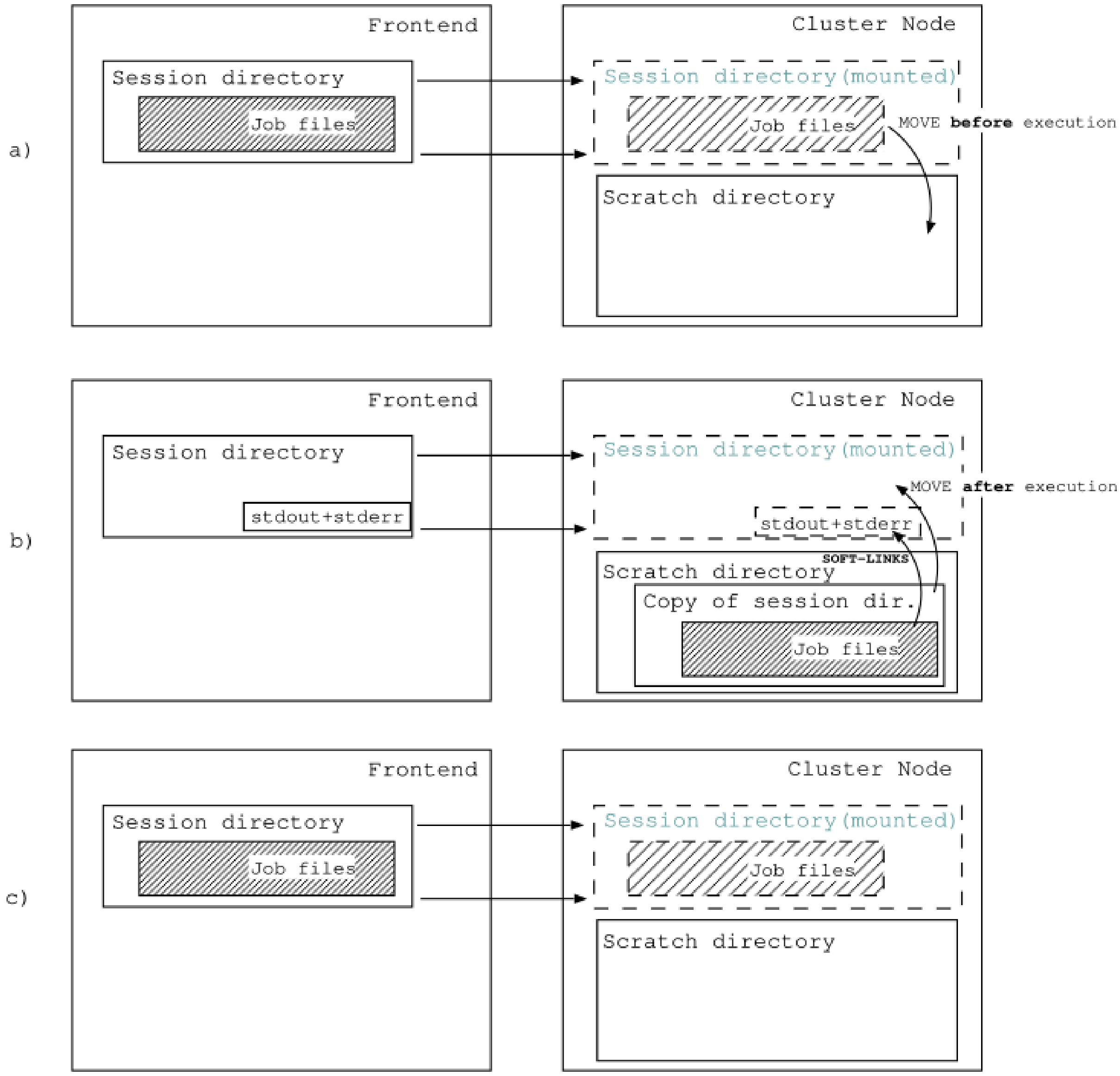


Fig. 4.5: Sessiondir is shared between ARC CE and WNs, scratchdir is defined and available only on nodes.

Configuration:

```
[lrms]
movetool = rsync -av

[arex]
shared_filesystem = yes
scratchdir = /mnt/scratch/arc
```

Variables set in the wrapper script:

```
RUNTIME_NODE_SEES_FRONTEND='yes'
RUNTIME_LOCAL_SCRATCH_DIR='/mnt/scratch/arc'
RUNTIME_LOCAL_SCRATCH_MOVE_TOOL='rsync -av'
```

Session directory is NOT shared, WN scratch directory is NOT shared

If the session directory is not shared the data movement between ARC CE and WN can be done by one or a combination of these two:

- **means of LRMS backend**
  How to implement the data movement between the ARC CE and WN depends on the particular batch system backend used.
  This is not described here, please refer to your batch system manuals.
  For example, in *PBS* this corresponds to `#PBS -W stagein` options. In *SLURM* it could be custom *prolog* and *epilog* scripts.
- **A custom made** *RTE*
  This topic is currently not part of this guide, but it is implemented in some ARC setup, so please ask for support from the community.

> **Warning:** This setup is not recommended, and is not actively supported by ARC. You can as always reach out to the ARC community for help and support if you need this setup.

Two known use cases of setting up ARC to work with such setup are described below.

Use-case 1: Static path to WN scratch directory

In this use case the path of the scratch directory on every WN is a static path that does not change depending on the job.

ARC will create a wrapper script that expects all the job files to be on the WN in the path `$RUNTIME_LOCAL_SCRATCH_DIR/<arc-job-id>`.

> **Warning:** In this use case, when the *job session directory* is not shared (`shared_filesystem = no`) *job scratch directory* MUST BE defined (`scratchdir = path`) to instruct LRMS where to find the files. Otherwise job submission to LRMS will fail.

After all input files are gathered in the *job session directory* on ARC CE, the LRMS or the custom RTE copies files to the *job scratch directory* on WN (Figure *a)*).

The job performs all I/O using local *job scratch directory*. After execution all declared output files (including *stdout* and *stderr*) must be staged out to the *job session directory* on ARC CE by LRMS or custom RTE (Figure *b)*).

When the output job files are available in the *job session directory* on ARC CE they are ready to be uploaded to external storage elements or be downloaded by the user (Figure *c)*).

Configuration:

```
[arex]
shared_filesystem = no
scratchdir = /mnt/scratch/arc
```

Variables set in the wrapper script:

```
RUNTIME_NODE_SEES_FRONTEND=''
RUNTIME_LOCAL_SCRATCH_DIR='/mnt/scratch/arc'
```

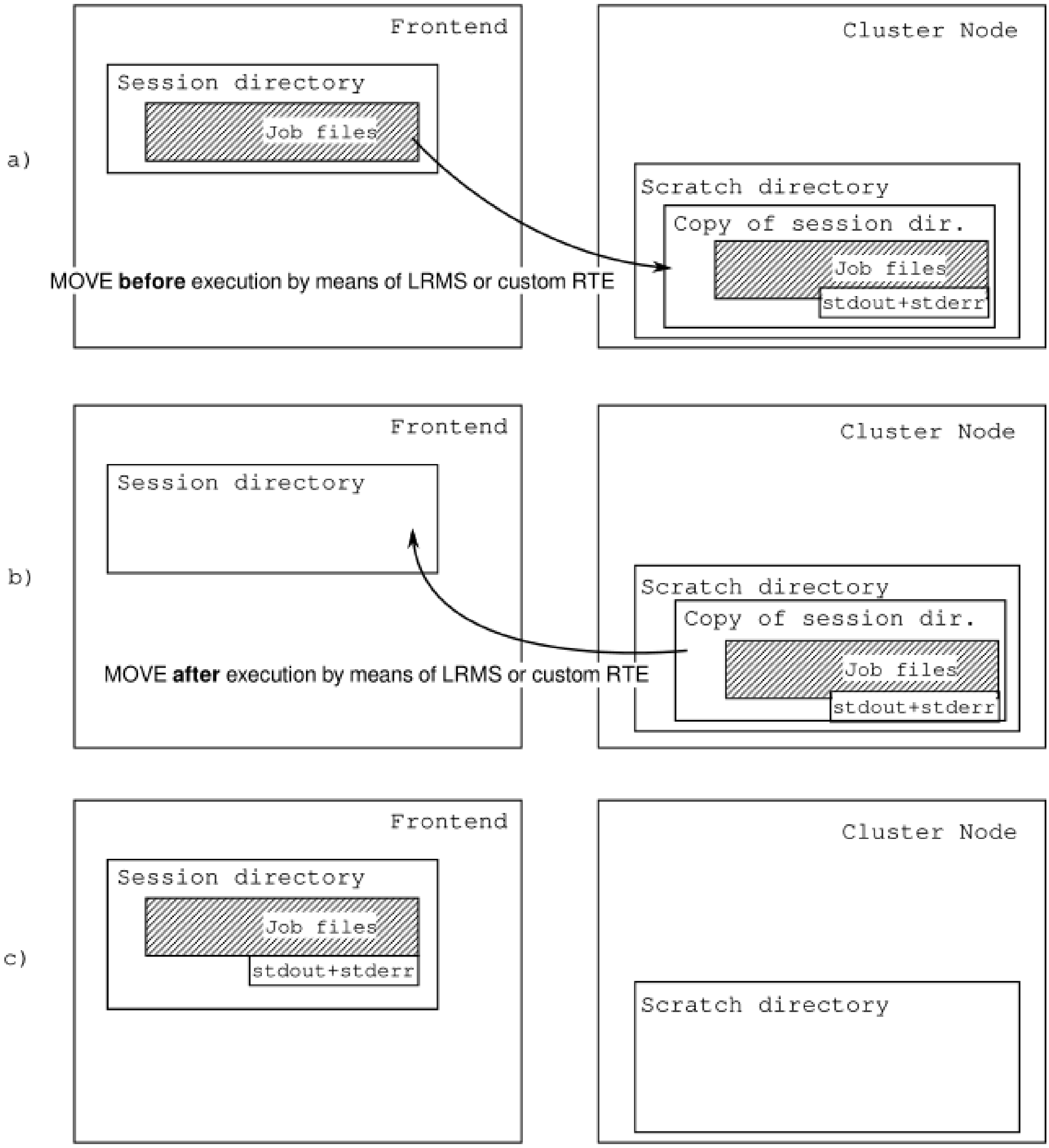


Fig. 4.6: NEITHER sessiondir NOR WN scratchdir are shared between ARC CE and WNs.

Use-case 2: Dynamic job path to the WN scratch directory

In this use case the LRMS generates a dynamic path for each submitted job. This path is not known to ARC during the creation time of the wrapper script, therefore it must be obtained in some way at runtime, after the job is submitted.

A sysadmin may already have defined such variable in their prolog/epilog scripts.

The workflow is very similar to the one described in the picture of the static path use case, with the difference that the **name** of the LRMS specific environment variable that contains the dynamic path can be defined and configured in a special *RTE* called *ENV/LRMS-SCRATCH*, so that it will be used at runtime to generate a path of the kind `<lrms-job-id>/<arc-job-id>`.

The sysadmin can now create a custom RTE with LRMS specific commands and make use of the variable above when referencing the dynamic dir.

In this scenario the arc.conf variable *scratchdir* can be defined but it is not mandatory, as the LRMS may have a completely different base path for each node/job.

However, if configured, the scratchdir path will be prefixed to the dynamic path, as in: `<arc-conf-scratchdir-var>/<lrms-job-id>/<arc-job-id>`

Please refer to the documentation of the bundled *ENV/LRMS-SCRATCH* runtime environment for more details.

Configuration:

```
[arex]
shared_filesystem = no
```

Variables set in the wrapper script:

```
RUNTIME_NODE_SEES_FRONTEND=''
RUNTIME_LOCAL_SCRATCH_DIR=''
```

Accounting with legacy JURA

WARNING: This component was deprecated in ARC 6.4 and completely removed in ARC 6.8!

> **Warning:** Information in **this chapter is relevant only for 6.0-6.3 ARC releases**.
>
> Starting from ARC 6.4 release the *Accounting Subsystem* with local accounting database will be introduced. Make sure you are reading the documentation that match your ARC CE release version.

The *Job Usage Reporter of ARC* (JURA) is a component which is capable to create standard-complient usage records from job usage information provided by the A-REX (*Job Log* files) and send the records to remote accounting services.

JURA is capable of creating two types of usage records from the job log files:

- Usage Record 1.0 (UR) XML format to be sent to an SGAS LUTS (Logging and Usage Tracking Service)
- Compute Accounting Record (CAR) XML format to be sent to APEL

Overview

Figure shows the overview of accouting workflow in ARC. More details can be found in *Legacy JURA Accounting Technical Details*.

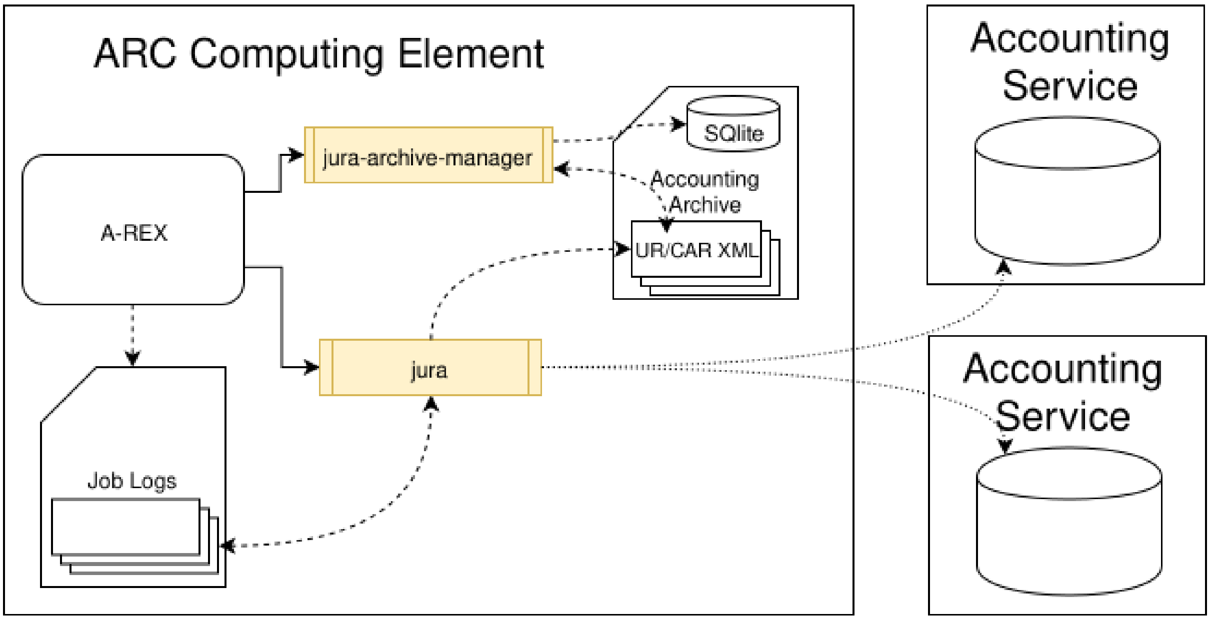


Fig. 4.7: Accounting records processing in ARC CE: creating, processing, publishing and archiving

When accounting is enabled, the sequence of events is the following:

1. A-REX writes the per-job accouting records to *Job Logs* directory (`logs` directory inside the *controldir*).
2. A-REX periodically runs `jura`. Default and minimum period is hourly but can be increased with *urdelivery_frequency* option.
3. Jura reads jobs accounting information from *Job Logs* and depending on the targets cofigured in *arc.conf* creates accounting records according to required format.
4. Jura publishes accounting records to the configured accounting services.
5. If records archiving is enabled, JURA also puts the records into *Accounting Archive*
6. When archiving functionaly is used, A-REX also runs the `jura-archive-manager` process which manages the archive structure and indexes the records in the local database. The local accounting database can then be used to *lookup statistics* or to *republish data*.

Enabling accounting records reporing

JURA is a stand-alone binary application that is a part of the A-REX package and can be enabled with *[arex/jura] block*.

JURA has a dedicated log file as defined by the *logfile* option. Log rotation is set for the default `/var/log/arc/jura.log` location. Accounting logs can be also viewed with `arcctl`:

Changed in version 6.4: Extra `legacy` argument is required to access legacy accounting subsystem functionality. Use `arcctl accounting legacy logs`.

```
[root ~]# arcctl accounting logs
```

Accounting services for sending the records are configured with dedicated sub-blocks. You need to define a separate block with an unique targetname for every target server used.

The usage record of each job will be reported to all of the destinations, unless the `vofiler` option is configured for some of them to filter records by VO name.

Configuring reporting to SGAS

The *SGAS sub-block* enables and configures an SGAS accounting server as a target destination to which JURA will send properly formatted Usage Record 1.0 (UR) XML records.

The *targeturl* option is the only mandatory parameter to configure an SGAS target. In the specific setup cases you can also apply *VO filtering* and *set prefix* for local job IDs.

Example:

```
[arex/jura/sgas: NeIC]
targeturl = https://grid.uio.no:8001/logger
urbatchsize = 80
```

Configuring reporting to APEL

The *APEL sub-block* enables and configures an APEL accounting server as a target destination to which JURA will send properly formatted Compute Accounting Record (CAR) XML records.

The *targeturl* option defines the APEL broker URL to send records to. Unlike the native APEL client that relies on Top-BDII infrastructure in runtime to find the blockers, ARC uses a dedicated target URL in the configuration. However, it is possible to look up available targets with `arcctl`:

```
[root ~]# arcctl accounting apel-brokers
http://mq.cro-ngi.hr:6163/
http://broker-prod1.argo.grnet.gr:6163/
[root ~]# arcctl accounting apel-brokers --ssl
https://mq.cro-ngi.hr:6162/
https://broker-prod1.argo.grnet.gr:6162/
```

SSL or non-SSL connecton should be defined accordingly with the *use_ssl* option.

You also need the *GOCDB name* of the resource and the *APEL topic* to which JURA will publish accounting records. For a correct production accounting setup it is recommended to specify resource *benchmarking results*.

Example:

```
[arex/jura/apel: EGI]
targeturl = https://mq.cro-ngi.hr:6162
topic = /queue/global.accounting.cpu.central
gocdb_name = RAL-LCG2
benchmark_type = Si2k
benchmark_value = 2625.00
use_ssl = yes
```

Note: Pubishing to APEL relies on SSM framework that has own logfile viewable with `arcctl accounting logs --ssm`.

Records archiving

The archiving functionality allows to store generated usage records in a specified directory on the disk.

The *[arex/jura/archiving] block* enables this functionality. Directory path for the archivedJ ura usage records is defined with the *archivedir* option.

Once archive is enabled, Jura writes generated usage records to files named `usagerecord[CAR].<jobid>.<random>`. If a A-REX job log file is processed repeatedly – for example because of temporary connection failures to the remote accounting service – and a respective usage record archive file already exists, then the usage record is not generated again. Instead, the content of the archive file is used without change (the creation time stamp is also retained).

Note: Records archive can grow very rapidly on heavy loaded sites with huge ammount of jobs comming daily. To limit the archiving time, use the *archivettl* option.

To better organize archived records and index the accounting data in the local database, the `jura-archive-manager` helper process is run periodically by A-REX when archive is enabled.

The archive manager:

- reads the resource usage data and stores it to an indexed SQLite database file `accounting.db`.
- moves files according to the following directory structure: `YYYY-MM/DD/<jj>/<kkkkk>.<random>.{UR|CAR}`, where `jj` is first 2 characters of `<jobid>` and `kkkkk` is the rest of `<jobid>`

Lookup local accounting data

Data in the local accounting database can be viewed with `arcctl` for each type of destination. Timeframe of interest and many other filters can be specified, e.g.:

Changed in version 6.3: Extra `legacy` argument is required to access legacy accounting subsystem functionlity. Use `arcctl accounting legacy stats`.

```
[root ~]# arcctl accounting stats -t apel -b 2018-11-01 -e 2018-11-30
Statistics for APEL jobs from 2018-11-01 00:00:02 till 2018-11-29 23:59:13:
  Number of jobs:           355168
  Total WallTime: 48146 days, 18:06:50
  Total CPUTime:  71927 days, 10:33:33
[root ~]# arcctl accounting stats -t apel -v
ops
belle
atlas
ops.ndgf.org
[root ~]# arcctl accounting stats -t apel --filter-vo belle
Statistics for APEL jobs from 2015-04-13 07:28:52 till 2019-04-02 00:25:57:
  Number of jobs:           869400
  Total WallTime: 216866 days, 3:33:33
  Total CPUTime:  193604 days, 19:40:58
```

Republishing records

When something goes wrong with accounting services, network etc, it may be needed to republish local records again.

Warning: Republishing is ONLY possible when *Records archiving* is enabled and records still exist for the required time period.

The *ARC Control Tool* should be used to republish data. It is tightly integrated with the accounting archive database: `arcctl` will look up needed records in the database for the defined republishing period, prepare them for republishing and than invoke `jura` with correct configuration settings.

Warning: It is strongly recommended to AVOID running republishing in ARC5-way, executing the `jura` command manually. Even if you manage to get it working, it can cause unpredictable results.

Define the timeframe of interest and run the command to rebuplish data:

Changed in version 6.4: Extra `legacy` argument is required to access legacy accounting subsystem functionality. Use `arcctl accounting legacy republish`.

```
[root ~]# arcctl accounting republish -b 2018-12-01 -e 2018-12-31 -s https://sgas.
↪grid.org.ua:8443/sgas
[root ~]# arcctl -d DEBUG accounting republish -b 2019-04-01 -e 2019-04-10 -a https://
↪mq.cro-ngi.hr:6162
```

Next generation ARC accounting: arcctl example queries

General statistic queries examples

```
[root arc-dev]# arcctl accounting stats
A-REX Accounting Statistics:
  Number of Jobs: 238
  Execution timeframe: 2019-07-09 18:42:18 - 2019-07-12 15:16:56
  Total WallTime: 6 days, 1:35:37
  Total CPUTime: 4:19:45 (including 0:00:00 of kernel time)
  Data staged in: 62.1M
  Data staged out: 1.7K

[root arc-dev]# arcctl accounting stats -o
brief            data-staged-in   jobcount         json             walltime
cputime          data-staged-out  jobids           users            wlcgvos
[root arc-dev]# arcctl accounting stats -o wlcgvos
moldyngrid
ops
matmoden

[root arc-dev]# arcctl accounting stats --filter-vo ops -o jobcount
222

[root arc-dev]# arcctl accounting stats --filter-vo ops --start-from 2019-07-11
A-REX Accounting Statistics:
  Number of Jobs: 131
  Execution timeframe: 2019-07-11 00:16:08 - 2019-07-12 15:33:26
```

(continues on next page)

(continued from previous page)

```
  Total WallTime: 0:20:38
  Total CPUTime: 0:02:02 (including 0:00:00 of kernel time)
  Data staged in: 219.3K
  Data staged out: 1.0K

[root arc-dev]# arcctl accounting stats --filter-queue grid \
> --filter-user /DC\=org/DC\=ugrid/O\=people/O\=KNU/CN\=Andrii\ Salnikov
A-REX Accounting Statistics:
  Number of Jobs: 8
  Execution timeframe: 2019-07-09 18:42:18 - 2019-07-10 17:31:59
  Total WallTime: 0:01:20
  Total CPUTime: 0:00:00 (including 0:00:00 of kernel time)
  Data staged in: 61.7M
  Data staged out: 0

[root arc-dev]# arcctl accounting stats --filter-endpoint org.ogf.glue.emies.
↪activitycreation \
> --end-till 2019-07-10 -o data-staged-out
75

[root arc-dev]# arcctl accounting stats  --filter-state failed --filter-vo moldyngrid␣
↪-o data-staged-in
21574172

[root arc-dev]# arcctl accounting stats --filter-extra rte ENV/PROXY -o json | jq .
{
  "stageout": 0,
  "rangeend": 1562938441,
  "count": 86,
  "cpukerneltime": 0,
  "users": [
    "/DC=EU/DC=EGI/C=HR/O=Robots/O=SRCE/CN=Robot:argo-egi@cro-ngi.hr",
    "/DC=EU/DC=EGI/C=GR/O=Robots/O=Greek Research and Technology Network/
↪CN=Robot:argo-egi@grnet.gr"
  ],
  "wlcgvos": [
    "ops"
  ],
  "cpuusertime": 0,
  "cputime": 0,
  "rangestart": 1562688395,
  "stagein": 398180,
  "walltime": 172
}

[root arc-dev]# arcctl accounting stats --filter-extra jobname "test 04"
A-REX Accounting Statistics:
  Number of Jobs: 4
  Execution timeframe: 2019-07-10 13:40:00 - 2019-07-10 17:31:59
  Total WallTime: 0:00:40
  Total CPUTime: 0:00:00 (including 0:00:00 of kernel time)
  Data staged in: 30.9M
  Data staged out: 0

[root arc-dev]# arcctl accounting stats --filter-extra jobname "test 04" -o jobids
g3AMDmCCP5unf5481mks8bjnABFKDmABFKDmN9IKDmIBFKDmnjU0ym
```

(continues on next page)

(continued from previous page)

```
nEhMDmvqQ5unf5481mks8bjnABFKDmABFKDmN9IKDmZBFKDmtXEYGn
UufLDmmmS5unf5481mks8bjnABFKDmABFKDmmSMKDmNBFKDm9nRnVo
bdoLDmCnS5unf5481mks8bjnABFKDmABFKDmmSMKDmaBFKDmfKldDn
```

Job-specific accounting queries

```
[root arc-dev]# arcctl accounting job events␣
↪bdoLDmCnS5unf5481mks8bjnABFKDmABFKDmmSMKDmaBFKDmfKldDn
2019-07-10 17:30:26 ACCEPTED
2019-07-10 17:30:26 PREPARING
2019-07-10 17:30:26 DTRDOWNLOADSTART
2019-07-10 17:30:27 SUBMIT
2019-07-10 17:30:27 DTRDOWNLOADEND
2019-07-10 17:30:29 INLRMS
2019-07-10 17:30:47 LRMSSTART
2019-07-10 17:30:57 LRMSEND
2019-07-10 17:31:59 FINISHING
2019-07-10 17:31:59 FINISHED

[root arc-dev]# arcctl accounting job transfers ␣
↪bdoLDmCnS5unf5481mks8bjnABFKDmABFKDmmSMKDmaBFKDmfKldDn
Data transfers (downloads) performed during A-REX stage-in:
  http://download.nordugrid.org:80/packages/nordugrid-arc/releases/6.1.0/src/
↪nordugrid-arc-6.1.0.tar.gz (from cache):
    Size: 5.2M
    Download timeframe: 2019-07-10 17:30:26 - 2019-07-10 17:30:27
  http://download.nordugrid.org:80/packages/nordugrid-arc/releases/6.0.0/src/
↪nordugrid-arc-6.0.0.tar.gz (from cache):
    Size: 5.1M
    Download timeframe: 2019-07-10 17:30:26 - 2019-07-10 17:30:27
No stage-out data transfers (uploads) performed by A-REX.

[root arc-dev]# arcctl accounting job info ␣
↪bdoLDmCnS5unf5481mks8bjnABFKDmABFKDmmSMKDmaBFKDmfKldDn
Job bdoLDmCnS5unf5481mks8bjnABFKDmABFKDmmSMKDmaBFKDmfKldDn accounting info:
===========================================================================
Job description:
  Job was submitted at 2019-07-10 17:30:26 via "org.ogf.glue.emies.activitycreation"␣
↪interface using "https://arc.matmoden.kiev.ua:443/arex" endpoint.
  Job owned by "/DC=org/DC=ugrid/O=people/O=KNU/CN=Andrii Salnikov" as a member of
↪"moldyngrid" WLCG VO.
  It was targeted to the "grid" queue with "381309.head.energrid.ipme.kiev.ua" LRMS␣
↪ID.
  Job completed with exit code 0 at 2019-07-10 17:31:59.
  Following job properties are recorded:
    Localuser: prdmdg24
    Clienthost: 130.235.185.244:58458
    Lrms: pbs
    Nodenames: n2
    Jobname: test 04
Resource usage:
  Execution timeframe: 2019-07-10 17:30:26 - 2019-07-10 17:31:59
  Used WallTime: 10
  Used CPUTime: 0 (including 0 of kernel time)
```

(continues on next page)

(continued from previous page)

```
  Used WN Scratch: 0
  Max physical memory: 528
  Max virtual memory: 0
  Used CPUs: 1 on 1 node(s)
  Data staged in: 0
  Data staged out: 0
Used RunTime Environments:
  There are no RTEs used by the job.
Auth token attributes provided:
  VOMS FQAN: /moldyngrid
  VOMS FQAN: /moldyngrid/Role=VO-Admin
  VOMS FQAN: /moldyngrid/Role=production
  VOMS FQAN: /moldyngrid/edu
```

About benchmarks and accounting publishing

After the changes in the accounting system in ARC 6.4.0 there were some issues related to missing benchmark values in the accounting records. Some of them are related to a bug that unfortunately snuck into the codebase, however sites can have issues with benchmarks for other reasons too.

This page is aimed to clarify how benchmarks are recorded and propageted, in what situations problems can occur, and how to fix them.

If you see `HEPSPEC 1.0 is being used` in the `jura.log` - (some of) your job records are missing benchmark values.

Follow the questions to clarify your case.

Which version of ARC you have?

Depending on the ARC version, there are several issues related to the benchmark values processing:

- ARC **< 6.4.0**:
  - completely different accounting codebase is in use, information in this document is irrelevant. General advise is definitely to update to the recent version, bugs in the old codebase will not be fixed.
- ARC **< 6.5.0**:
  - bug with handling benchmark values in the publishing code
  - HTCondor backend with non-shared filesystem have missing benchmarks
  - APEL summaries query performance is degrading with increasing ammount of stored records
- ARC **< 6.8.0**:
  - HTCondor backend with non-shared filesystem have missing benchmarks
  - APEL summaries query performance is degrading with increasing ammount of stored records
- ARC **>= 6.8.0**:
  - all known benchmark related issues are fixed
  - if you see `HEPSPEC 1.0 is being used` there is some valid reason for it, including missconfiguration

Note: It is important to understand that benchmark values **are part of the job accounting record**. Benchmark data in the job accounting record is **defined on the job start time** and stored when job is just finished.

If you have jobs started before update of ARC or configuration fix (depending on your case) - you need to manually fix already stored records. There is no way changes will be applied retrospectively to already stored records automatically.

What are the reasons for missed benchmark value in the job records?

There are several valid reasons when you will see `HEPSPEC 1.0 is being used` message:

1. The job was started when ARC was at version < 6.5
2. The job was started when the *[queue:name] block* in `arc.conf` had no proper benchmark defined
3. The permissions or other issues (including HTCondor backend bug with non-shared filesystem in ARC < 6.8.0) prevents the writing of `.diag` files on the worker nodes
4. The job failed in LRMS before even execution of initial jobscript wrapper part (node failure, etc).

The last issue is simply can happen very rarely and nothing to do with it, but such jobs has zero cputime, so benchmark is really irrelevant.

Nevertheless, to eliminate the log message that annoys admins and avoid aditional type of summary records during the publishing, ARC 6.8.0 introduced *benchmark* option in *[lrms] block* that will be used as a fallback if the benchmark metric is missing in the job data.

Warning: Again! The *benchmark* option in *[lrms] block* has no influence on already stored records. It is during **storing** time but **NOT publishing** time.

How to fix missing benchmark values manually?

Already stored accounting records that has no benchmark values can be fixed by issuing an `sqlite` query that adds benchmark value. Following example assumes the controldir is the default `/var/spool/arc/jobstatus`:

```
[root ~]# sqlite3 /var/spool/arc/jobstatus/accounting/accounting.db "insert into␣
↪JobExtraInfo
  (RecordID, InfoKey, InfoValue) select distinct RecordID, 'benchmark', 'HEPSPEC:12.1'
  from JobExtraInfo where RecordID not in
  (select RecordID from JobExtraInfo where InfoKey='benchmark');"
```

If you discover that some records use the default benchmark of `HEPSPEC 1.0` instead of your desired benchmark value in `arc.conf` (e.g. you had added benchmark values after job start) you can update the values as well:

```
[root ~]# sqlite3 /var/spool/arc/jobstatus/accounting/accounting.db "update␣
↪JobExtraInfo
  set InfoValue = 'HEPSPEC:12.1' where InfoKey = 'benchmark' and InfoValue =
↪'HEPSPEC:1.0';"
```

What should I know to avoid running into bechmark issues?

To understand how the `HEPSPEC 1.0 is being used` occurs in the `jura.log` there are 3 points to understand:

1. JURA is only the publisher and it sends the data about the jobs stored in the local ARC accounting database. NO values from `arc.conf` (apart from where to publish records) are used during publishing.
2. Info about benchmarks is part of the job accounting data stored in the ARC local accounting database when the job is in the finishing state. Moreover, the static data, including the benchmark defined in `arc.conf` are defined during the jobscript generation (job start time). Any update to `arc.conf` AFTER the job start HAS NO EFFECT on already stored records.
3. In case of publishing to APEL, the default method to use is *APEL summaries*. This means that `jura` will send (update) the total counters about last 2 month of data that aggregated per VO, DN, Endpoint (include queue) and Benchmark! CONSEQUENTLY if any single job within 2 month timeframe is missing the benchmark data - this warning about using HEPSPEC 1.0 will be there!

> **Warning:** **For ARC < 6.8.0** the APEL summary query includes grouping by benchmark which was out of scope of the initial ARC accounting database design. The extra tables join is harmful to performance on heavy loaded sites! The recommended mitigation to save ARC CE CPU cycles is to go back to individual usage records publishing with `apel_messages = urs` option.
>
> **In the ARC 6.8.0** the APEL summary querying were improved and performance hit is not that valuable. You can use summaries on the heavy loaded sites as well.

It is also important to understand the **chain of benchmark propagation** for the issues troubleshooting:

1. The value of *benchmark* defined in the *[queue:name] block* block in `arc.conf` is written to the `.diag` file as it is on the frontend (controldir).
2. The `.diag` file from the control directory is copied next to the job's session directory and either shared to worker node (shared sessiondir case) or moved by LRMS. See more details about shared vs non-shared sessiondir in the *Job scratch area* document.
3. During job execution jobscript writed data to `.diag` on the worker node. This includes benchmark that can be redefined in runtime (e.g. by *RunTime Environments in ARC6*)
4. After job completion the `.diag` from worker node is moved to the frontend's session directory if sessiondir is not shared.
5. On the frontend `.diag` from session directory merged with `.diag` in the control directory and more information from the LRMS accounting is added to it.
6. A-REX parse the `.diag` in the control directory and store data to the database. From ARC 6.8.0 at this stage the default fallback *benchmark* is added to the data from `arc.conf` if missing in the `.diag`.

So, should I do something if I see "HEPSPEC 1.0 is being used" message?

If this is a rare single job that just failed in LRMS before writing the accounting data - nothing to worry about.

But if it annoys you, you can fix even single job data manually as describer above. Or starting from ARC 6.8.0 you can define the fallback *benchmark* to aviod it completely.

To identify how many jobs are missing benchmark data in the database, run the following query:

```
[root ~]# sqlite3 /var/spool/arc/jobstatus/accounting/accounting.db "select JobID␣
↪from AAR
  where RecordID not in ( select RecordID from JobExtraInfo where InfoKey='benchmark
↪');"
```

This returns list of the job IDs with missing benchmark data. Than you can use:

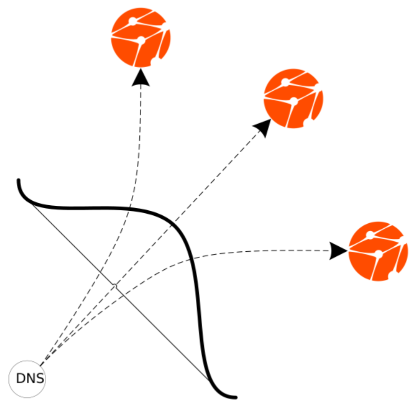
```
[root ~]# arcctl accounting job info <JobID>
```

to find what are those jobs.

If there are many, than something definitely goes wrong and you should:

1. Check if you are facing the knows issues if you are running ARC < 6.8.0. ARC update + manual records fix will solve your problem in this case.
2. Check the `arc.conf` syntax in respect to benchmark. It should be defined in the *[queue:name] block* and use either `HEPSPEC` or `si2k`. Manual records fix for already stored records is needed anyway.
3. Check the `.diag` file contains information, use `arcctl` to check stored data, check A-REX logs for any hints.
4. Open a bugzilla ticket if nothing helps.

4.3 ARCHERY



The **ARC Hierarchical Endpoints Registry (ARCHERY)** is a novel DNS-based registry for e-Science infrastructures to store service endpoints and execution environment definitions.

ARCHERY inherits all the benefits of the DNS infrastructure since deployment, management and operation of the system is fully relying on the DNS protocol.

ARCHERY implements a minimalistic *data model* representing services and their endpoints within e-Infrastructures and embeds the service endpoint information directly into the DNS database.

All the services available within an e-Infrastructure are indexed by means of an ARCHERY instance.

Nordugrid Collaboration, the organization behind the ARC software, operates a top-level ARCHERY hosted within `nordugrid.org` DNS zone.

Similar hierarchical service catalogues can be deployed by any organization or project under a dedicated DNS zone. Instructions for howto create such registry is given *here*.

Note: Sites are NOT required to run any reristration service on an ARC6 CE.

Read *this document* to understand what needs to be done so that an ARC6 CE would appear in an ARCHERY registry.

Previously a different technology was used as a service registry that was based on LDAP (see details here).

Following documents cover the generic aspects of the ARCHERY deployment and operation, including the very specific case of NorduGrid infrastructure:

4.3.1 ARCHERY deployment for NorduGrid

The NorduGrid topology for ARCHERY

The root element of the hierarchical service registry tree is the `nordugrid.org` DNS zone hosted by Niels Bohr Institutet.

The root zone holds references to:

- *per-country registries* in the dedicated sub-zones under `<country>.archery.nordugrid.org` domains or
- *projects-based registries* under project-owned specific DNS zones (e.g. `archery.myproject.org`)

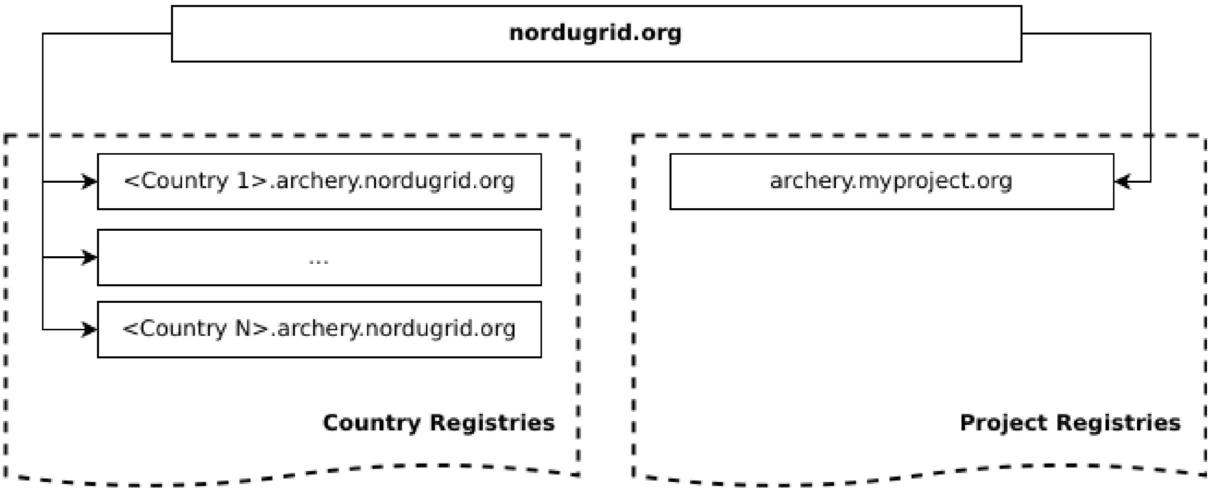


Fig. 4.8: The NorduGrid ARCHERY topology model

Per-Country ARCHERY DNS zones of the NorduGrid

Services indexed by NorduGrid are naturally belonging to a country. Therefore the deployed NorduGrid ARCHERY topology contains a set of contry-level DNS zones.

These per-country ARCHERY zones are maintained in a collaborative manner by different organizations responsible for specific country services.

A country-level DNS subzone was created for every country that previously run an ARC CE (services were taken from the monitor and GOCDB)

Note: Zones whose maintenance is *not delegated yet* are served by Niels Bohr Institutet along with the root `nordugrid.org` zone.

The summary table shows the country-level ARCHERY entry points together with the administrating contact details.

Country	Main entry point	Additional entry points	Administration contact
Denmark	`dk.archery.nordugrid.org`		Petter Urkedal, urkedal@nbi.dk
Estonia	`ee.archery.nordugrid.org`		Zone is not delegated yet
Finland	`fi.archery.nordugrid.org`		Zone is not delegated yet
Hungary	`hu.archery.nordugrid.org`		Zone is not delegated yet
Portugal and Spain	`pt.archery.nordugrid.org`		Zone is not delegated yet
Lithuania	`lt.archery.nordugrid.org`		Zone is not delegated yet
Norway	`no.archery.nordugrid.org`		Zone is not delegated yet
Slovakia	`sk.archery.nordugrid.org`		Zone is not delegated yet
Slovenia	`si.archery.nordugrid.org`		Zone is not delegated yet
Sweden	`se.archery.nordugrid.org`		Zone is not delegated yet
Switzerland	`ch.archery.nordugrid.org`		Zone is not delegated yet
Ukraine	`ua.archery.nordugrid.org`	`grid.org.ua`	Andrii Salnikov, archery@grid.org.ua
UK	`uk.archery.nordugrid.org`		Zone is not delegated yet

Contact us if you want to take over the administration of a contry sub-zone.

Project-based ARCHERY Registries of NorduGrid

Other way to organize services is to group them by the host project. ARCHERY allows services to be affiliated with multiple projects.

It is assumed that projects are administrating their own list of services and that information is stored under a dedicated ARCHERY project-owned DNS zone.

The summary table collects ARCHERY entry points and administration contact information for projects indexed by the NorduGrid.

Registry	Entry point	Administration contact
NeIC Services for ATLAS	`atlas.archery.ndgf.org`	Dmytro Karpenko, dmytrok@fys.uio.no

ARCHERY DNS zone administration tasks

As an ARCHERY DNS zone administrator you are expected to do following administration tasks:

- one-time initial *DNS zone setup*,
- defining and keeping up to date ARCHERY instance *services topology* in the configuration file,
- *operate regular runs* of *archery-manage* to keep DNS zone info up to date (e.g. cron-job).

Registering site in the ARCHERY

In ARCHERY sites are indexed top-to-bottom and the list of ARC CEs a maintained on the country-level and/or project-level.

As a site administrotor you should technically do nothing for site registration.

Warning: In contrast to previous indexing system, ARCHERY does not requires bottop-to-top registration.

Static list of CEs is managed by ARCHERY administrator. In case your CE is not yet in the registry[1] contact your country registry manager to find the details about the established registration procedures.

Contact information for known ARCHERY managers can be found below: they will provide details for the organizational procedures (if any) upon request.

You can also contact us in case of any questions.

4.3.2 Initial setup of ARCHERY instance

This document describes how to setup an ARCHERY instance in the DNS zone, including both DNS aspects and initial service endpoint information embedding.

The described procedures apply to either *NorduGrid deployment* case or any other ARCHERY instance setup flavors, e.g. country-level, project/experiment, virtual organization, etc.

Note: Administrative access to the DNS server is required during the ARCHERY initial setup to create DNS zone and configure remote access to this zone.

Further *operating* of ARCHERY instance DOES NOT require administrative access to the DNS.

Choose DNS zone name

ARCHERY registry instance is accessible by DNS name, that is used as an entry point (e.g. to *submit jobs* to the infrastructure).

This guide uses `example.org` as an organization-owned domain name that will contain a zone for ARCHERY setup.

Despite it is possible to put records directly into the `example.org` zone, for manageability and security[1] reasons it is strongly advised to have dedicated DNS sub-zone configured for ARCHERY records.

In the *ARCHERY deployment for NorduGrid* per-country sub-zone names are *pre-defined*.

Any other setup can use arbitrary sub-zone name. This guide uses `index.example.org` DNS zone for ARCHERY setup.

[1] You can check NorduGrid Monitor that takes ARC CE list from ARCHERY

[1] The zone content will be updated dynamically. Setting up a dedicated sub-zone is the easiest way to isolate ARCHERY data and completely eliminate the risk of changing other records that are not related to ARCHERY.

Generate transaction signature key to manage ARCHERY

ARCHERY implies the usage of Dynamic DNS (DDNS) updates to manage the data inside DNS zone.

The `archery-manage` tool, that is part of NorduGrid ARC middleware, will do the DDNS updates for you as described below.

DDNS eliminates typos, allows to maintain up-to-date information and keeps it simple operate ARCHERY without administrative access to the DNS server itself.

Use the following command to generate the key:

```
[user ~]$ dnssec-keygen -a HMAC-MD5 -b 256 -n USER archery
```

From the generated files you need a *secret* part to be included in both BIND and `archery-manage` configuration.

Define key in BIND

Note: Configuration examples in this guide are provided for BIND, however you can use any name server implementation configured in a similar way.

Create the `/etc/named/archery.key` and put the generated secret key inside:

```
key archery_key {
      algorithm hmac-md5;
      secret "S0Me+SecRet+keYgener@tedwithdnssec==";
};
```

Include the key definition into `/etc/named.conf` using the following config line:

```
include "/etc/named/archery.key";
```

Keyfile for archery-manage

Create a file `archery-manage.key` and put the generated key in the following format:

```
archery_key:S0Me+SecRet+keYgener@tedwithdnssec==
```

Configure DNS zone for ARCHERY

It is generally required by the worldwide DNS infrastructure that at least one slave DNS server should be configured for every DNS zone for reliability reasons.

In this guide the following addressed will be used:

- Primary (master) DNS: `ns1.example.org (192.0.2.100)`
- Seondary (slave) DNS: `ns2.example.org (192.0.2.200)`

Define zone in BIND

Add zone definition to master DNS `/etc/named.conf`:

```
zone "index.example.org." IN {
    type master;
    file "master/index.example.org.db";
    notify yes;
    also-notify {
        # slave DNS IP address
        192.0.2.200;
    };
    allow-transfer {
        # slave DNS IP address
        192.0.2.200;
    };
    allow-update {
        key archery_key;
    };
};
```

Please observer the `allow-update` directive that authorize DDNS update requests signed by `archery-manage` key.

The secondary DNS should be configured without any special options:

```
zone "index.example.org." IN {
    type slave;
    file "slave/index.example.org.db";
    masters {
        192.0.2.100;
    };
    allow-transfer {
        192.0.2.100;
    };
};
```

Create zonefile with a basic zone info

Basic zonefile requires only `SOA` record. It will be filled with data by `archery-manage` later.

You can use following zonefile template (timers are subject to arrange depending on the planned update frequency):

```
$ORIGIN example.org.
$TTL 3600
index   IN SOA    ns1.example.org. hostmaster.example.org. (
             2018082401 ; serial
             1200       ; refresh (20 minutes)
             180        ; retry (3 minutes)
             604800     ; expire (1 week)
             60         ; minimum (1 minute)
             )
        NS    ns1.example.org.
        NS    ns2.example.org.
```

Define records in parent zone

Note: If you setup a country-level index for the Nordugrid infrastructure such records are defined in the parent `nordugrid.org` zone.

Please provide your DNS setup information to us instead of following this section.

Define `NS` records[2] to refer to defined subzone:

```
$ORIGIN example.org.
# dedicated ARCHERY zone
index   NS ns1.example.org.
index   NS ns2.example.org.
```

To create an ARCHERY entry point in the parent zone you can:

- define `CNAME` record to use `example.org` as an entry point:

  ```
  $ORIGIN example.org.
  # ARCHERY entry point
  _archery CNAME _archery.index
  ```

- OR define `TXT` resource record with ARCHERY data pointing to group:

  ```
  $ORIGIN example.org.
  # ARCHERY entry point
  _archery TXT "u=index.example.org t=archery.group"
  ```

The same technique can be used to define any other DNS aliases for an entry point (even in a completele different domain).

`CNAME` is recommended if you referring only one ARCHERY group.

Populate ARCHERY DNS zone with initial data

The *archery-manage* is a dedicated tool to manage information in the ARCHERY DNS zone. It is available as `nordugrid-arc-archery-manage` package in the NorduGrid repositories and EPEL.

The tool uses the configuration file in the plain text of JSON format that define services topology. Configuration file syntax is very simple and described in details in the *operations guide*.

For initial data provisioning you should *run archery-manage once* supplying config, DNS zone and transaction signature key path *generated in the previous steps*.

After zone is populated with data *Operating ARCHERY instance* comes down to kepping it up to date, running periodic updates.

[2] If you plan to use a different out-of-scope domain names in `NS` don't forget to add glue `A` records.

4.3.3 Operating ARCHERY instance

JSON topology configuration file for ARCHERY

New in version 6.5.

JSON configuration file for `archery-manage` is a most flexible and customizable topology source for provisioning ARCHERY.

Following sections exmplains JSON syntax with the example use-cases.

Simple ARC services group

The simplest JSON config that describes single group of ARC services (matching the `arcce-list` plain-text source) can be written as follows. The `arc-services` keyword will trigger automatic endpoints discovery from ARC information system.

```
{
  "arc-services": [
     "arc1.example.org",
     "arc2.example.org"
  ]
}
```

Defining nested groups and arbitrarty services

The following JSON configures the 2 subgroups - `ARC` and `Storage`.

The `ARC` subgroup contains 2 ARC CE. ARC CE endpoints will be discovered automatically by means of querying information system.

The `Storage` subgroup contains 2 manually defined services. All service endpoints specification is included into JSON file.

```
{
   "groups": [
      {
        "id": "ARC",
        "arc-services": [ "arc1.example.org", "arc2.example.org" ]
      },
      {
        "id": "Storage",
        "services": [
          {
            "id": "se1.example.org",
            "type": "DPM",
            "endpoints": [ { "httpg://se1.example.org:8446/srm/managerv2": "SRM" } ]
          },
          {
            "id": "se2.exmple.org",
            "type": "dCache",
            "endpoints": [
              { "gsiftp://ccsrm.ihep.ac.cn:2811": "gsiftp" },
              { "httpg://ccsrm.ihep.ac.cn:8446/srm/managerv2": "SRM" },
              { "xroot://ccsrm.ihep.ac.cn:1094": "xroot" }
            ]
          }
```

(continues on next page)

(continued from previous page)

```
        ]
      }
   ]
}
```

Integration with BDII

The `archery-manage` is able to fetch services and their endpoints from Site-BDII.

The `external-source` keyword in JSON configuration allows to inject discovered services to the desired branch of the ARCHERY registry tree.

```
{
  "groups": [
    {
      "id": "NGI_XX",
      "type": "ngi.type",
      "groups": [
        {
          "id": "XX-Site1",
          "type": "site.type",
          "external-source": {
             "sitebdii": "ldap://bdii.site1.example.org:2170/GLUE2DomainID=XX-Site1,
↪o=glue",
          }
        },
        {
          "id": "XX-Site2",
          "type": "site.type",
          "external-source": {
             "sitebdii": "ldap://bdii.site2.example.org:2170/GLUE2DomainID=XX-Site2,
↪o=glue",
          }
        }
      ]
    },
    {
      "id": "NGI_YY",
      "type": "ngi.type",
      "groups": [
        {
          "id": "YY-WLCG-Site",
          "type": "site.type",
          "external-source": {
             "sitebdii": "ldap://bdii.wlcgsite.example.org:2170/GLUE2DomainID=YY-WLCG-
↪Site,o=glue",
             "filters": [ "vo:atlas,cms", "portscan" ]
          }
        }
      ]
    },
  ]
}
```

Plese notice that for `YY-WLCG-Site` site, during the information fetching from the Site-BDII, only services (and their endpoints) that match defined filters will be added to ARCHERY registry. In particular, it is services that according to published AccessPolicy allows to use endpoints by `atlas` or `cms` VOs and pass the network reachability test.

ARCHERY as a cummunity trusted software registry

New in version 6.5.

The `archery-manage` is able to provision community trusted software rigistry objects into the ARCHERY zone.

The JSON configuration should define the `software` and points to *RunTimeEnvironment scripts* location:

```
{
  "software": {
     "rtes_dir": "/home/community/rtesroot"
  }
}
```

Read more details in *this document*.

Referencing existing ARCHERY objects

ARCHERY allows to embedd overlapped trees into the DNS.

For example if particular service or group is already represented in DNS, it can be "linked" to another ARCHERY tree.

To specify such linking to existing objects the `external-archery-object` keyword should be defined in JSON config.

```
{
  "groups": [
    {
      "id": "Sweden",
      "type": "country",
      "arc-services": [
          "arc1.example.org",
          "arc2.example.org",
          "arc3.example.org"
      ]
    },
    {
      "external-archery-object": "dns://_archery.norway.nordugrid.org"
    },
    {
      "id": "Infrastructure Services",
      "services": [
        {
          "external-archery-object": "dns://voms.services.cern.ch"
        },
        {
          "id": "voms.ndgf.org",
          "type": "org.glite.voms",
          "endpoints": [
              { "voms://voms.ndgf.org:15015/nordugrid.org": "org.glite.voms" },
              { "https://voms.ndgf.org:8443/voms/nordugrid.org": "org.glite.voms-admin
↪" }
          ]
        }
```

(continues on next page)

(continued from previous page)

```
      ]
    }
  ]
}
```

Example JSON configuration above defines 3 subgroups in the ARCHERY:

- Sweden ARC services defined with `arc-services`
- Reference to existing country-level ARCHERY deployment under `norway.nordugrid.org` domain
- Common `Infrastructure Services` group that includes manual specification of `voms.ndgf.org` service endpoints and external VOMS service defined in another ARCHERY instance under `voms.services.cern.ch` domain.

Custom DNS data in ARCHERY zone controlled by archery-manage

It is possible to add arbitrary records into the ARCHERY DNS zone.

In particular it is useful to handle subzone delegation by declaring the NS and A records in the same configuration file.

Use `raw-dns` keyword to defiene array of the DNS records to be managed in the zone. Each record is represented by object that have `name`, DNS record `type` and `rdata` field that contains:

- string for a single record
- list of strings for set of records
- `null` if this record shold be removed from the DNS

```
{
  "groups": [
    {
      "id": "si",
      "arc-service": [
        "meja.arnes.si",
        "jost.arnes.si"
      ]
    },
    {
      "external-archery-object": "dns://_archery.ua.archery.nordugrid.org"
    }
  ],
  "raw-dns": [
     { "name": "ua", "type": "NS", "rdata": [
         "ns1.ua.archery.nordugrid.org.",
         "ns2.ua.archery.nordugrid.org."
       ]
     },
     { "name": "ns1.ua", "type": "A", "rdata": "194.44.249.94" },
     { "name": "ns2.ua", "type": "A", "rdata": "194.44.249.10" }
  ]
}
```

Example JSON configuration above defines:

- 2 subgroups: one for Slovenia (`si`) and one delegated for Ukraine
- NS and 2 glue A records to define subzone delegation to the different server

Managing ARCHERY data in DNS

ARCHERY reuses the existing DNS infrastructure services thus eliminating the need to develop, deploy and operate new set of custom dedicated services.

To simplify the process of *rendering* ARCHERY records and injecting those into the DNS, ARC6 comes with the *archery-manage* information management tool.

The `archery-manage` tool had been designed to simplify common operations with ARCHERY, including registry initial bootstrap, data migration from the other service registries and keeping dynamic information up to date. It discover service endpoints, querying the resources in the defined topology configuration, then based on this information generate DNS records suitable for ARCHERY operations.

Relying on the dynamic DNS updates feature, the `archery-manage` had been designed to modify data in the DNS zone remotely, eliminating the need to interact with DNS zone configuration itself after the initial setup.

This approach also makes the setup and access rights delegation simple, fully separating the DNS hosting itself and ARCHERY data management machine.

Example of the DNS zone configuration can be found in *Initial setup of ARCHERY instance* guide.

The archery-manage data processing

It is important to understand the `archery-manage` data processing chain to efficiently maintain the ARCHERY instance:

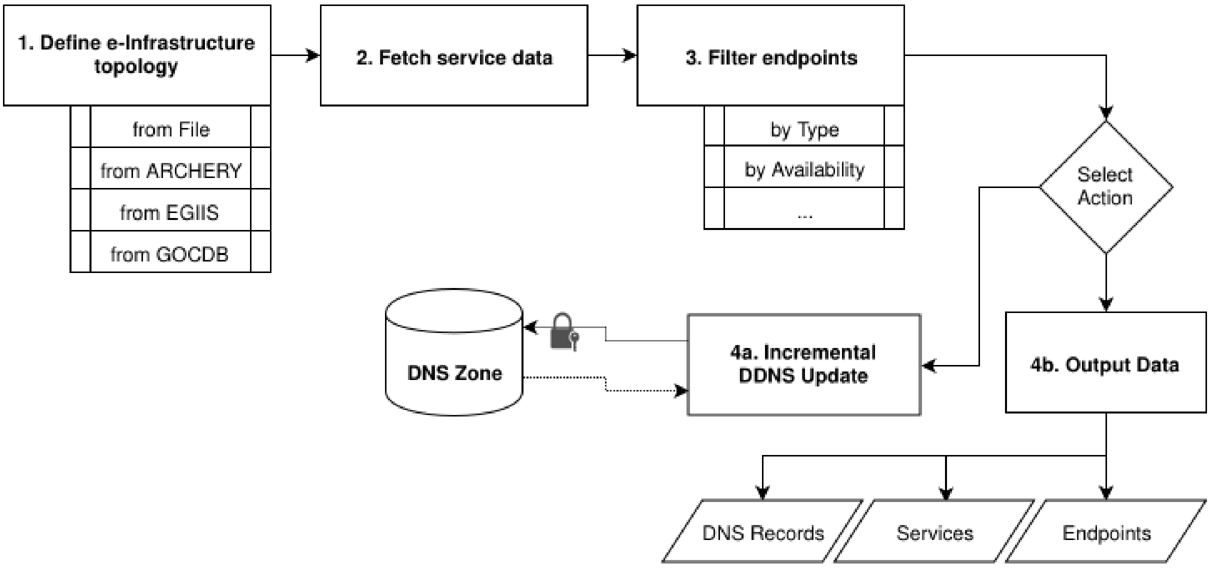


Fig. 4.9: The archery-manage data processing chain

Step 1. Define e-Infrastructure topology

Topology data defines how services are grouped within the e-Infrastructure. It comes either from a configuration file or from other databases that holds such information (including another ARCHERY instance). Interaction with already established databases (e.g. GOCDB) simplifies the integration and/or migration process.

Static list of ARC CE hostnames defined line-by-line in the plain text file is a trivial topology source that can form a group of computing elements that represent country or organization.

Step 2. Fetch service data

Topology database provides the pointers to information services that can be used to query service data. During this step the `archery-manage` tool discovers available endpoints and fetches service information.

For ARC CE hosts, the infosys LDAP GLUE2 is used to discover available endpoints with a fall-back to legacy LDAP Nordugrid schema querying.

Step 3. Filter endpoints

Set of discovered endpoints later passed to the filtering process. Based on the endpoint data (e.g. endpoint type), or additional testing (e.g. endpoint network availability check) endpoints that does not pass the filters are excluded.

Filters are extensible by design. The up to date list of currently supported filters can be obtailed passing `-f help` to `archery-manage`. The overview of mostly used filters can be found *in operations guide*.

Step 4a. Incremental DDNS Update

The target automation use-case is to push the discovered data to the DNS database. This is done automatically with Dynamic DNS updates over the network. Comparing the data already available in the DNS with discovered information, `archery-manage` constructs the incremental update that only applies the difference.

Step 4b. Output data

In addition to automatic updating of the DNS database, it is also possible to automate manual operation use-case. For this the tool following the same processing chain, can be used to print out endpoint or service lists with their types or the ARCHERY DNS records that can be manually added to DNS zone configurations.

The first and main task of the administrator of ARCHERY DNS zone is to keep services topology up to date.

Services topology is a static list of e-Infrastructure services (and optionaly their grouping) defined in the simple confiuration file (see below).

The *archery-manage* tool than uses the topology configuration file, fetch necessary information about the defined services and update data in *ARCHERY DNS zone*.

As an ARCHERY DNS zone administrator you should establish regular updates (e.g. cron-job or similar) to keep the information up to date.

The following sections will provide details about this operations tasks.

Define infrastructure services topology

In the simplest way, when a topolgy is a flat list of ARC services - the text file with hostnemaes specified line-by-line can be used.

More advanced topologies (e.g. grouping or non-ARC services) can be described in the *JSON topology configuration file for ARCHERY*.

Simple topology: flat list of ARC CEs

CE hostanames can be defined line-by-line in a plain text file:

```
ce01.example.org
ce02.example.org
arc6.example.org
```

Alternatively you can use *JSON config* syntax as well:

```
{
  "arc-services": [
     "ce01.example.org",
     "ce02.example.org",
     "arc6.example.org"
  ]
}
```

The *archery-manage* will automatically detect ARC service endpoints using information services on ARC CE and push it to DNS zone with an incremental DDNS update.

Hierarchical topology and non-ARC services

ARCHERY is NOT restricted to the ARC services only and can hold any e-Infrastructure service endpoints information.

To define such services the JSON configuration file, as a most flexible source of topology definition for *archery-manage* should be used.

Please consult the *JSON topology configuration file for ARCHERY* document for such topologies configuration examples.

Run archery-manage to update DNS zone information

To populate DNS zone with endpoint information based on the configured services topology you should run `archery-manage`:

```
[user ~]$ archery-manage -s arcce-list:ces.list --ddns-update \
            --domain index.example.org --ddns-master-ip 192.0.2.100 \
            --ddns-tsig-keyfile archery-manage.key
```

The `-s` key that define the *type* of configuration file to use and *path* to configuration file separated by colon. The `arcce-list` type corresponds to the plain text list of ARC services. The `json` type should be used for *JSON configuration file*.

The `--ddns-tsig-keyfile` points to the location of *generated transaction signature key*.

Options `--domain` and `--ddns-master-ip` defined the dedicated ARCHERY DNS zone name and master DNS server IP respectively.

Note: Updates are performed over the network, so you can run `archery-manage` on any host. It SHOULD NOT be the DNS server itself.

The *archery-manage* writes a logs to stderr that indicate the status of each operations performed during the run, including fetching, filtering and DDNS updates. You can increase logs verbosity with `-d` option.

Consider to add filters to `archery-manage`:

Filter: Port connectivity

Check network connectivity to endpoint TCP port and filter endpoints that do not pass this test.

It is advised to have the `portscan` filter enabled during the regular operations.

```
[user ~]$ archery-manage -s arcce-list:ce.list -f portscan ...
```

Filter: Endpoint type

By default any service endpoints types that are defined or discovered will be published. If you want to filter endpoints based on type there is a dedicated filter to apply.

For example, if you want only ARC REST endpoints in the ARCHERY:

```
[user ~]$ archery-manage -s json:/etc/archery.json -f type:org.nordugrid.arcrest ...
```

Both ARC REST and LDAP GLUE2 endpoints:

```
[user ~]$ archery-manage -s json:/etc/archery.json -f type:org.nordugrid.arcrest,org.
↪nordugrid.ldapglue2 ...
```

Filter: VO

Note: VO filtering is only available when service endpoints are automatically discovered based on information services (e.g. ARC services, Site-BDII, etc)

For project-based ARCHERY deployment it is also useful to filter endpoints based on VO access policy. Only endpoints that advertise specified VO support will be added[1]:

```
[user ~]$ archery-manage -s arcce-list:ce.list -f vo:exampleVO ...
```

Check the data is embedded to DNS

Once you populate ARCHERY DNS zone with data, you can try to *query the ARCHERY data from the DNS* to verify the update is working. This includes manual DNS queries or *job submission*.

Setup regular updates to ARCHERY

To keep information about endpoints up to date setup a CRON job (or Systemd Timer) to run `archery-manage` regularly.

In combination with at least *port filtering* this allows to eliminate stale endpoints and actualize information in registry.

[1] Resource information (`GLUE2PolicyRule` in GLUE2 and `nordugrid-cluster-acl` in Nordugrid LDAP) will be used as a source of supported VOs.

ARCHERY operations hints

Optimize information fetching frequency

Available services endpoints itself is not subject of rapid change. But the enpoint availability status is.

The *archery-manage* automatic endpoints fetching can be done less frequent as actual endpoint availability scan using the *JSON configuration* output:

1. Setup endpoints fetching to expanded JSON config.

   Run *archery-manage* with defined topology and desired filters but without DDNS update. This job can be run less frequent, e.g. once daily.

   ```
   [user ~]$ archery-manage -s json:/etc/archery.json -f vo:exampleVO -
   ↪o json > /etc/archery-expanded.json
   ```

2. Update information in the ARCHERY DNS zone using expanded JSON config.

   For actual ARCHERY DNS zone updates use obtained expanded JSON config. This job should be run more frequently (e.g. every 5 minutes) with a *portscan filter* to eliminate stale endpoints.

   ```
   [user ~]$ archery-manage -s json:/etc/archery-expanded.json -f␣
   ↪portscan \
             --ddns-update --ddns-tsig-keyfile archery-manage.key \
             --domain index.example.org --ddns-master-ip 192.0.2.100
   ```

Configure LDAP-monitor to use ARCHERY

The popular ARC CE monitoring tool - the LDAP monitor, can be used with ARCHERY out-of-the box.

It support fetching endpoints[2] from ARCHERY with the following configuration in `settings.inc`:

```
<?php
 $archery_list = array (
   array (
    "endpoint" => "example.org",
   )
 );
?>
```

4.3.4 Querying ARCHERY registry data

On the registry client level ARCHERY benefits from the distributed DNS caching on many levels, making the registry scalable.

Since DNS client is an integral part of any operating system, obtaining and processing service endpoint information from ARCHERY comes down to parsing and interpreting the data obtained from the DNS in accordance to the ARCHERY *data model rendering*.

There are several tools and options available to fetch ARCHERY data:

[2] Nordugrid monitor only works with `org.nordugrid.ldapng` data (and LDAP GLUE2 is experimental), so Nordugrid LDAP schema publishing is mandatory to use monitor.

The archery-manage tool as registry client

Using ARCHERY as a topology source for *archery-manage* allows to use tool output capabilities as a registry client. For example:

```
[user ~]$ archery-manage -s archery:moldyngrid.org -o arc-CEs
golowood.mao.kiev.ua
arc.imbg.org.ua
arc.univ.kiev.ua
uagrid.org.ua
grid.isma.kharkov.ua

[user ~]$ archery-manage -s archery:egi.grid.org.ua -o services
glue:sehn02.atlas.ualberta.ca/data                           : org.dcache.storage
glue:lcg-se1.sfu.computecanada.ca/data                       : org.dcache.storage
urn:ogf:ComputingService:lcg-ce1.sfu.computecanada.ca:arex   : org.nordugrid.arex
urn:ogf:ComputingService:lcg-ce2.sfu.computecanada.ca:arex   : org.nordugrid.arex
neutsrv1.triumf.ca_bdii-site_497425383                       : bdii_site
<output omitted>

[user ~]$ archery-manage -s archery:nordugrid.org -o endpoints --json | jq .
[
{
  "t": "org.nordugrid.gridftpjob",
  "u": "gsiftp://deckard.dcsc.ku.dk:2811/jobs",
  "rr_data": "u=gsiftp://deckard.dcsc.ku.dk:2811/jobs t=org.nordugrid.gridftpjob"
},
{
  "t": "org.nordugrid.ldapglue2",
  "u": "ldap://deckard.dcsc.ku.dk:2135/o=glue",
  "rr_data": "u=ldap://deckard.dcsc.ku.dk:2135/o=glue t=org.nordugrid.ldapglue2"
},
<output omitted>
```

Any DNS client for manual data retrieval

Any DNS client or library can be used to obtain ARCHERY resource record sets. The *rendering* is designed to be simple and raw data is human readable:

```
[user ~]$ host -t TXT _archery.nordugrid.org
_archery.nordugrid.org descriptive text "u=uk.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=ua.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=fi.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=lt.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=si.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=hu.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=ch.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=atlas.archery.ndgf.org t=archery.group"
_archery.nordugrid.org descriptive text "u=pt.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=se.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=no.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=sk.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=ee.archery.nordugrid.org t=archery.group"
_archery.nordugrid.org descriptive text "u=dk.archery.nordugrid.org t=archery.group"

[user ~]$ host -t TXT _archery.dk.archery.nordugrid.org
_archery.dk.archery.nordugrid.org descriptive text "u=dns://e06b294c0d._archery.dk.
```

(continues on next page)

(continued from previous page)

```
↪archery.nordugrid.org. t=archery.service"

[user ~]$ host -t TXT e06b294c0d._archery.dk.archery.nordugrid.org
e06b294c0d._archery.dk.archery.nordugrid.org descriptive text "o=service t=org.
↪nordugrid.arex id=deckard.dcsc.ku.dk"
e06b294c0d._archery.dk.archery.nordugrid.org descriptive text "u=ldap://deckard.dcsc.
↪ku.dk:2135/o=glue t=org.nordugrid.ldapglue2"
e06b294c0d._archery.dk.archery.nordugrid.org descriptive text "u=ldap://deckard.dcsc.
↪ku.dk:2135/Mds-Vo-Name=local,o=grid t=org.nordugrid.ldapng"
e06b294c0d._archery.dk.archery.nordugrid.org descriptive text "u=gsiftp://deckard.
↪dcsc.ku.dk:2811/jobs t=org.nordugrid.gridftpjob"
```

ARC Client/SDK endpoint retrieval plugin

ARCHERY endpoint retrieval plugin for ARC client/SDK is a part of release starting from ARC 5.4.4.

It is installed within core plugins set and cen be used transparently with `-g` option:

```
[manf@arc-client ~]$ arctest -J 2 -g moldyngrid.org -d INFO
INFO: Configuration (/etc/arc/client.conf) loaded
INFO: Configuration (/home/manf/.arc/client.conf) loaded
INFO: Using proxy file: /home/manf/.globus/x509_user_proxy
INFO: Using certificate file: /home/manf/.globus/usercert.pem
INFO: Using key file: /home/manf/.globus/userkey.pem
INFO: Using CA certificate directory: /etc/grid-security/certificates
INFO: Broker Random loaded
INFO: Found service endpoint index.moldyngrid.org (type org.nordugrid.archery)
INFO: Status for service endpoint "dns://d9b05b0008._archery.index.moldyngrid.org."␣
↪is set to inactive in ARCHERY. Skipping.
INFO: Status for service endpoint "dns://ac525b5fea._archery.index.moldyngrid.org."␣
↪is set to inactive in ARCHERY. Skipping.
INFO: Found service endpoint dns://7c99fb635c._archery.index.moldyngrid.org. (type␣
↪archery.service)
INFO: Found service endpoint dns://616f87be8e._archery.index.moldyngrid.org. (type␣
↪archery.service)
INFO: Found service endpoint dns://46e36be5ec._archery.index.moldyngrid.org. (type␣
↪archery.service)
INFO: Found service endpoint ldap://grid.isma.kharkov.ua:2135/Mds-Vo-Name=local,
↪o=grid (type org.nordugrid.ldapng)
INFO: Found service endpoint https://grid.isma.kharkov.ua:60443/arex (type org.ogf.
↪glue.emies.activitymanagement)
<output omitted>
INFO: Found service endpoint gsiftp://arc6.univ.kiev.ua:2811/jobs (type org.nordugrid.
↪gridftpjob)
INFO: Found service endpoint ldap://arc6.univ.kiev.ua:2135/o=glue (type org.nordugrid.
↪ldapglue2)
INFO: Found service endpoint https://arc6.univ.kiev.ua:443/arex (type org.nordugrid.
↪arcrest)
Submitting test-job 2:
&( executable = "/usr/bin/env" )( stdout = "stdout" )( stderr = "stdout" )( gmlog =
↪"gmlog" )( jobname = "arctest2" )( clientxrsl = "&( executable = ""/usr/bin/env""␣
↪)( jobname = ""arctest2"" )( stdout = ""stdout"" )( join = ""yes"" )( gmlog = "
↪"gmlog"" )" )
Client version: nordugrid-arc-6.2.0
Test submitted with jobid: gsiftp://arc6.univ.kiev.ua:2811/jobs/
↪RHZKDmPeedvnjwO5upha6lOqABFKDmABFKDmIeLKDmABFKDmItfmym
```

NorduGrid monitor is fetching ARCHERY data

NorduGrid infrastructure monitoring web application (aka monitor) is able to visualize realtime ARC CE information obtained using ARCHERY service endpoint discovery.

If you have own monitor setup, you can easiely *configure* ARCHERY as an information source.

Community instance of ARCHERY capable to hold execution environment definitions (known as *RunTime Environments*). This turns ARCHERY into all-in-one community registry that cover both *resources available to community* and *execution software envitonments* verified and used by community.

Following document covers the ARCHERY usage as RunTime Environments registry:

4.3.5 ARCHERY as a community-defined RTEs registry

New in version 6.5.

Community-defined RTEs is a new concept that aimed to automate software environment provisioning for distributed computing infrastructures.

Community-defined RTEs are prepared and digitally singed by community. All RTEs are than indexed in the registry to be discoverable. The ARC CE admin can deploy such RTEs with a signle *ARC Control Tool* command after establishing a trust chain with community.

This document describes how to use ARCHERY as a Community-defined software environment registry.

1. Signing keys

The trust-chain between community and site-admin is based on the digital signatures. All Community-defined RTEs are supposed to be signed using OpenPGP standard for signatures. Technical implementation relies on the GNU Privacy Guard (GPG) software.

If you already have GPG keypair it can be used to sign community RTEs.

If not, generate a new keypair with:

```
[user ~]$ gpg --gen-key
```

2. Preparing RTEs

Community-defined RTE script is any script that follows *RunTime Environments in ARC6* design for both naming scheme and the content.

In many production cases community RTEs require additional files that need to be deployed on ARC CE.

Such files can be listed within RTE script metadata (at the top of file), using the `download` keyword, for example:

```
# download: url:https://example.org/mySW.sif␣
↪checksum:sha256:1c0176d901301be17f86df3330f121d0ae603652eced38e8dbae413f75fad670
```

This will instruct the `arcctl` to download the `mySW.sif` singularity image from `https://example.org/`, verify the file checksum and place it into the software deployment directory for this community.

During the job execution, these downloaded files are available to payload.

Path to the directory containing these files stored in the `RUNTIME_JOB_SWDIR` variable in the job context.

Note: If RTE requires complete software package bundle it is advised to have Singularity container or at least a single tarball to be deployed on the ARC CE. Than write a simple commands for *RTE stage 1* to extract the files or define paths to the container image.

3. Signing and publishing RTEs

The *ARCHERY Manage Tool* completely automates the RTEs signing and publishing process.

Just do it

In case your RTEs are organized in the typical directory-based structure, all you need to do is to add following configuration into the *JSON topology configuration file for ARCHERY*:

```
{
  "software": {
     "rtes_dir": "/home/community/rtesroot"
  }
}
```

This configuration will instruct `archery-manage` to:

- export public key from GPG database (default location) to be published
- index all RTEs available in the specified directory
- extract descriptions from RTE scripts if available (to be published)
- sign all RTE scripts with GPG
- embedds public key, RTEs index and signed RTEs content into the ARCHERY zone

Customize the process

GPG location and key

In case your GPG database is not in the standard location and/or you have several keypairs in the database use the following options to point `archery-manage` to the right place:

```
{
  "software": {
     "gpg_home": "/srv/cummunity-gpg",
     "gpg_keyid": "community@example.org",
     "rtes_dir": "/home/community/rtesroot"
  }
}
```

Manually define public key

In case you are not going to sign RTEs on the same machine you probably have to define the community public key manually.

One way to do it - provide URL that points to the key location:

```
{
  "software": {
     "pubkey_url": "https://example.org/gpg.public.key",
     "rtes_dir": "/home/community/rtesroot"
  }
}
```

Or you can put base64-encoded key value[1] directly into the configuration:

[1] Use `gpg --export | base64 -w0` to get value

```
{
  "software": {
     "pubkey":
↪"mQENBF4y4ssBCADN9317J1KiEZSkDX2T00sCjtRLcL46XcN2llxHPFJm43ziG2GVqFHdoLQEhoN4ozvfNKxndTFkNTh6aP2Cd
↪hLD5/M12Iq4ayYdBBwLJFdZDg/PSclX3kRFS4CDDvVtlXfQ1Jc/XLaVOk7nFDdbTbFEvfenMgcen/
↪cxmlj1GI7IplNoiFoBjLP72NwbPW/
↪WdxGkPzHO+9ypW1WSYHNTL9442zXypcOYyfRhxqae5D+pSIEmptu7279Hr9u7OlhYBRQW4uGI7CTiuqWxBYUKVK1s0UyjB79EJ
↪VBE5jYENAA7uRe3hxeBqvlm0wmohBQ2QTDSPgwuZqp7M5kiK/
↪FPf8q0QtyZGoYlCGUQc0dsBXyRjnHOsUiTKl0M9/
↪PDc+u+Y2teR2XibdQ8ja8CedU09NT5Lbhx+MSv2azLSxLjB7zAVEzPCfxgcXX3G5zfNFsO4+5FoR1GsoGirEN9dkZAxGa71hMh
↪RBDV85aiE4RLeCZ+LQL5IND6ftezQuw3GfRzy8w5EycGCpfWoAr+D5WBhibkBDQReMuLLAQgAr9onAVIzORKjbCapUGVCiTsNp
↪",
     "rtes_dir": "/home/community/rtesroot"
  }
}
```

Host signed RTEs instead of embedding into DNS

The RTE scripts are typically small enoygh to be emedded directly into the ARCHERY DNS zone. But if you want to publish it on the web instead, just use `signed_rtes_url` option.

```
{
  "software": {
     "signed_rtes_url": "https://example.org/rtes/",
     "rtes_dir": "/home/community/rtesroot"
  }
}
```

In this case ARCHERY will store only the references to signed RTE scripts in `https://example.org/rtes/` location.

Signed RTEs srcipts will be saved locally in the `signed` directory[2] and NEEDS TO BE manually uploaded to the web-hosting to be actually accessible in that location.

Manually define RTEs

Instead of organizing RTEs in the directory-like structure, they can be explicitely described in the configuration one-by-one. This allows more flexibility, as you can point to already existing and signed RTEs published anywhere.

To manully describe community RTEs define `rtes` array in the configuration:

```
{
  "software": {
    "rtes": [
      {
        "name": "APPS/COMMUNITY/SW1-1.0.0",
        "description": "Community SW1",
        "url": "https://example.org/rtes/SW1-1.0.0.signed",
      },
      {
        "name": "APPS/COMMUNITY/SW2-1.7.0",
        "description": "Community SW2",
        "data":
↪"owGbwMvMwMHYf+1u+Kqw5aWMaxmPJ3EEG+ka6xnoGcRZrbinrJCSWpxclFlQkpmfZ6Xgm5/
```

(continues on next page)

[2] can be also redefined with additional `signed_rtes_dir` option

(continued from previous page)

```
↪jUpnnXpSZohBspBCcX15SnliUqqCRklmkyQVUm1+el5OfmKJQWpRjlVFSUlBspa+fm5+TUpmXDtSjl1+Url9UklqsX2ykV5yZp
↪FwA=",
        },
        {
          "name": "APPS/COMMUNITY/SW3-0.2.1",
          "path": "/home/community/rtes/APPS/COMMUNITY/SW3-0.2.1"
        }
      ]
    }
}
```

Array contains objects describing RTEs. Each object defines the name, optionally description and points to the content of community-defined RTE script, using one of the following options:

- url - provide URL to signed RTE script
- data - provide base64-encoded signed RTE script content
- path - local path to plain (unsigned) RTE script location

To sign RTE script without archery-manage, you can use following command:

```
[console ~]# gpg --output rtescript.signed --sign rtescript.sh
```

It is possible to use both rtes array and rtes_dir simultaneously.

Note: If you are looking for more details behind the ARCHERY idea - read this paper.

4.4 ARC Admin Tools Reference

This section holds an online version of the command line options reference for ARC admin tools.

4.4.1 ARC Control Tool

4.4.2 ARCHERY Manage Tool

4.4.3 ARC Configuration Parser

CHAPTER

FIVE

TECHNICAL DOCUMENTS DESCRIBING ARC COMPONENTS

Following documents gives a deep technical description of the various ARC components. If you are looking for architecture internals (how parts of ARC was designed) you can follows this section.

5.1 ARC Data Services Technical Description

5.1.1 A-REX Data Cache technical description

Structure of the cache directory

Cached files are stored in sub-directories under the `data` directory in each main cache directory. Filenames are constructed from an SHA-1 hash of the URL of the file and split into subdirectories based on the two initial characters of the hash. In the extremely unlikely event of a collision between two URLs having the same SHA-1 hash, caching will not be used for the second file.

When multiple caches are used, a new cache file goes to a randomly selected cache, where each cache is weighted according to the size of the file system on which it is located.

For example: if there are two caches of 1TB and 9TB then on average 10% of input files will go to the first cache and 90% will go to the second cache.

Some associated metadata including the corresponding URL and an expiry time, if available, are stored in a file with the same name as the cache file, with a `.meta` suffix.

For example, with a cache directory `/cache` the file `srm://srm.nordugrid.org/grid/atlas/file1`:

- is mapped to `/cache/data/37/b19accc950c37876a61d2de6e238d38c9e94c0`,
- the file `/cache/data/37/b19accc950c37876a61d2de6e238d38c9e94c0.meta` contains the original URL and an expiry time if one is available.

At the start of a file download, the cache file is locked, so that it cannot be deleted and so that another download process cannot write the same file simultaneously. This is done by creating a file with the same name as the cache filename but with a `.lock` suffix. This file contains the process ID of the process and the hostname of the host holding the lock. If this file is present, another process cannot do anything with the cache file and must wait until the cache file is unlocked (i.e. the *.lock* file no longer exists). The lock is continually updated during the transfer, and is considered stale if 15 minutes have passed since the last update. These stale locks, caused for example by a download process exiting abnormally, will therefore automatically be cleaned up. Also, if the process corresponding to the process ID stored inside the lock is no longer running on the host specified in the lock, it is safe to assume that the lock file can be deleted. If a file is requested which already exists in the cache (and is not locked), the cache file is not locked, but checks are done at the end of cache processing to ensure the file was not modified during the processing.

How the cache works

If a job requests an input file which can be cached or is allowed to be cached, it is stored in the selected cache directory, then a hard link is created in a per-job directory, under the `joblinks` subdirectory of the main cache directory. Then depending on the configuration, either the hard-link is copied or soft-linked to the SD. The former option is advised if the cache is on a file system which will suffer poor performance from a large number of jobs reading files on it, or the file system containing the cache is not accessible from worker nodes. The latter option is the default option. Files marked as executable in the job will be stored in the cache without executable permissions, but they will be copied to the SD and the appropriate permissions applied to the copy.

The per-job directory is only readable by the local user running the job, and the cache directory is readable only by the A-REX user. This means that the local user cannot access any other users' cache files. It also means that cache files can be removed without needing to know whether they are in use by a currently running job. However, as deleting a file which has hard links does not free space on the disk, cache files are not deleted until all per-job hard links are deleted.

Warning: If a cache is mounted from an NFS server and the A-REX is run by the root user, the server must have the `no_root_squash` option set for the A-REX host in the `/etc/exports` file, otherwise the A-REX will not be able to create the required directories.

Note: Note that when running A-REX under a non-privileged user account, all cache files will be owned and accessible by the same user, and therefore modifiable by running jobs. This is potentially dangerous and so caching should be used with caution in this case.

If the file system containing the cache is full and it is impossible to free any space, the download fails and is retried without using caching.

Before giving access to a file already in the cache, the A-REX contacts the initial file source to check if the user has read permission on the file. In order to prevent repeated checks on source files, this authentication information is cached for a limited time. On passing the check for a cached file, the user's DN is stored in the `.meta` file, with an expiry time equivalent to the lifetime remaining for the user's proxy certificate. This means that the permission check is not performed for this user for this file until this time is up (usually several hours). File creation and validity times from the original source are also checked to make sure the cached file is fresh enough. If the modification time of the source is later than that of the cached file, the file will be downloaded again. The file will also be downloaded again if the modification date of the source is not available, as it is assumed the cache file is out of date. These checks are not performed if the DN is cached and is still valid.

The A-REX checks the cache periodically if it is configured to do automatic cleaning. If the used space on the file system containing the cache exceeds the high water-mark given in the configuration file it tries to remove the least-recently accessed files to reduce size to the low water-mark.

Cache cleaning

When *[arex/cache/cleaner] block* is defined the cache is cleaned automatically periodically (every 5 minutes) by the A-REX to keep the size of each cache within the configured limits. Files are removed from the cache if the total size of the cache is greater than the configured limit. Files which are not locked are removed in order of access time, starting with the earliest, until the size is lower than the configured lower limit. If the lower limit cannot be reached (because too many files are locked, or other files outside the cache are taking up space on the file system), the cleaning will stop before the lower limit is reached.

Since the limits on cache size are given as a percentage of space used on the filesystem on which the cache is located, it is recommended that each cache has its own dedicated file system.

If the cache shares space with other data on a file system, the option *calculatesize=cachedir* should be set in `arc.conf` so that the cache limits are applied on the size of the cache rather than the file system.

With large caches mounted over NFS and an A-REX heavily loaded with data transfer processes, cache cleaning can become slow, leading to caches filling up beyond their configured limits. For performance reasons it may be advantageous to disable cache cleaning by the A-REX, and run the *cache-clean* tool (usually /usr/libexec/arc/cache-clean) independently on the machine hosting the file system.

Caches can be added to and removed from the configuration as required without affecting any cached data, but after changing the configuration file, the A-REX should be restarted. If a cache is to be removed and all data erased, it is recommended that the cache be put in a *draining* state until all currently running jobs possibly accessing files in this cache have finished. In this state the cache will not be used by any new jobs, but the hard links in the *joblinks* directory will be cleaned up as each job finishes. Once this directory is empty it is safe to delete the entire cache.

Caches may also be marked as *read-only*, so that data cached there can be used by new jobs, but no new data will be written there. This feature is available from ARC version 6.7. Note that read-only caches are not cleaned by A-REX.

Exposing the Cache

Normally the ARC cache is internal to the CE and is not exposed to the outside. However it may be beneficial to allow reading cache files, if for example the file is lost from Grid storage or as a fallback when Grid storage is down. This can be done via HTTPS through the A-REX web services interface.

Specifying *[arex/ws/cache] block* opens remote read access to certain cache files for certain credential properties. When configured this allows cached files to be read from the A-REX WS endpoint, for example if file `gsiftp://my.host/file1` is cached at CE `a-rex.host` the file is accessible (if credentials allow) at:

```
https://a-rex.host/arex/cache/gsiftp://my.host/file1
```

Since remote reading can increase the load on A-REX, the number of concurrent requests should be limited. This can be done using the *max_data_transfer_requests* configuration option.

Indexing the Cache content

The *ARC Cache Index (ACIX)* provides a way to discover locations of cached files.

5.1.2 A-REX data transfer framework (DTR) technical description

This page describes the data staging framework for ARC, code-named DTR (Data Transfer Reloaded).

Overview

ARC's Computing Element (A-REX) performs the task of data transfer for jobs before and after the jobs run. The requirements and the design steps for the data staging framework are described in *DTR Design and Implementation Details*. The framework is called DTR (Data Transfer Reloaded) and uses a three-layer architecture, shown in the figure below:

The Generator uses user input of tasks to construct a Data Transfer Request (also DTR) per file that needs to be transferred. These DTRs are sent to the Scheduler for processing. The Scheduler sends DTRs to the Pre-processor for anything that needs to be done up until the physical transfer takes place (e.g. cache check, resolve replicas) and then to Delivery for the transfer itself. Once the transfer has finished the Post-processor handles any post-transfer operations (e.g. register replicas, release requests). The number of slots available for each component is limited, so the Scheduler controls queues and decides when to allocate slots to specific DTRs, based on the prioritisation algorithm implemented. See *DTR priority and shares system* for more information.

This layered architecture allows any implementation of a particular component to be easily substituted for another, for example a GUI with which users can enter DTRs (Generator) or an external point-to-point file transfer service (Delivery).

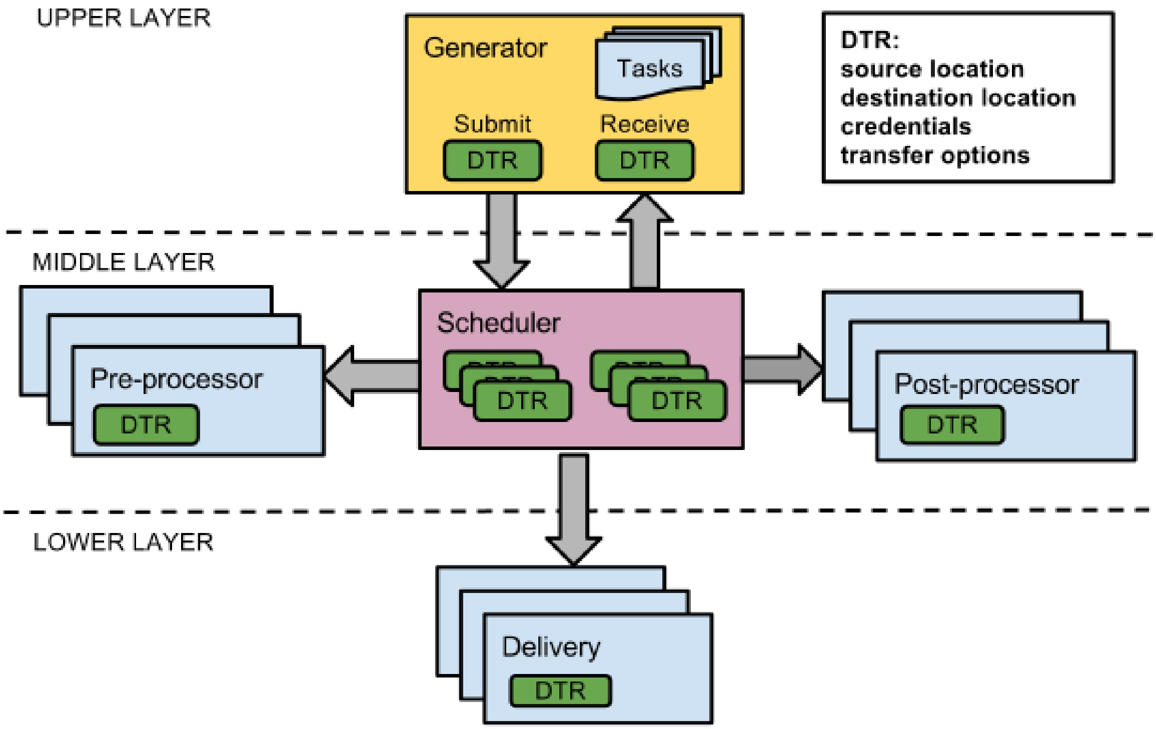


Fig. 5.1: DTR three-layer architecture

Implementation

The middle and lower layers of the architecture (Scheduler, Processor and Delivery) are implemented as a separate library **libarcdatastaging** (in `src/libs/data-staging` in the ARC source tree). This library is included in the **nordugrid-arc** common libraries package. It depends on some other common ARC libraries and the DMC modules (which enable various data access protocols and are included in `nordugrid-arc-plugins-*` packages) but is independent of other components such as A-REX or ARC clients. A simple Generator is included in this library for testing purposes. A Generator for A-REX is implemented in `src/services/a-rex/grid-manager/jobs/DTRGenerator.(h|cpp)`, which turns job descriptions into data transfer requests.

Configuration

Data staging is configured through the `[arex/data-staging]` block in arc.conf. Reasonable default values exist for all parameters but the *[arex/data-staging] block* can be used to tune the parameters, and also enable *multi-host data staging*. A selection of parameters are shown below:

Parameter	Explanation	Default Value
maxdelivery	Maximum delivery slots	10
maxprocessor	Maximum processor slots per state	10
maxemergency	Maximum emergency slots for delivery and processor	1
maxprepared	Maximum prepared files (for example pinned files using SRM)	200
sharetype	Transfer share scheme (dn, voms:vo, voms:group or voms:role)	None
definedshare	Defined share and priority	_default 50
Multi-host related parameters		
deliveryservice	URL of remote host which can perform data delivery	None
localdelivery	Whether local delivery should also be done	no
remotesizelimit	File size limit (in bytes) below which local transfer is always used	0
usehostcert	Whether the host certificate should be used in communication with remote delivery services instead of the user's proxy	no

Description of other data-staging paramters can be found in *[arex/data-staging] block*. The multi-host parameters are explained in more detail in *ARC Data Delivery Service Technical Description*

Example:

```
[data-staging]
maxdelivery = 10
maxprocessor = 20
maxemergency = 2
maxprepared = 50
sharetype = voms:role
definedshare = myvo:production 80
deliveryservice = https://spare.host:60003/datadeliveryservice
localdelivery yes
remotesizelimit = 1000000
```

Client-side priorities

To specify the priority of jobs on the client side, the `priority` element can be added to an XRSL job description, eg:

```
("priority" = "80")
```

For a full explanation of how priorities work see *DTR priority and shares system*.

gm-jobs -s

The command "gm-jobs -s" to show transfer shares information now shows the same information at the per-file level rather than per-job. The number in "Preparing" are the number of DTRs in TRANSFERRING state, i.e. doing physical transfer. Other DTR states count towards the "Pending" files. For example:

```
Preparing/Pending files        Transfer share
        2/86                   atlas:null-download
        3/32                   atlas:production-download
```

As before, per-job logging information is in the controldir/job.id.errors files, but A-REX can also be configured to log all DTR messages to a central log file in addition through the `logfile` parameter.

Using DTR in third-party applications

ARC SDK Documentation gives examples on how to integrate DTR in third-party applications.

Supported Protocols

The following access and transfer protocols are supported. Note that third-party transfer is not supported.

- file
- HTTP(s/g)
- GridFTP
- SRM
- Xrootd
- LDAP
- Rucio
- S3
- ACIX
- RFIO/DCAP/LFC (through GFAL2 plugins)

Multi-host Data Staging

To increase overall bandwidth, multiple hosts can be used to perform the physical transfers. See *ARC Data Delivery Service Technical Description* for details.

Monitoring

In A-REX the state, priority and share of all DTRs is logged to the file controldir/dtr.state periodically (every second). This can then used by the Gangliarc framework to show data staging information as ganglia metrics.

Advantages

DTR offers many advantages over the previous system, including:

High performance
When a transfer finishes in Delivery, there is always another prepared and ready, so the network is always fully used. A file stuck in a pre-processing step does not block others preparing or affect any physical transfers running or queued. Cached files are processed instantly rather than waiting behind those needing transferred. Bulk calls are implemented for some operations of indexing catalogs and SRM protocols.

Fast
All state is held in memory, which enables extremely fast queue processing. The system knows which files are writing to cache and so does not need to constantly poll the file system for lock files.

Clean
When a DTR is cancelled mid-transfer, the destination file is deleted and all resources such as SRM pins and cache locks are cleaned up before returning the DTR to the Generator. On A-REX shutdown all DTRs can be cleanly cancelled in this way.

Fault tolerance
The state of the system is frequently dumped to a file, so in the event of crash or power cut, this file can be read to recover the state of ongoing transfers. Transfers stopped mid-way are automatically restarted after cleaning up the half-finished attempt.

Intelligence
Error handling has vastly improved so that temporary errors caused by network glitches, timeouts, busy remote services etc are retried transparently.

Prioritisation
Both the server admins and users have control over which data transfers have which priority.

Monitoring
Admins can see at a glance the state of the system and using a standard framework like Ganglia means admins can monitor ARC in the same way as the rest of their system.

Scaleable
An arbitrary number of extra hosts can be easily added to the system to scale up the bandwidth available. The system has been tested with up to tens of thousands of concurrent DTRs.

Configurable
The system can run with no configuration changes, or many detailed options can be tweaked.

Generic flexible framework
The framework is not specific to ARC's Computing Element (A-REX) and can be used by any generic data transfer application.

Open Issues

- Provide a way for the infosys to obtain DTR status information
  - First basic implementation: when DTR changes state write current state to .input or .output file
- Decide whether or not to cancel all DTRs in a job when one fails
  - Current logic: if downloading, cancel all DTRs in job, if uploading don't cancel any
  - Should be configurable by user - also EMI execution service interface allows specifying per-file what to do in case of error
- Priorities: more sophisticated algorithms for handling priorities
- Advanced features such as pausing and resuming transfers

Related Documents

DTR Design and Implementation Details

This page documents the requirements and design stages of the new data staging framework which took place around mid-2010.

Issues with previous implementation

1. Queueing happens per job - that makes it impossible to use potentially more effective processing order
2. Slow data transfers block those which could be done faster.
3. Big data transfers block small ones.
4. Jobs waiting for already cached (or to be cached) files are blocked by other jobs in queue.
5. Special features of sophisticated protocols are not taken into account - like SRM's "try later".
6. No priorities aka flexible queues.
7. No support for different credentials for different files.
8. No bandwidth handling.
9. No handling of files with different access latency (eg tape vs disk)
10. No mechanism to choose a preferred replica for the LFC (catalog) inputs, for example: if replicas are in ndgf, swegrid, signet, unige, try with ndgf first.

Task Summary

The initial task was NOT to solve all these issues. The task was to create a framework which could be extended to solve them later, ot to find/adopt such a framework.

Requirements

1. Effective usage of bandwidth. Whenever any transfer is paused due to any reason (tape stage in, retry later) for estimated time another transfer should use available bandwidth.
2. Transfer negotiation (with protocols such as SRM) should be independent of physical data transfer.
3. Each transfer should be capable to use own credentials.
4. Transfer should be capable of pausing (temporary cancel) and resuming (if protocol allows).
5. Automatic/dynamic redistribution of bandwidth is needed to allow short transfer to pass through even while big transfers are taking whole bandwidth.
6. Transfer from multiple alternative locations.
7. Cache checks should happen independently of data transfer to avoid locks.
8. Jobs where all files are cached should be processed immediately.
9. Better description of file source/destination than just URL (options are difficult to handle, something nicer is needed)
10. Priorities at different levels: among user groups, inside groups. Any other levels? 3 possible levels: among VO, users/roles inside VO, inside user identity.
11. Ability for users to set relative priority of their own jobs, both before submission and while job is in queue.

Security Requirements

- It must be built into the design that no process has higher privileges than necessary
- Elevated privileges are required for:
  - Access to cache. Cache files are only read/writeable by root user so they cannot be modified by jobs
  - Access to session and control directories. Access to the these directories should be performed under the uid of the job owner. The current method of running as root and chown'ing must not be used.
- Elevated privileges are not required for any other parts of the system such as scheduling

Performance Requirements

- Must scale higher than current highest workloads
- Must be able to handle up to 10000 active jobs (between ACCEPTED and FINISHED)
- Must be able to handle up to 1000 active physical transfers whilst ensuring all available bandwidth is used
- Must be able to handle transfers which have to wait for several hours before the physical file is ready to transfer

Possible solution to URL options problem

There has been a long standing problem with the format and syntax used to express URLs and associated metadata and options. While not directly related to data staging, it will be addressed as part of the data staging work.

Architecture Proposal

- **3 layers:**
  1. Higher layer processing data staging requirements of jobs, collecting requested attributes/properties, resolving priorities (flattening them), managing credentials.
  2. Middle layer schedules individual transfers, suspends and resumes them, distributes bandwidth, etc.
  3. Lower level handles individual transfer protocol, communicates with middle layer to acquire, release and pre-allocate resources (mostly bandwidth), caches connections (if possible).
- Any layer can be outsourced to external service, for example gLite FTS
- **Basic file staging stages:**
  1. Identify transfer options
  2. Check cache
  3. Evaluate authorization (may require bandwidth)
  4. Resolve location (meta-URL, if needed, may require bandwidth)
  5. If needed repeat all steps from beginning
  6. Do transfer
  7. Post-processing (eg register replicas, release cache locks)

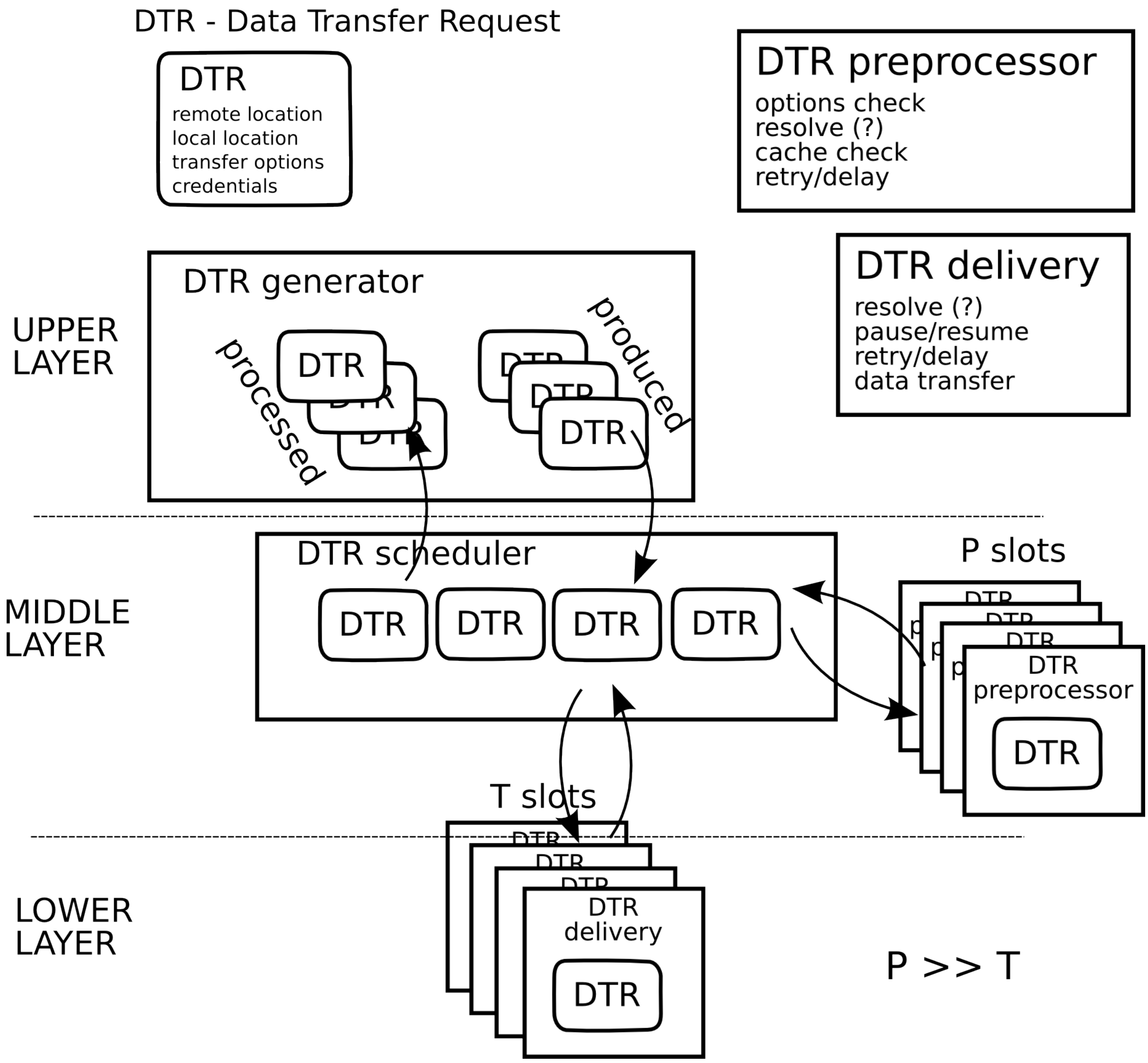


Fig. 5.2: Functional components and layers.

Requirements for components interfaces (based on protocol descriptions and architecture)

DTR Description

DTR stands for Data Transfer Request. This is the structure that contains several fields that fully describe the file transfer to be performed. One DTR is generated by the generator per each file transfer.

A detailed description and state diagrams of DTRs can be found in the *Detailed description of DTRs*

DTR Generator to DTR Scheduler

- Push DTR from Generator to Scheduler:
  - DTR contains description of single entity to be transferred - uploaded or downloaded. That includes endpoint, transfer parameters and limits, credentials - possibly multiple, etc.
  - Multiple DTRs may be affiliated together. Possible reasons and uses:
    - Belong to same job
    - Belong to bunch of jobs which user indicated as preferably processed together
    - Belong to same VO and assigned priorities to be applied within group
    - Failure of one DTR in group may cancel processing of other DTRs (not sure, may be implemented in Generator)
  - DTR may have assigned priorities levels. Probably related to groups.
- Receive DTR from Scheduler to Generator:
  - Returned DTR indicates outcome of processing, either positive or negative. In last case it includes description of encountered problems and level of severity.
- Cancel DTR in Scheduler
- Modify DTR properties in scheduler. Possible usage: - Manipulate priorities

DTR Scheduler to DTR Preprocessor

- Push DTR from Scheduler to Preprocessor
  - Because DTR preprocessing is supposed to take short time it may include processing timeout
- Receive DTR from Preprocessor to Scheduler
  - Returned DTR indicates outcome of processing
    - Positive
      - DTR comes with information need for further processing either in Preprocessor or Delivery unit
      - DTR may contain multiple/alternative additional endpoints
      - Probably such DTR may be presented in tree-like diagram
    - Failure - includes description of encountered problems and level of severity
    - Delayed processing
      - Includes retry time and possible margins.
      - Scheduler must ensure this DTR will go back to Preprocessor within specified time margins.
- Cancel DTR in Preprocessor

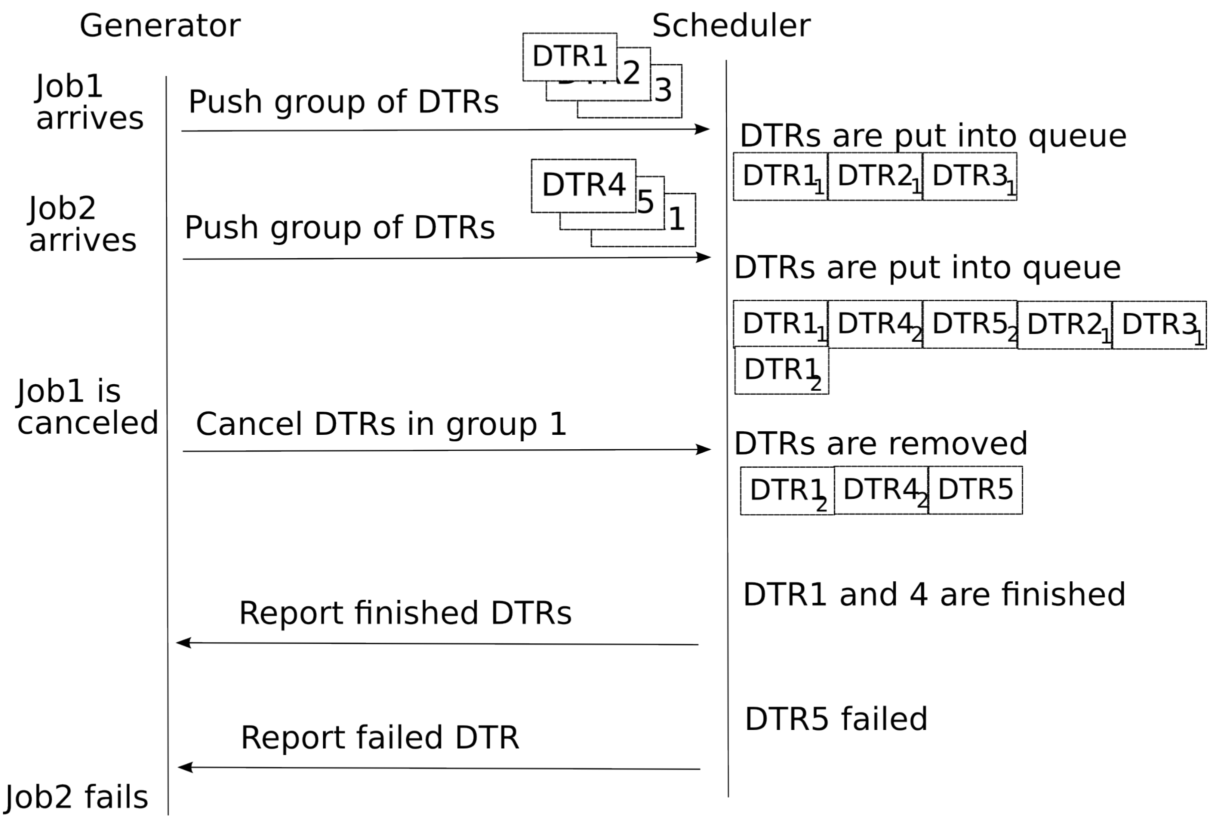


Fig. 5.3: Example of communication session between DTR Generator and DTR Scheduler.

DTR scheduler to DTR delivery

- Push DTR from Scheduler to Delivery
  - DTR may have bandwidth assigned
  - DTR may have timeout related parameters assigned - minimal transfer rate, maximal inactivity timeout, etc.
- Cancel DTR in Delivery
- Suspend DTR in Delivery - should DTR leave Delivery or should it stay there?
- Receive DTR from Delivery to Scheduler. Returned DTR indicates outcome of processing
  - Positive
  - Partially positive (partial data delivered)
  - Redirection
  - Failure - includes description of encountered problems and level of severity
- Get information about bandwidth currently used by Delivery
- Modify assigned bandwidth
  - May be used to free some bandwidth for urgent transfers

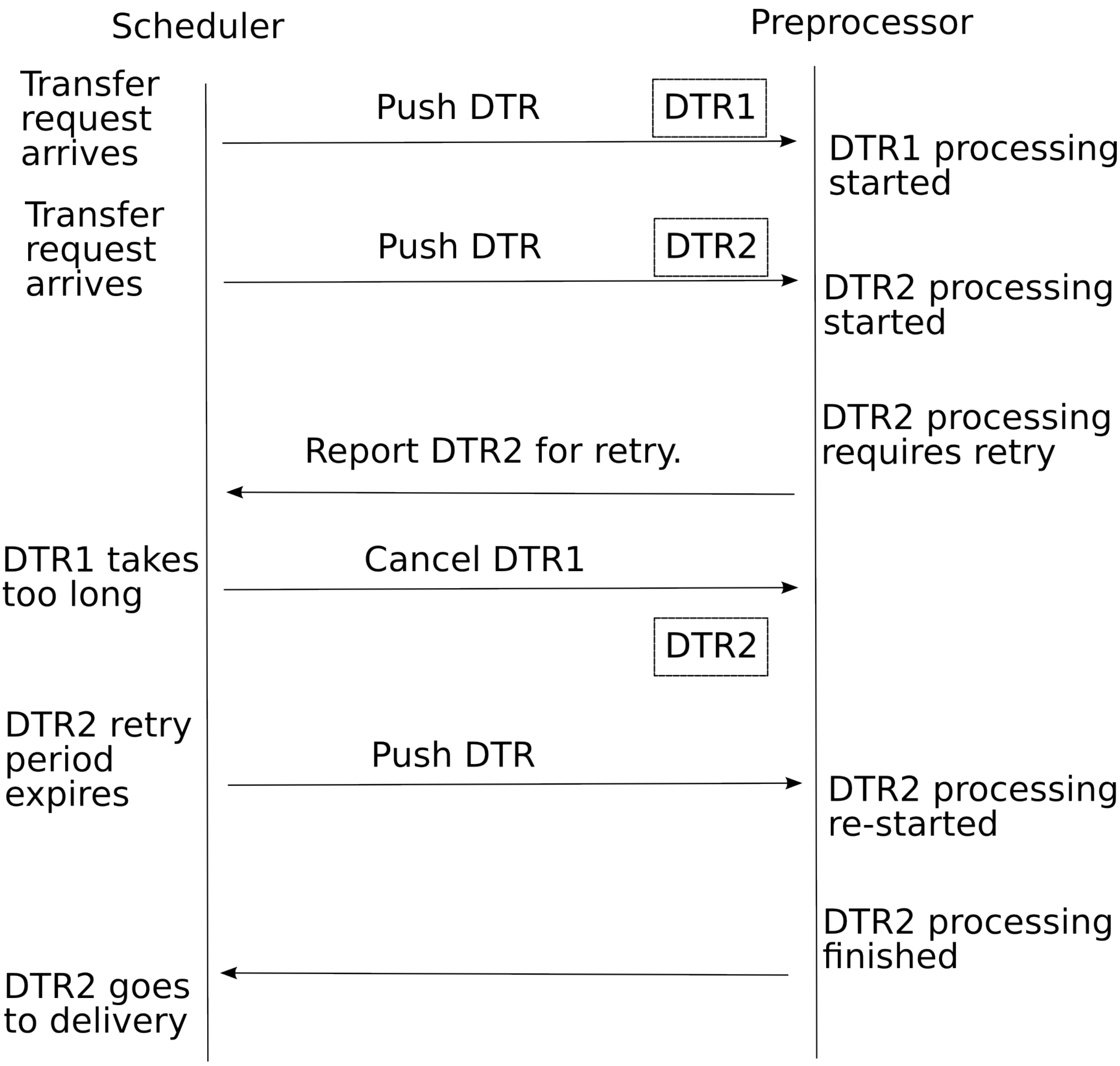


Fig. 5.4: Example of communication session between DTR Scheduler and DTR preprocessor.

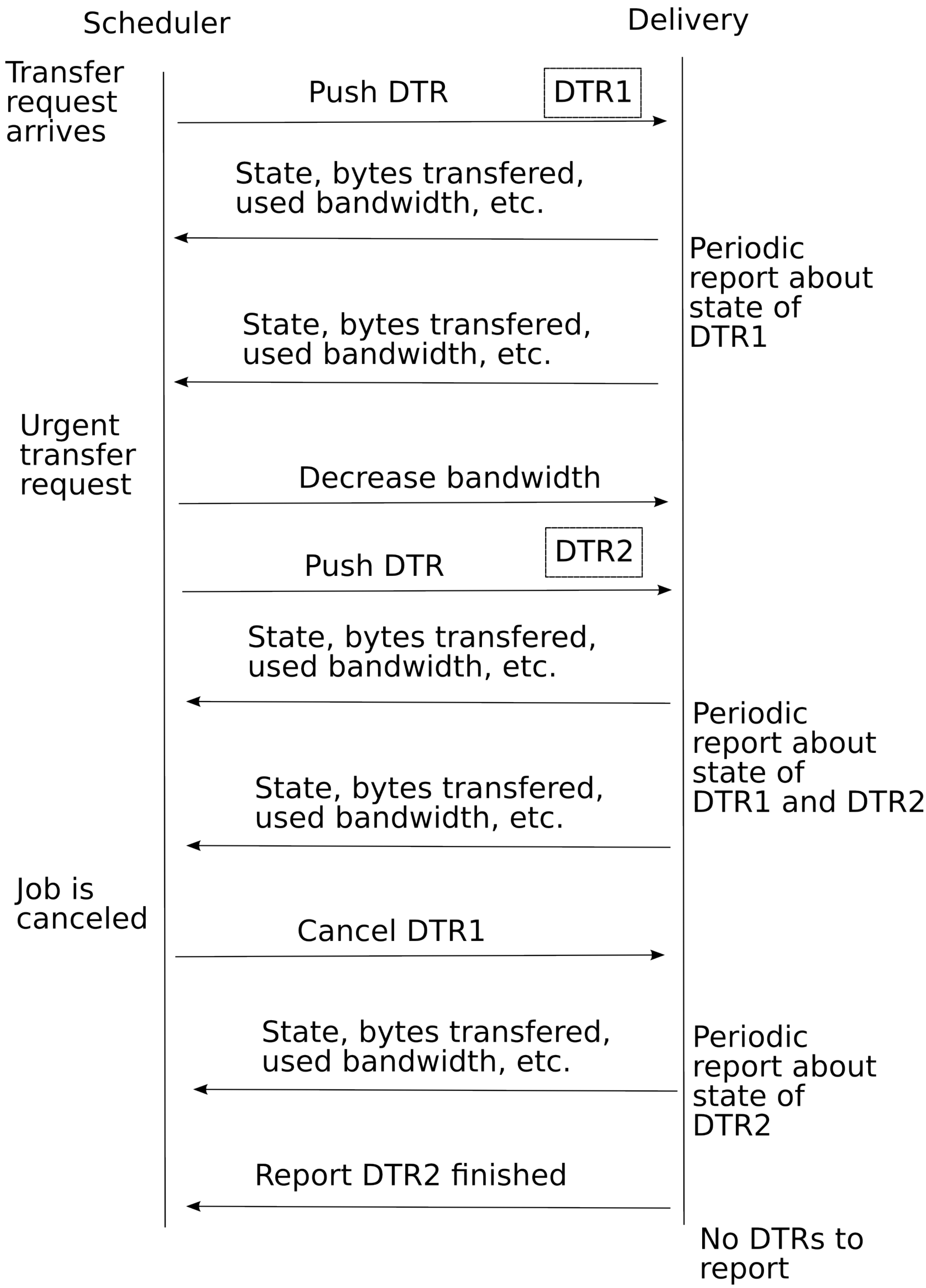


Fig. 5.5: Example of communiction session between DTR Scheduler and DTR Delivery.

Component Workflows

Generator

The Generator is an integral part of the a-rex process. Internally it performs 4 tasks in following order:

- Makes DTRs out of existing job descriptions also assigning priorities and grouping information and makes them available to the Scheduler.
- Communicates immediate requests like DTR cancel or suspend to the Scheduler.
- Monitors DTR states as reported by the Scheduler (and possibly by other modules) in order to provide feedback to client tools of A-REX asking for job state.
- Receives finished/failed DTRs from the Scheduler and initiates job processing continuation. Note: This part may be merged with previous one.

Scheduler

"Queues" are queues of DTRs waiting to enter pre/post-processing or delivery. They are kept internal to the scheduler. The scheduler is the only place with complete knowledge of the system and the only place where priorities are controlled and error conditions are handled. When an event is received and involves sending the DTR to a queue, the DTR is put at a certain position in the queue. Depending on the DTR priority, other DTRs in the queue may be moved, paused or stopped to allow higher priority DTRs to proceed quicker or consume more resources. On receiving an event, the relevant queue is examined and action taken (eg if a delivery finished start a new one).

Reactions to new events

For simplicity error conditions are not included in the workflow here but described separately on the *Detailed description of DTRs*. They are examined by the scheduler, which will decide the workflow - do any necessary post-processing, decide whether to retry (immediately or after some delay) or to report back to the generator that the DTR definitively failed. DTR state transitions are also described in more detail and with diagrams on the *Detailed description of DTRs*.

- New DTR from generator
  - if cacheable or meta-protocol:
    - add to pre-processor queue (for cache check, replica resolution etc)
  - else if ready to be delivered (base protocol for src and dest):
    - add to delivery queue
- DTR returned from pre-processor
  - if cached:
    - send to post-processor to be linked
  - else if need more pre-processing:
    - add to pre-processing queue
  - else:
    - add to delivery queue
- DTR returned from delivery
  - if post-processing required (index registration, cache linking, release request):
    - put in post-processor queue

  - else:
    * return to generator
- DTR returned from post-processor:
  - Return to generator
- DTR cancel notification from generator
  - if before pre-processing:
    * return to generator
  - else if in post-processing:
    * wait until finished and then send back to clean up
  - else:
    * cancel immediately and add to post-processing queue for clean up
- DTR modify notification from generator
  - change request immediately, and modify queue if appropriate

Processor

The processor is divided into two logical parts: the pre-processor and the post-processor. Either part is invoked by the scheduler as a process/thread and has the DTR to process. Therefore, the pre- or post-processor can be a straightforward function, performing the next steps:

The pre-processor:

- check endpoint for its presence in cache
  - if successful, mark DTR as BYPASS_TRANSFER, return the DTR to the scheduler
  - if file is not in cache, construct cache endpoint for DTR destination and return to scheduler
- resolve the replicas of the file, if needed
  - return the DTR with a failure if no locations have been found
  - return the list of replicas found to scheduler
  - Note: The pre-processor doesn't care if the resolved locations represent meta-protocols themselves, it's the scheduler's job to determine it and possibly send this DTR for the pre-processing once again.
- query an endpoint
  - supply information on size, checksum, creation date, access latency as available according to protocol
- If an asynchronous request needs to be performed, for example SRM prepareToGet
  - Start request, mark DTR as STAGING_PREPARING_WAIT, return to the scheduler. An estimated wait time may be set by the pre-processor from information supplied from the remote service.
- If a polling request arrives from the scheduler
  - Check state of asynchronous request and report back to scheduler success, wait more, or error.
- If the cancellation request arrives from the scheduler
  - interrupt the operation, mark the DTR as PREPROCESSING_CANCELLED, return to the scheduler
- If the preprocessor hits the timeout during performing these tasks
  - interrupt the operation, mark the DTR as PREPROCESSING_NOT_FINISHED, return to the scheduler.

Possible features of the pre-processor in the future

- for each of resolved locations:
  - check options one by one (we still have to define transfer options, below is the example), mark in the list to which extent it satisfies the option
    - can the location provide a required bandwidth
    - other options, specified for the user
    - keep processing all the options even if some of them are not satisfied – the scheduler may later review the options and start the processing again
  - request the file size (if possible) and compute estimated transfer time, mark it in the list for this location
- if there are in the list locations that satisfied all the checks
  - return the DTR as TRANSFER_READY to the scheduler with the list of these locations and their estimated transfer times
- if there are no locations in the list that satisfied all the checks
  - return the DTR with the least severe encountered failure (from most severe to least severe, Location is dead - Location is not authorized - Location doesn't satisfy the options), so the scheduler can either drop the DTR (in case of dead/non-authorized storages) or review transfer options and try again (in case of unsatisfied options)

The Post-processor:

- release stage requests;
- register a replica in an index service;
- release cache locks;

Delivery

Delivery is a component where the transfer orders are handled. Delivery process is listening to the Scheduler to receive the ready DTRs for transferring or any other events such as decreasing the bandwidth or suspending and canceling the ongoing jobs. Delivery process, reports the status of the DTRs and events to the scheduler periodically.

- Transfer request delivery
  - Pick up a DTR from delivery queue.
  - Check the source and destination, set the bandwidth, timeout and etc.
  - Start transferring by placing the received data in a buffer.
  - Periodically report the status of DTR such as transfered bytes, used bandwidth to Scheduler.
  - By finishing the data transfer do a checksum
    - if checksum is correct return a SUCCESS status to the Scheduler
    - else retry transfer again
- High priority transfer request event
  - Change the DTR to use max bandwidth n
- Job suspension
  - Suspend a transfer request
  - Keep the transfered data information in the delivery process
  - Release the used bandwidth
  - Report the status of suspended DTR to Scheduler
- Transferring cancel

  - Stop transferring
  - Release the resources if any is in use
  - Clean up the buffer
- Status report to Scheduler
  - Periodically collect the information of the DTRs in progress.
  - Calculate used bandwidth, transfered data bytes, status
  - Report the status to the Scheduler

Protocol Interfaces

Current Interface

The following interface is defined for each protocol through the DataPoint class:

- StartReading - Start reading data from the source
- StopReading - Stop reading data from the source
- StartWriting - Start writing data to destination
- StopWriting - Stop writing data to destination
- Resolve - Find physical replicas from an indexing service
- Check - Verify that the current credentials have access
- ListFiles - Find metadata
- Remove - Delete
- (Pre)Register and (Pre)Unregister - add and delete in indexing service

These do not fulfil all the requirements of the DTR interfaces described above.

New Interface

The main limitation of the current interface is that it does not handle efficiently those protocols such as SRM which involve asynchronous preparation steps. In the new framework, having to wait for a file to be prepared should not block other activities. Therefore the proposal is to split protocols as now into meta and direct protocols, but introduce a third class of stageable protocols and appropriate extra methods to handle them. A stageable protocol could also be a meta protocol or a direct protocol. Extra methods are also needed to handle pausing and cancellation of transfers.

For Meta Protocols (eg LFC, RLS)

- Resolve - Resolve replicas in an indexing service
- (Pre)Register and (Pre)Unregister - add and delete in indexing service

For Stageable Protocols (eg SRM, Chelonia)

- PrepareReading - Prepare the storage service for reading. This may involve preparation of Transport URLs (TURLs) which should then be used for physical data transfer, or operations like reading data from tape to disk. If the protocol's implementation is asynchronous then this method may return a status that tells the caller to wait and poll later. The caller should call this method again after some period of time (the remote service may provide an estimate of the preparation time).
- PrepareWriting - Prepare the storage service for writing. Works similarly to PrepareReading.
- FinishReading - Release or abort requests made during PrepareReading, usually called after physical transfer has completed.
- FinishWriting - Release or abort requests made during PrepareWriting, usually called after physical transfer has completed.
- Suspend - Pause current preparation of transfer
- Resume - Resume suspended preparation of transfer (Note: depending on the protocol, suspend and resume may be implemented as stop current request and start new request)
- Cancel - Stop preparation of transfer

For Direct Protocols (eg FTP, HTTP)

- StartReading - Start reading data from the source
- StopReading - Stop reading data from the source
- StartWriting - Start writing data to destination
- StopWriting - Stop writing data to destination
- Modify - Change parameters of transfer such as bandwidth limits
- Suspend - Pause current transfer
- Resume - Resume suspended transfer
- Cancel - Stop current transfer

For All Protocols

- Check - Verify that the current credentials have access
- List - Find metadata
- Remove - Delete

Existing ARC Code

We should aim to re-use as much as possible of the existing ARC code. The arc1 code base will be used for all developments. Code for job description handling and interfacing with the LRMS can remain unchanged, all that concerns us is the code handling the PREPARING and FINISHING states. Job state handling is done in states.cpp. This code can remain largely unchanged but in the ActJobPreparing/Finishing methods the job enters the new system at the upper layer.

The lower level code for data transfer for each protocol is handled in the DMCs. These can largely remain unchanged except for the extra methods in the meta-protocols above.

Caching code can remain unchanged.

PREPARING/FINISHING State Semantics

The semantics of these states may need to be changed - at the moment PREPARING/FINISHING means that the job is transferring data and PENDING those states means a job cannot enter the state due to some limit. In our new system there is less of a distinction between jobs waiting and jobs transferring data, also some files within the job may be transferring while some are waiting. Once the job enters the upper layer of the new system it will be in a staging state even though it may have to wait a long time before any data is transferred.

Processes and Threads

In the current architecture a persistent A-REX thread is spawned by HED. A new data staging process (downloader/uploader) is forked for each job to stage its input or output files. Within each process a thread is created per file. The status of each thread is reported back to the loader process through callbacks and the process exit code tells the A-REX the status of all transfers. There are some problems with this approach:

- The A-REX knows nothing about what happens inside the loader until the process exits and cannot communicate with the loader
- It is not possible to change the uid of each thread individually so they all run under the uid of the downloader/uploader. The uid of the downloader/uploader depends on configuration. If possible uid of mapped user is used. But if there is cache shared among users that uid will be root or uid of A-REX.

To solve this last problem, processes in the delivery layer writing to or from the session dir must run under the locally mapped uid. Writing to the cache must always be done as root and so cache and non-cache downloads must be done in different processes. This leads to the conclusion that the delivery layer must be separate processes from the scheduling layer. Then to solve the first problem there needs to be a method of communication between the delivery and scheduling layers, which must be two way, so that the scheduler can modify on-going transfers and the transfer can report its status back to the scheduler.

Notes on running processes under mapping uid:

- It is only open() operation which needs to be run under special uid. If we follow convention that all open() operations in an executable are called through single wrapper function and put global lock around it, then we can have filesystem access under selected uid inside multi-threaded application. Unfortunately for NFS the process needs to maintain the uid throughout the whole transfer.
- This may introduce performance issue if open() operation takes too long, like in case of remote file system.
- open() may be called by an external library which we have no control over
- according to 'man open' "UID mapping is performed by the server upon read and write requests" and hence suggested approach will fail on NFS.

Operations carried out by the pre- and post-processor require access to the filesystem in the following steps

- Cache preparation - locking the cache file and checking its existence. This must be done as root.
- Cache finalisation - copying or linking the cached file to the session directory. The hard link to the per-job dir must be done as root but the soft linking or copying to the session dir must be done as the mapped user.
  - Last is rather "must" than "may" to make it work on NFS with root_squash=on. The situation may exist where cache is on NFS and then process copying file must switch uid while accessing file and its copy.
- Access to proxy - the proxy is needed when contacting secure remote services. The proxy in the control dir is owned by the mapped user. Therefore either we have to:
  - Make a copy of the proxy owned by root - this does not fit our security requirements above
    - Note1: This is how it is done in current implementation. There is no security hole here because this is same proxy which was obtained by A-REX and written to control directory under root uid. So this proxy already belonged to root and making belong to root again makes little difference.
    - Note2: Because as a rule proxy is stored on local file system it is always accessible by root. Copying of proxy was needed in current implementation due to limitation of Globus libraries - those

were able to accept proxy only from file and were checking if proxy belonged to current uid. Because ARC data library allows assigning credentials to communication directly (at least it should, but may be not implemented for some protocols, needs checking) and because proxy is readable by root such trick is not needed anymore.

  - Use processes rather than threads for the pre- and post-processor, changing the uid of the process to the mapped user
    * This approach is most probablty not needed for this purpose but may be very much desirable for fighting NFS.
  - As suggested above, use global open() function to open with mapped id then switch back to root - this suffers the same problems mentioned above
    * There is no need to do that for proxy because root can always open files belonging to other users

The generator will be a thin layer between the main A-REX process which moves jobs from state to state and the scheduler which processes DTRs - when a job enters the PREPARING or FINISHING state the generator will create DTRs and send them to the scheduler. For the scheduler to be efficient it should run as a separate process (or thread). Each pre-processing step such as resolving replicas in a catalog or a cache permission check should be fast (< few seconds) but should be run asynchronously in order not to block the scheduler if something bad happens.

In summary we have the following processes corresponding to the three layers:

- Persistent main A-REX process (thread of HED) and persistent generator thread
- Persistent scheduler process (or thread) with temporary threads for pre-processing
- Temporary delivery processes created per file transfer

Using processes vs threads should add no more CPU load, however more memory is required. For increasing performance it should be possible to reuse processes in a way similar to how threads are reused.

Implementation Choices

The current system uses files in the control directory for all communication. We may need something more sophisticated for this more complex system, either internal to the new system or also to replace the current files in control dir method. Possibilities:

- Files
- Sockets
- Database
- RPC
- Libevent - http://en.wikipedia.org/wiki/Libevent
- Pipes
- Message passing
  - DBus - http://www.freedesktop.org/wiki/Software/dbus
  - ActiveMQ - http://activemq.apache.org/
- ...

Suggestion 1

- Use persistent object - file, database record - for storing DTRs. Each object includes:
  - Description of DTR
  - Owner of DTR
  - Last requested action
  - Current state inside owner (or maybe inside every module).
- Each DTR has own ID - may be just file name or record key
- Keep simple communication channels between modules - like 2 pipes for example
  - Whenever DTR is changed its ID is sent over communication channel to module which is supposed to react
  - As backup modules can scan modification timestamps of objects periodically

Pros:

- Simple
- Persistency

Cons:

- Monitoring of ever changing state - like bandwidth usage - would require constant modification of files/records.
  - This problem could be solved by providing information which needs no persistency through communication channel. But that would make communication more complex.
  - Another possible solution is to mmap (should work well with files) objects and flush them to persistent store only if persistent part of information is modified.

Suggestion 2 (used in current implementation)

- DTR objects can be rather complicated, so keep them only in memory
- In case of process failure all DTRs are reconstructed from control files into initial NEW state
- DTRs are passed as objects between threads
  - Separate threads run for A-REX (including the Generator), Scheduler, and Delivery components
  - Delivery thread starts new processes for each transfer
    - These communicate simple status messages ("heartbeats") through pipes to main delivery thread
- Communication between threads through callbacks

Pros:

- Simplifies development - no need for complex persistency layer or serialisation code
- Fast communication through thread callbacks

Cons:

- Having lots of threads increases risks of deadlocks and race conditions
- No persistency
- No way to communicate from scheduler to transfer processes
  - If the only communication required is to pause/resume/cancel a transfer it can be done through signals eg SIGSTOP/SIGCONT/SIGTERM

Implementation idea for Suggestion 2

- Generator, Scheduler, Delivery processes are singletons and run as persistent threads. The Processor is a singleton.
  - Note: C++-wise it would be probably more correct to use static methods instead of singletons. With such approach where would be no need to handle singletons outside of class itself hence simplifying interface. Although inside class there may be singleton.
- A-REX initiates a request to the data staging system by calling Generator::receiveJob()
  - Generator creates DTRs and after they finish it modifies the job state directly
  - A-REX can query job status from Scheduler via Generator
- Generator communicates with scheduler by calling Scheduler::receive_dtr method
  - There is also a Scheduler::cancel_dtrs method which is called by the Generator to cancel DTRs
- Scheduler has an instance of DTRList class, which is the storage for all the DTRs in the system.
- Scheduler communicates (passes a DTR) to any other process by calling DTR::push() method
- Pre-, post-processor and delivery communicate with scheduler by simply changing the status of DTR and changing the owner to "SCHEDULER". Scheduler will pick up DTRs with changed statuses automatically during next iteration.
- When DTR goes from the scheduler to the pre or post-processor, it calls the processDTR() method of the singleton processor within push(). The processor then spawns a thread and returns.
- When DTR goes from the scheduler to delivery, within push() it calls the processDTR() method of the singleton delivery. The delivery then spawns a process and returns.
  - The delivery singleton receives messages through pipes from the transfer processes and reports information in the DTR object.

Detailed description of DTRs

DTR stands for Data Transfer Request. This is the structure that contains several fields that fully describe the file transfer to be performed. One DTR is generated by the generator per each file transfer.

Fields of the DTR

More or less required:

- DTR ID
- source endpoint
- destination endpoint
  - for source and destination, a list of metadata such as file size, checksum, creation date etc
  - for source and destination (if applicable) a list of replicas
  - for source and destination (if applicable) current replica
  - for source and destination (if applicable) TURL or delivery-level URL used for transfer
  - for source and destination (if applicable) request ID (in the case of asynchronous requests to remote storage services)
- credentials
- cache information
  - if the file is cacheable, the filename in cache

  - cache directories configuration
  - caching state (already in cache, cache currently locked etc)
- local user information (uid/gid)
- Job ID this transfer belongs to
- priority of the transfer - a number set by the generator which flattens priorities
- transfer share this DTR belongs to
- sub-share the DTR belongs to - may be set by the Generator
- tries left
- flags to handle properties and strategies when dealing with index servers
  - flag to say whether DTR is replicating inside the same logical filename
  - flag to say whether DTR should force registration to an existing logical filename, if the source is different
- mapping info - mapping information of local files to which remote files may be mapped to in the configuration (copyurl/linkurl)
- status of the DTR
- error status
  - type of error
  - location of error
  - text description of error detail
- number of bytes transferred/offset
- timing properties
  - timeout - time which DTR is allowed to remain in current state
  - creation time
  - last modification time
  - process time - wait until this time to do further processing
- cancel (set to true if request is to be cancelled)
- bulk operation flags to combine several DTRs in a bulk request
- delivery endpoint, whether Delivery is to be carried out by a local process or remote service
- current owner - who is in charge for this DTR right now
- logger object, so each DTR can have its own log
- lock, since DTRs can be modified by several processes, for avoiding writing collisions

Possible

- affiliation (if we use the affiliation of multiple DTRs, see right below).
- history of states

Multiple DTRs may be affiliated together. Possible reasons and uses:

- Belong to same job
- Belong to bunch of jobs which user indicated as preferably processed together
- Belong to same VO and assigned priorities to be applied within group
- Failure of one DTR in group may cancel processing of other DTRs (should be implemented in Generator)

State transitions of DTR

All possible states of a DTR, with arrows indicating the normal flow of DTRs between states. Each state is explained in detail below. Error conditions are not included here but are shown in another diagram further down.

Status codes

The following table describes all non-error status codes, and also the action taken in the event of a cancellation request being received while in that state. In general if all of the data transfer has been completed before receiving a cancellation request, the destination file is not deleted. The main reason for this is to preserve cache files, as the user may wish to run the same job soon after cancelling it.

Table 5.

Status Code	Text Description
Statuses set by the generator	
NEW	The DTR has just been built by the generator
CANCEL	A request has been made to cancel the DTR
Statuses set by the scheduler	
CHECK_CACHE	The DTR destination is cacheable and the cache should be checked for the file's existenc
RESOLVE	The DTR source is a meta-protocol and should be resolved
QUERY_REPLICA	The DTR source should be queried to check existence, check file size, checksum etc.
PRE_CLEAN	The destination in the DTR should be deleted before writing
STAGE_PREPARE_SOURCE	The DTR source is a meta-protocol which must be prepared or staged
STAGE_PREPARE_DESTINA TION	The DTR destination is a meta-protocol which must be prepared or staged
TRANSFER_WAIT	The DTR is ready to be sent to delivery but must wait due to transfer limits or priority se
TRANSFER	The DTR should be transferred immediately
RELEASE_REQUEST	The DTR transfer has finished and any requests made on remote storage should be releas
REGISTER_REPLICA	The DTR destination is a meta-protocol and the new replica should be registered
PROCESS_CACHE	The DTR destination is cacheable and the cached file should be unlocked and linked/copi
DONE	The DTR completed successfully
CANCELLED	The DTR has been cancelled succesfully
ERROR	An error occurred with the DTR
Statuses set by the pre-processor	
CHECKING_CACHE	The pre-processor is checking the cache
CACHE_WAIT	The cache file is locked and the scheduler should wait before trying to obtain the lock
CACHE_CHECKED	The cache check is complete
RESOLVING	The pre-processor is resolving replicas
RESOLVED	The replica resolution is complete
QUERYING_REPLICA	The pre-processor is querying a replica
REPLICA_QUERIED	The replica querying is complete
PRE_CLEANING	The pre-processor is deleting the destination file
PRE_CLEANED	The destination file has been deleted
STAGING_PREPARING	The pre-processor is making a staging or preparing request
STAGING_PREPARING_WAI T	The staging or preparing request is not ready and the scheduler should wait before pollin
STAGED_PREPARED	The staging or preparing request is complete
Statuses set by the delivery	
TRANSFERRING	The transfer of the DTR is on-going
TRANSFERRED	The transfer completed successfully
Statuses set by the post-processor	
RELEASING_REQUEST	The post-processor is releasing a stage or prepare request
REQUEST_RELEASED	The release of stage or prepare request is complete
REGISTERING_REPLICA	The post-processor is registering a replica in an index service
REPLICA_REGISTERED	Replica registration is complete
PROCESSING_CACHE	The post-processor is releasing locks and copying/linking the cached file to the session d

Table 5.1 – c

Status Code	Text Description
CACHE_PROCESSED	Cache processing is complete

Error Conditions of DTRs

The following diagram shows possible error conditions and actions taken. For simplicity and because all error handling logic takes place within the scheduler, the pre- and post-processor and the delivery layers are not shown.

Errors are categorised into the following types:

Error	Explanation	Retry	Action
INTERNAL_LOG ERROR	Internal error in data staging logic	No	Stop processing and report back to generator
INTERNAL_PRO S_ERROR	Internal error like losing contact with an external process	Yes	Clean if necessary and retry
SELF_REF N_ERROR	Attempt to copy a file to itself	No	Return to generator
CACHE_E	A problem occurred in cache handling	Yes	Retry without caching
TEMPORARY_RE E_ERROR	Error such as connection timeout on remote service	Yes	Retry with an increasing back-off
PERMANENT_RE E_ERROR	Error such as file not existing, permission denied etc on remote service	No	Follow cancellation steps and return failed DTR to generator
LOCAL_FILE R	Error with a local file	No	Follow cancellation steps and return to generator
TRANSFER_SPEE ERROR	Transfer rate was below specified limits	Yes	Retry transfer. If all retries fail, report back to generator - it will make the decision on whether to cancel other related DTRs. (Future work: make decision on whether other transfers caused slow transfer and whether cancelling others would help or should be done)
STAGING_TIME _ERROR	The staging process took too long	No	Try a different replica - if none available, cancel and report back to generator

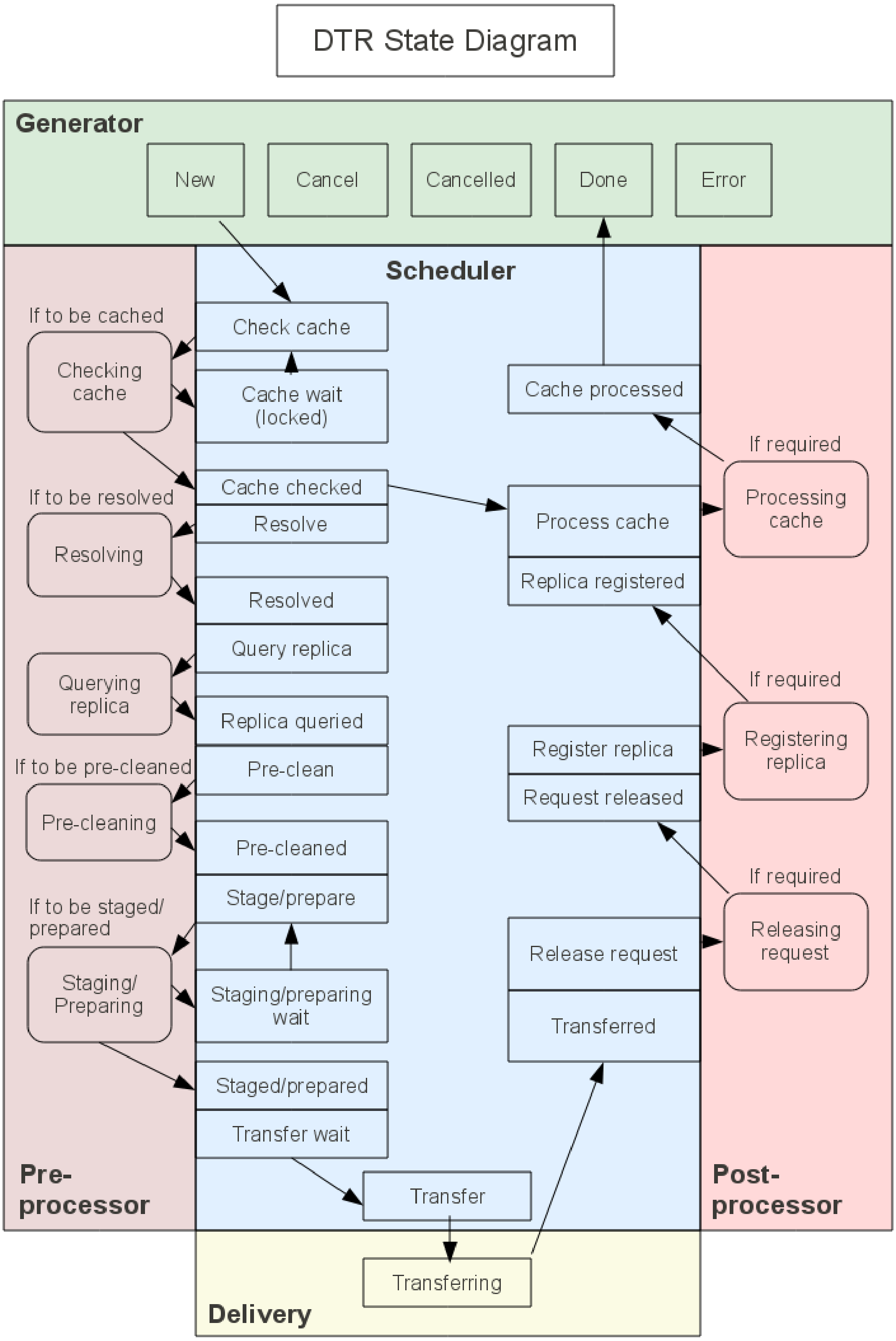


Fig. 5.6: DTR State Diagram

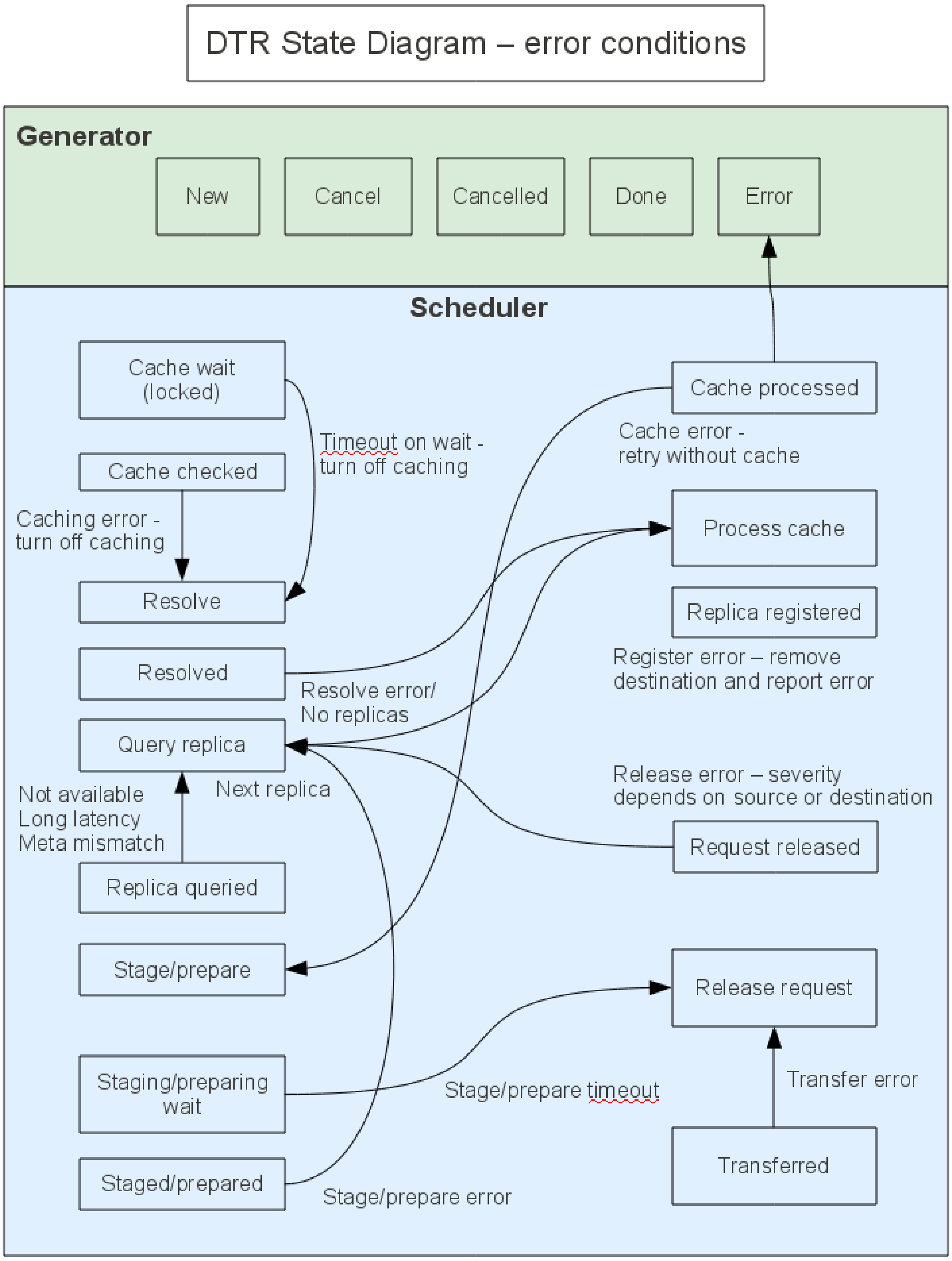


Fig. 5.7: DTR Error State Diagram

Methods of DTRs

DTR::push (DTR, receiver) – pass the DTR from one process to another, e.g. DTR::push (dtr, preprocessor)

Implementation

Within Data Staging framework there is a global list of DTRs. Pointers to the DTRs are passed around between components, which can modify them directly and push them between each other.

DTR priority and shares system

Here we describe the priority and shares system for the new data staging framework. During the design stage there were several ideas taken from other research in the field and the first implementation of the transfer shares model in ARC.

Ideas behind priorities and fair-share in data staging

The initial idea was giving every DTR that comes into the system a fixed priority, then sorting the queue according to priorities and launching the first N DTRs with the highest priorities. This scheme also allows easy incorporation of pre-emption: if the job with higher priority appears it just pushes other DTRs out of the front of the queue and then during the next scheduler loop we can start these DTRs and suspend the pushed ones.

However, this idea can potentially lead to the situation that demanded the implementation of transfer shares in ARC. If a user or VO with the highest priority submits bunch of jobs at once all the other will be blocked, because DTRs from this bunch will occupy the front of the queue for a long time.

The idea of transfer shares comes in handy now. The available transfer slots should be shared among different VOs/Users, so nobody would be blocked. VOs/Users with higher priority get more transfer slots than the other. However, strict limits on the number of slots per share are not flexible enough - if the transfer pattern changes then strict limits could cause problems, squeezing lots of users/jobs into one share with a few slots and blocking others. The User or VO must also be able to decide the relative priority of jobs within its own share.

Current Implementation

The ideas above led to the creation of two configurable properties: user-defined job priority and server-defined share priority. Users may define a priority for their jobs in the job description ("priority" attribute in xrsl), and this is a measure of the priority given to this job within the share it gets assigned to when submitted. On the server-side, it is possible to define a share type, and priorities of certain shares. The share priority is used to determine the number of transfer slots to assign to the share, taking into account which shares are currently active (active meaning the share has at least one DTR in the system).

When the Scheduler receives a new DTR, it is placed into a transfer share, which is defined by a User DN, VO, Group inside VO or role inside VO as it was in previous versions of ARC. Currently it's possible to use only one sharing criteria in the configuration, i.e. it's not possible to use simultaneously sharing by User and VO.

Priority is defined as a number between 1 and 100 inclusive - a higher number is a higher priority. In the A-REX configuration it is possible to specify a base priority for certain shares. If the DTR doesn't belong to any of these specified shares, it is placed in a "_default" share with default base priority (50). The scheduler sets the priority of the DTR to the base priority of the share multiplied by the user priority from the job description (default 50) divided by 100, therefore default priority of a DTR is 25. In this system the priority set in the job description effectively defines a percentage of the base priority. Thus service administrators can set maximum priority limits for certain shares, but users or VOs have full control of their jobs' priority within the share.

While revising the Delivery and Processor queues, the scheduler separates DTRs according to the shares they belong to. Inside every share DTRs are sorted according to their priorities. Then the scheduler determines the number of transfer slots that every active share can grab. The number is determined dynamically depending on priorities of active shares. Each share receives the number of slots which corresponds to the weight of its priority in the summed priority of all active shares. After the number of slots for each share is determined the scheduler just launches N[i] highest priority DTRs in each share, where N[i] is the number of transfer slots for i-th share.

The reason for weighting the DTR priority by the share priority is for occasions when the Scheduler considers the entire queue of DTRs, for example when allowing highest priority DTRs to pass certain limits.

Example: there are two active shares, one has base priority 60, the other 40. The summarized priority is 100 (60 + 40). The first share has a weight of 60%, the second 40%. So the first will grab 60% of configured transfer slots, and the second – 40%. If the system is configured with 5 Delivery slots, then the first share will take 3 slots and the second 2 slots. The 3 highest priority DTRs from the first share and 2 highest priority from the second share will be assigned to those slots.

Emergency Shares

To avoid the situation where a fixed limit of slots are used up by slow transfers and a new high priority transfer has to wait for a slot, we have "emergency" transfer slots. If there are transfers in the queue from a particular share, but all slots are filled with transfers from other shares, one emergency slot can be assigned to this share to allow transfers to start immediately. The share may use an emergency slot until any other transfer finishes, at which point the emergency slot becomes a regular slot and a new transfer does not start from the queue.

Sub-shares

The Generator can assign DTRs to "sub-shares" to give a higher granularity than the standard criteria and when assigning transfer slots. Sub-shares are treated as separate shares. In A-REX, different sub-shares are assigned to downloads and uploads, and in this case emergency transfer slots prove useful for preventing jobs not being able to finish because all transfer slots are taken by downloaders. If this happens then emergency slots can be used for uploads.

Potential Problems

- Within a share, high priority jobs block low priority jobs. Thus if there is a constant stream of high priority jobs in a share, then some low priority jobs in the same share may never run. Possible solutions:
  - Increasing the priority as the time spent in the queue increases (returning to previous priority after leaving the queue). This is currently implemented as increasing the priority by 1 every 5 minutes after the DTR's timeout has passed.
  - Changing simple highest-priority-first to a random algorithm where higher priorities are weighted higher
  - Making a higher granularity of shares by splitting each priority or priority range into its own share - this is probably too complicated

A-REX Configuration

The configuration varies depending on the ARC version. In the examples below VO roles are used to assign shares, the atlas slow_prod role is assigned a low priority share and the atlas validation role is assigned a higher priority share.

```
[arex/data-staging]
sharepolicy=voms:role
sharepriority=atlas:slow-prod 20
sharepriority=atlas:validation 80
```

If both shares are active and there are 10 slots available, then DTRs in the slow-prod share will get 2 slots and those in the validation share get 8 slots, and so the jobs in the validation share will have a higher throughput (assuming similar numbers of files and file sizes in each type of job).

Example

A user wants their job to be high (but not top) priority and specifies ("priority" = "80") in the job description. The user has a VOMS proxy with no role defined and submits the job to a site with the above configuration. The job is assigned to the default share and DTRs have priority 40 (50 x 80 / 100). The user then creates a VOMS proxy with the ATLAS validation role and submits another job with the same priority to the same site. This time the job goes to the configured atlas:validation share and the DTRs have priority 64 (80 x 80 / 100). Note that the priority of a DTR only affects the its position within a share and does not influence the distribution of slots between shares.

Dynamically modifying priorties

From version 6.2 ARC supports dynamically adjusting priorities of active DTRs for example to push certain transfers to the front of the queue.

This can be done by creating a file in the same location as the dtr.state file (by default the control dir) with the name dtr.state.prio. This contains lines of DTR ID and priority, eg:

```
c3e12bfa-ba9a-4a45-be73-0b3b2a206e1c 90
ff2d5329-5440-49aa-83cb-72daeb5e3e59 95
```

This file is read by the DTR system before it sorts the queue by priority and it changes any DTRs found in the file to the given priority. Then it renames the file to dtr.state.prio.read.

5.1.3 ARC Data Delivery Service Technical Description

This page describes how to set up DTR over multiple hosts.

Introduction

In DTR the transfers can be split over multiple hosts to scale up the data transfer throughput. Remote hosts only perform simple point-to-point transfers between physical endpoints and all the logic is kept in the main A-REX host. This has many advantages:

- Keeping all the high-level logic in one place allows intelligent load balancing.
- The remote hosts do not do submission to the batch system and so all LRMS configuration only needs to be done in one place.
- A problem with a remote host does not mean a lost job - the transfer can simply be retried on another host.
- The setup and configuration of a remote host is very simple.

The datadelivery service has been developed within HED and can be deployed on one or many remote hosts. Once a DTR has passed all the pre-processing steps and is ready for transfer, the Scheduler can send it to a remote datadelivery service to execute the transfer. The Scheduler will then poll the request until it has completed. The Scheduler spreads randomly the total number of transfers (up to the configured limit and depending on the configured access rights of each service) between all the staging hosts.

The architecture is shown in the figure below. The datadelivery service on the remote host acts as a very simple Scheduler which sends all transfer requests it receives straight to Delivery.

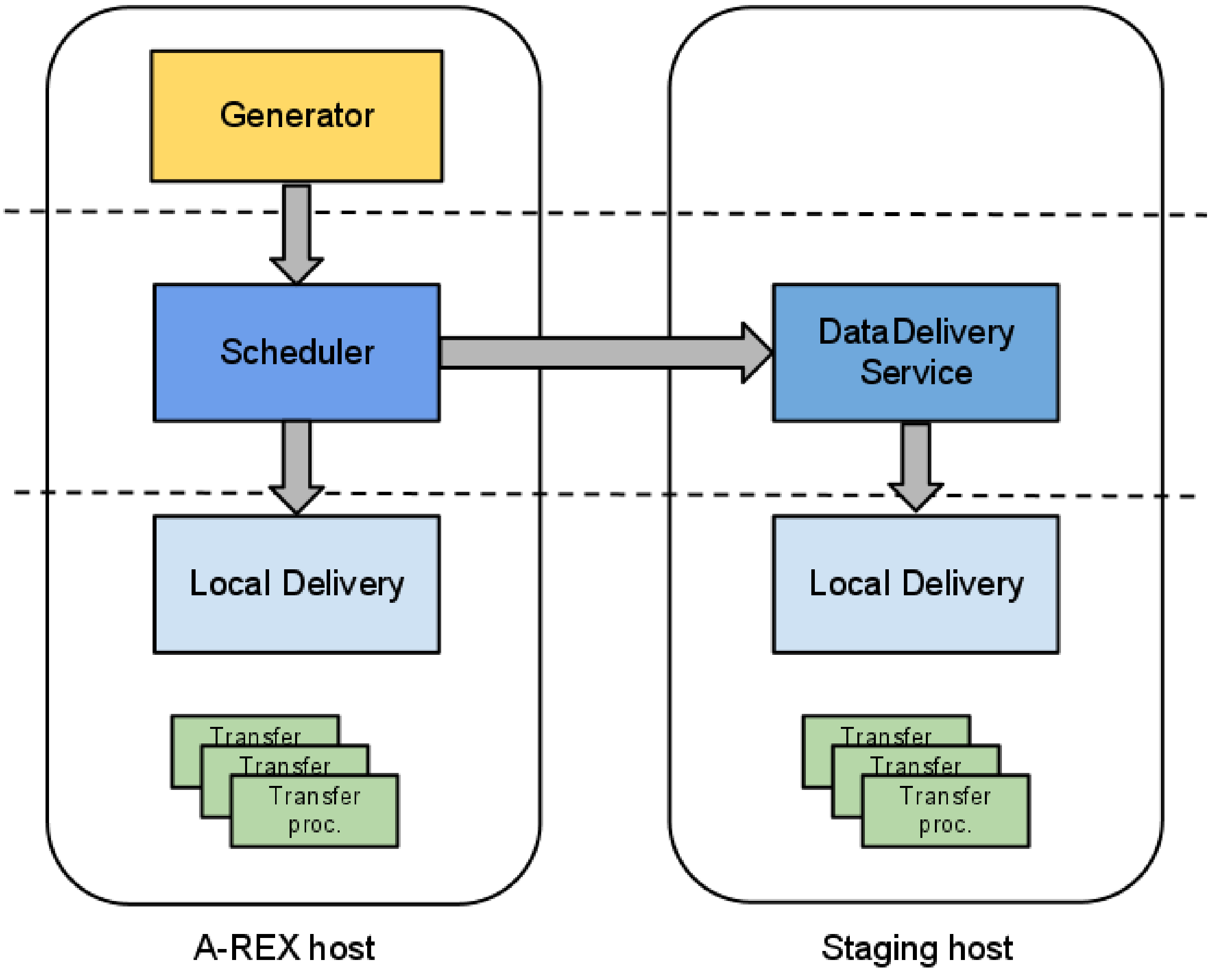


Fig. 5.8: Multi-host datastaging

Installation

No extra components are necessary on the A-REX host.

For the remote staging hosts, the package **nordugrid-arc-datadelivery-service** can be found in the usual EPEL or NorduGrid repositories, or is built automatically when building the source tree. This package should be installed on each remote host. It depends on some core ARC packages and the usual external packages that an ARC installation requires. Note that it is not necessary to install A-REX on the staging hosts.

IMPORTANT If support for transfer protocols that are not included in the **nordugrid-arc-plugins-base** package (e.g. GridFTP or Xrootd) is required then the corresponding plugins packages must also be installed.

CA certificates are required on each remote host to authenticate connections with storage services. Depending on the local set up, each host may also require a host certificate (more info below).

To start|stop the service (as superuser):

```
systemctl start|stop arc-datadelivery-service
```

The log file for the service can be found at /var/log/arc/datadelivery-service.log (configurable - see below). If logrotate is running, this log will be rotated every day.

Configuration

Remote Hosts

The datadelivery service uses the [datadelivery-service] section of the arc.conf configuration file.

By default the datadelivery service runs with TLS enabled and a host certificate is required for each host. The path to the host credentials may be specified by the usual x509 options. Host certificates are not a strict requirement and it is possible to run without TLS using the secure=no option. This means that A-REX can no longer verify the authenticity of the remote hosts and so the decision to run with or without host certificates should be based on local site policy.

The following configuration options are supported:

Parameter	Explanation	Default Value
x509_host_k	Path to host key	/etc/grid-security/hostkey.pem
x509_host_c	Path to host cert	/etc/grid-security/hostcert.pem
x509_cert_di	Path to CA certificates	/etc/grid-security/certificates
hostname	Hostname of service host	localhost
port	Port on which service runs	443
pidfile	pid file	/var/run/arched-datadelivery-service.pid
logfile	Log file	/var/log/arc/datadelivery-service.log
loglevel	Logging level (0 (FATAL) to 5 (DEBUG))	2 (WARNING)
user	User under which service runs (should only be changed in special cases)	root
secure	Set to "no" if the service should run without a host certificate	yes
allowed_ip	IP address authorized to access service (can be specified multiple times)	No default, must be specified
allowed_dn	DN authorized to access service (can be specified multiple times)	No default
transfer_dir	Path the service is allowed to read/write to (can be specified multiple times)	No default, must be specified must be specified

At least one allowed_ip and at least one transfer_dir are the only mandatory parameters, but it can be useful to change interface, port and loglevel from the default.

Since the service can copy files to and from the service host, it is dangerous to allow open access to any clients. Usually allowed_ip is set to the IP address of the A-REX host since this is the only host which should have access to the service. The service can be further locked down by specifying authorized DNs so that only certain users are allowed to have their jobs' files staged by the service. Note that DN filtering is not possible if secure=no is specified.

It is also dangerous to allow requests which can copy to or from anywhere on the host filesystem, so filesystem access is restricted through the transfer_dir parameter, which specifies the path(s) that requests are allowed to use. The service is able to read and write to any path matching transfer_dir. transfer_dir(s) should be specified which match every cache and session dir. In some situations it may be desirable to set for example one cache per remote host where the cache is local to the host. A-REX checks on start-up which dirs are accessible by which remote hosts and uses that info to direct the DTRs to the right hosts.

Configuration example:

```
[datadelivery-service]
loglevel = 3
hostname = delivery.host.1
port = 60002
allowed_ip = 1.2.3.4
transfer_dir = /var/arc/session
transfer_dir = /var/arc/cache
```

If the site has multiple caches like **/var/arc/cache-01**, **/var/arc/cache-02** etc, then this configuration will allow the delivery service to write to all of them.

It is vital that the cache and session file systems are mounted on the same path on the A-REX and each remote host, as the system assumes that a transfer to the cache or session directory can be done using the same path on all hosts. It is also vital that user accounts are in sync across all hosts, so that a user represented by a uid on the A-REX host maps to the same account on all the remote hosts. No mapping of DN to local user is done on the remote hosts, and so no mapping infrastructure like gridmap files is required. The local user id of the user owning the transfer is passed in the request to the service and the transfer process is executed under that uid (except when writing to cache, when the root account is used).

The service is started and stopped by the arc-datadelivery-service systemd unit:

```
systemctl start arc-datadelivery-service
```

A-REX Host

The configuration for setting up A-REX to use remote datadelivery services uses options in the [arex/data-staging] section of the regular arc.conf.

Parameter	Explanation	Default Value
deliveryservice	URL of remote host which can perform data delivery. Hostname and port must match those specified in the datadelivery service configuration.	None
localdelivery	Whether local delivery should also be done	no
remotesizelimit	File size limit (in bytes) below which local transfer is always used	0
usehostcert	Whether the host certificate should be used in communication with remote datadelivery services instead of the user's proxy	no

Example:

```
[arex/data-staging]
deliveryservice = https://delivery.host.1:60002/datadeliveryservice
deliveryservice = https://delivery.host.2:60002/datadeliveryservice
localdelivery = yes
remotesizelimit = 1000000
```

Multiple remote datadelivery services can be specified, and the Scheduler will randomly divide DTRs between those services where the transfer is allowed (according to transfer_dir configuration on each service). The presence of a datadeliveryservice option turns off the regular "local" delivery on the A-REX host so only the remote service will be used. If it is desired to also do transfers on the A-REX host, **localdelivery = yes** must be used. If no remote services are specified then local delivery is always enabled.

If the datadelivery service is running with secure=no then https should be replaced by http in the deliveryservice URLs.

Normally the credentials of the user who submitted the job are used for communication between A-REX and remote datadelivery services, however for extra security it is possible to use the A-REX host certificate for this instead, by specifying **usehostcert = yes**. In this case the host cert is only used for establishing the secure connection with the remote service, it is still the user's credentials which are delegated and used for the transfer. The host cert must also be able to be used as a client certificate, in other words must have the X509 extension "X509v3 Extended Key Usage: TLS Web Client Authentication". This option has no effect if the datadelivery service is running with secure=no.

Communication with remote services involves some degree of overhead such as the SSL handshake, delegating credentials etc, and when transferring small files this overhead can become a significant fraction of the transfer time. Therefore it is possible to specify a file size limit (in bytes) with **remotesizelimit**. Any files smaller than this limit will use local transfer, even if local transfer is disabled through **localdelivery = no**.

Deployment Scenarios

Several ways of deploying multi-host data staging are possible, using any number of remote hosts. Two examples are shown here.

Shared Storage

The cache and session directories are on a storage system such as Lustre or GPFS, which is mounted on all hosts. All hosts can access the cache and session directory and so DTRs will be split randomly between them. No transfer is done on the A-REX host.

Local Caches

Each remote host has its own local disk as a cache and these caches are mounted on the A-REX host. DTRs will be sent to the host corresponding to the cache chosen for the DTR so that all cache transfers are to local disks. The session directory is not available on the remote hosts so all uploads and non-cache downloads will run on the A-REX host.

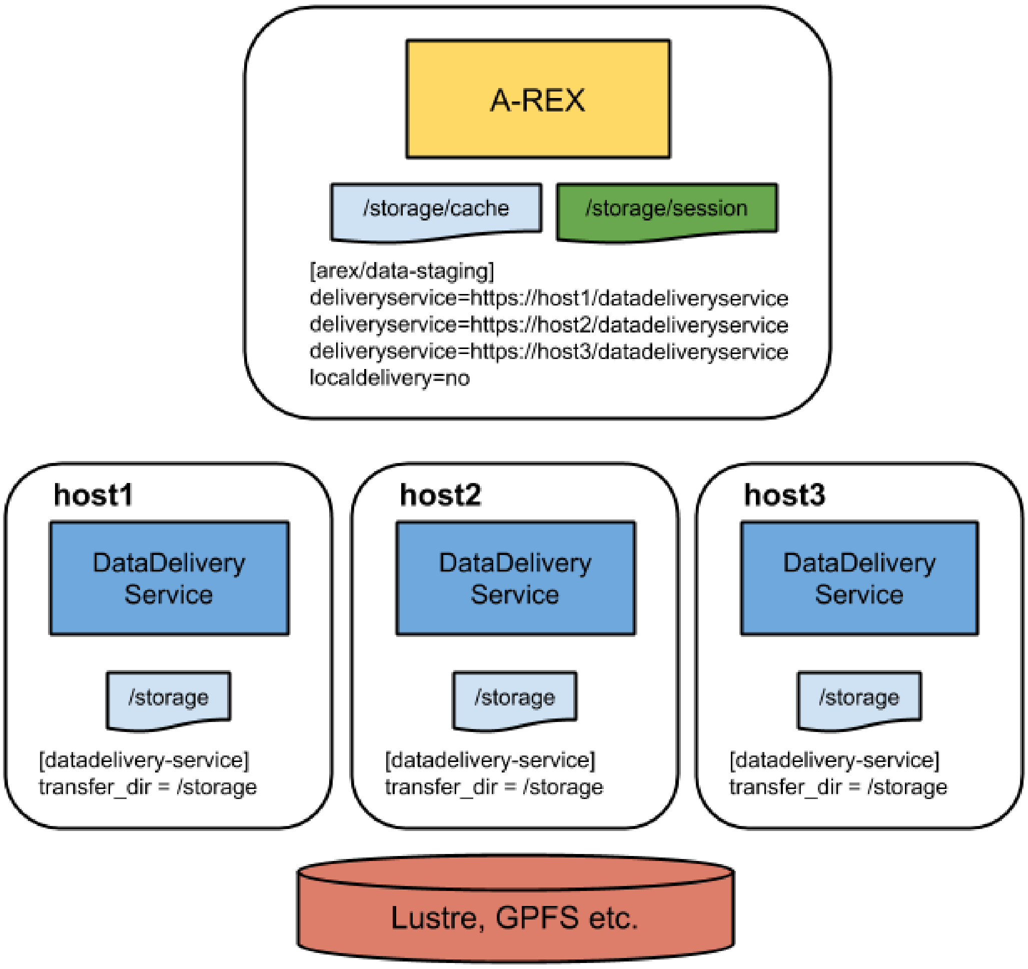


Fig. 5.9: DDS deployment: shared storage

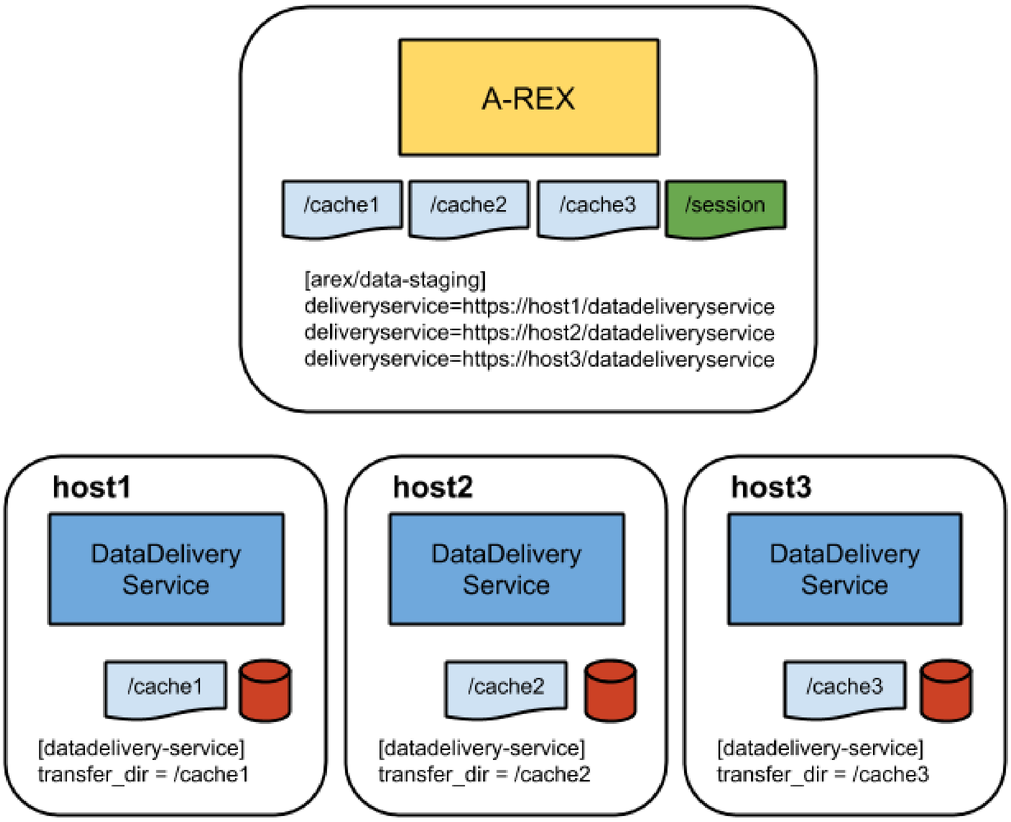


Fig. 5.10: DDS deployment: local disk for cache on DDS nodes

Security

The remote datadelivery service may require a host certificate (see above) and allow incoming connections from the A-REX host, but no other incoming connections are required. It must have outbound connectivity to the world to perform the transfers. As explained above, access is normally restricted to the A-REX host and transfers restricted to the cache and session directories. For each transfer A-REX delegates the credentials of the user who submitted the job to the remote service, which creates a temporary file containing the credentials. This file is deleted when the transfer finishes. Creating a file should become unnecessary if pure in-memory credentials are fully supported by ARC and all transfer protocols.

The transfer process itself is executed under the uid of the session directory owner, unless the transfer is to cache in which case it is executed by root. It is therefore important that user accounts are synchronised across all hosts. The file systems with the cache and session directories must be mounted on the same paths with the same user access rights on all hosts.

Proxies

HED services do not support legacy proxies such as those generated by default by voms-proxy-init. In order to use remote datadelivery services, the jobs must be submitted to A-REX using RFC proxies, which can be generated by arcproxy or giving the option -rfc to voms-proxy-init. If a legacy proxy is used, local transfer will be used even if it is disabled in the configuration.

Monitoring Remote Hosts

When the first DTR is received by the Scheduler, it pings all the configured remote datadelivery services to check that they are running and to get the list of allowed directories from each one. If a service is unreachable it will not be used. If all services are unreachable then local delivery will be used even if it is turned off in the configuration. The allowed directories information is used to direct DTRs to services where the transfer is allowed. This procedure is repeated every 5 minutes (for the first DTR received after the 5 minute limit), and only the successful services are used until the next check (unless none are successful, in which case the check is done for every DTR until one succeeds). This means that configuration changes in the remote hosts will be picked up automatically after some time. However any changes in A-REX's configuration, such as adding a new remote host, require an A-REX restart to be effective.

If remote datadelivery services are enabled, the number of DTRs assigned to each one can be seen on separate plots in Gangliarc.

5.1.4 The ARC Cache Index (ACIX)

The ARC Cache Index (ACIX) is a catalog of locations of cached files.

It consists of two components, one on the computing resource: the *ACIX Scanner*, and the *ACIX Index* which indexes the cache locations retrieved from the ACIX Scanners. These components can be found respectively in the packages `nordugrid-arc-acix-scanner` and `nordugrid-arc-acix-index`. They both depend on a third package, `nordugrid-arc-acix-core`.

ACIX Scanner

The ACIX Scanner periodically scans the *A-REX cache* and constructs a Bloom filter of cache content. This filter is a way of representing the cache content in an extremely compressed format, which allows fast query of any element of the filter and efficient upload of the content to an index server.

This type of compression however has the possibility of giving false-positives, i.e. a certain file may appear to be present in the cache according to the filter when it is not. The ACIX Scanner runs in an HTTPS server and the filter is accessible at the endpoint `https://hostname:5443/data/cache`.

It scans the caches specified in the A-REX `arc.conf`. It does not require any configuration but some options can be changed and it is important to make sure the ACIX Scanner port (default 5443) is open in the firewall.

ACIX Index

The ACIX Index server runs independently of the ACIX Scanner and A-REX, but can be deployed on the same host as both of them. It is configured with a list of ACIX Scanners and periodically pulls the cache filter from each one. It runs within an HTTPS server through which users can query the cached locations of files. Configuration uses the regular `arc.conf` file in the *[acix-index] block*. Here ACIX Scanners are specified by the *cachescanner* option. For example:

```
[acix-index]
cachescanner = https://my.host:5443/data/cache
cachescanner = https://another.host:5443/data/cache
```

The ACIX Index server can be queried at the endpoint `https://hostname:6443/data/index` and the list of URLs to check are given as comma-separated values to the option "url" of this URL, e.g:

```
https://hostname:6443/data/index?url=http://www.nordugrid.org:80/data/echo.sh,\
 http://my.host/data1
```

A JSON-formatted response is returned, consisting of a dictionary mapping each URL to a list of locations. If remote access to cache is configured as described above then the location will be the endpoint at which to access the cached file, for example `https://a-rex.host/a-rex/cache`. If not then simply the hostname will be returned.

Using ACIX with A-REX Data Staging

ACIX can be used as a fallback mechanism for A-REX downloads of input files required by jobs by specifying *use_remote_acix* in the *[arex/data-staging] block* of arc.conf, e.g.:

```
[arex/data-staging]
use_remote_acix = https://cacheindex.ndgf.org:6443/data/index
```

If a download from the primary source fails, A-REX can try to use any cached locations provided in ACIX if the cache is exposed at those locations. In some cases it may even be preferred to download from a close cache rather than Grid storage and this can be configured using the *preferredpattern* configuration option which tells A-REX in which order to try and download replicas of a file.

Using ACIX for ARC client brockering

ACIX can also be used for data-based brokering for ARC jobs. An ACIX-based broker plugin written in Python comes packaged with the ARC client tools (in `$ARC_LOCATION/share/arc/examples/PythonBroker/ACIXBroker.py`) and can be used for example with:

```
[user ~]$ arcsub -b PythonBroker:ACIXBroker.ACIXBroker:https://cacheindex.ndgf.
↪org:6443/data/index
```

Target sites for job submission are ranked in order of how many input files required by the job are cached there. See the comments inside this Python file for more information.

Deployment use-case

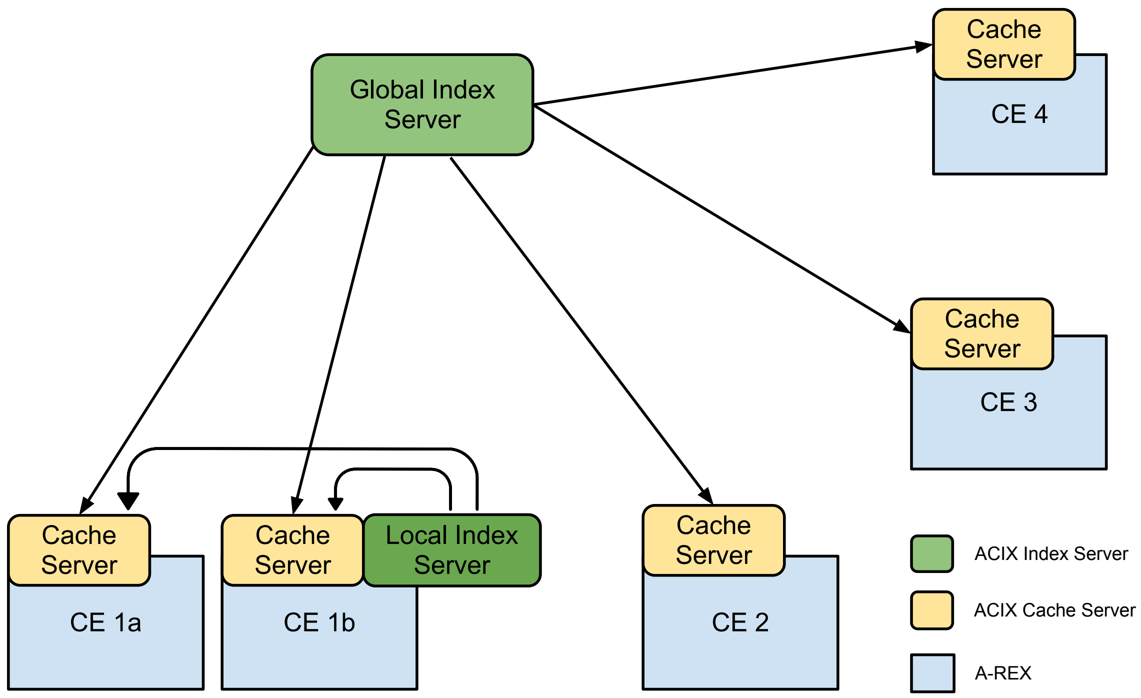


Fig. 5.11: ACIX deployment scenario, with one global ACIX Index and a local ACIX Index for CE 1a and CE 1b.

Fig. 5.11 shows an example ACIX set up. Each CE runs a ACIX Scanner and there is a central ACIX Index server which pulls content from all CEs. In addition there is one site with two CEs, CE 1a and CE 1b.

In order to do data-based brokering on just those two sites (and ease the load on the global ACIX Index server), a local ACIX Index is running which pulls content from only these two sites. In such a setup if may be desired to prefer to dowload data from the cache on CA 1a to CE 1b and vice versa, so those CEs could be configured with the Local ACIX Index server as the `use_remote_acix` and each other's hostname first in `preferredpattern`.

5.1.5 CandyPond technical description

This page describes technical details of CandyPond, which stands for "Cache and deliver your pilot on-demand data".

Description and Purpose

The ARC caching system automatically saves to local disk job input files for use with future jobs. The cache is completely internal to the computing element and cannot be accessed or manipulated from the outside. CandyPond exposes various operations of the cache to the outside and can be useful in a pilot job model where input data for jobs is not known until the job is running on the worker node. When the pilot picks its payload it can contact CandyPond to gain access to a file that is already cached, or if it is not cached ask CandyPond to download the file to cache.

Installation and Configuration

CandyPond is an integral part of A-REX and is available as part of the `nordugrid-arc-arex` package.

It is enabled in A-REX by adding the block `[arex/ws/candypond]` to arc.conf. The `[arex]`, `[arex/ws]` and `[arex/data-staging]` blocks are also required.

Runtime Environment Configuration

A runtime environment `ENV/CANDYPOND` exists to provide a convenient python module `arccandypond` to the job running on the worker node. This can either be used as a command line interface or as a python API.

Note that the `ENV/PROXY` runtime environment is also needed in order to have access to the proxy on the worker node.

`ENV/CANDYPOND` will automatically detect the correct URL of the CandyPond service, but if it is desired to use a different URL then it can be set with

```
arcctl rte params-set ENV/CANDYPOND CANDYPOND_URL <url>
```

Command Line Interface

`arccandypond get <url> <file>` can be used in place of whatever usual command the job would use to download input data. This command asks Candypond to download the url to cache if not already present, and link to the file specified in the job's working directory.

`arccandypond check <url>` can be used to check if the given url already exists in the cache. It will exit with `0` if the file is present, 1 if not, or 2 if an error occurred.

Python API

The job can import the module and use the `CacheLink` and `CacheCheck` methods to perform the equivalent of `get` and `check` commands.

Example Use Case

In this example a job is submitted which uses `arccandypond` to download input data to cache and have it available to the job.

The xrsl file defines the required runtime environments. Note that no input files are specified.

```
$ cat candypond.xrsl
&
("executable" = "candypond.sh")
("runtimeenvironment" = "ENV/CANDYPOND")
("runtimeenvironment" = "ENV/PROXY")
("jobname" = "candypond_test" )
("walltime" = "3600" )
("cputime" = "3600" )
("stderr" = "stderr")
("stdout" = "stdout")
("gmlog" = "gmlog")
("outputfiles" =
   ("stdout" "")
   ("stderr" "")
   ("gmlog" "")
)
```

The job script uses candypond to download the input file to cache and link to the job's working directory:

```
$ cat candypond.sh
#!/bin/sh
arccandypond get http://www.nordugrid.org:80/data/run.sh run.sh
ls -lrt
echo
cat run.sh
```

Submit the job:

```
$ arcsub candypond.xrsl
Job submitted with jobid: https://...
```

Check the output:

```
$ arccat https://...
{'http://www.nordugrid.org:80/data/run.sh': ('0', 'Success')}
total 28
-rwx------ 1 dcameron dcameron   257 Apr 10 20:23 candypond.sh
drwx------ 4 dcameron dcameron  4096 Apr 10 20:23 arc
-rw------- 1 dcameron dcameron 10662 Apr 10 20:23 user.proxy
-rw------- 1 dcameron dcameron     0 Apr 10 20:23 stderr
lrwxrwxrwx 1 dcameron dcameron    89 Apr 10 20:23 run.sh -> /opt/var/arc/cache/
↪joblinks/eOGODmV3WYunPSAtDmVmuSEmABFKDmABFKDmJSFKDmGBFKDm5lgA5m/run.sh
-rw------- 1 dcameron dcameron    62 Apr 10 20:23 stdout

#!/bin/sh

GCC=`which g++ 2>/dev/null`
echo $GCC
if [ -z $GCC ]; then
  echo "Could not find the g++-compiler!"
  exit 0
fi
```

(continues on next page)

(continued from previous page)

```
make
chmod 755 prime
./prime $1
```

Note the symbolic link to the cache.

Issues and Notes

- Calls to `arccandypond get` may block for a long time if the file needs to be downloaded to cache and A-REX is already busy with data staging or the file is very large. A timeout option will be added in the future.
- CandyPond (like the A-REX web service interface it is part of) does not accept legacy proxies. This type of proxy is created by default with older versions of grid/voms-proxy-init, but an RFC-compliant proxy can be generated using the -rfc option.
- CandyPond links files to the session dir. If a scratch directory is used for executing the job, the cache files are moved there from the session directory. This requires that the scratch dir is accessible from the CanyPond host, so it cannot be used in situations where the scratch directory can only be accessed by the underlying LRMS.

5.2 ARC Accounting Technical Details

New in version 6.4.

Changed in version 6.12.

> **Warning:** Information in this chapter is relevant only for 6.4+ ARC releases.
>
> Moreover ARC 6.12 get accounting changes to address the APEL move to ARGO messaging service protocol. If you are publishing to APEL you must update to 6.12+ ARC release.

Note: If you are looking for the information about the technical details of legacy accounting subsystem in 6.0-6.3 ARC releases please read *Legacy JURA Accounting Technical Details* but it is highly recommended to update to recent release.

General accounting configuration and operations flows are described in *Accounting Subsystem*. This section contains more technical details about implementation of each component of accounting subsystem.

5.2.1 Job accounting information processing workflow

Collecting the accounting information

The A-REX *Accounting* subsystem is part of the core A-REX functionality starting from the 6.4 release. The main functionality of A-REX *Accounting* subsystem is to handle the data writing to local SQLite accounting database for the every job state change.

The data sources of the *Accounting* data are per-job files in the *control directory*:

- `.local` file contains general information associated with the job. All IDs, ownership, authtokenattributes are taken from this file. The data in `.local` are written and updated by the A-REX *JobControl* modules.

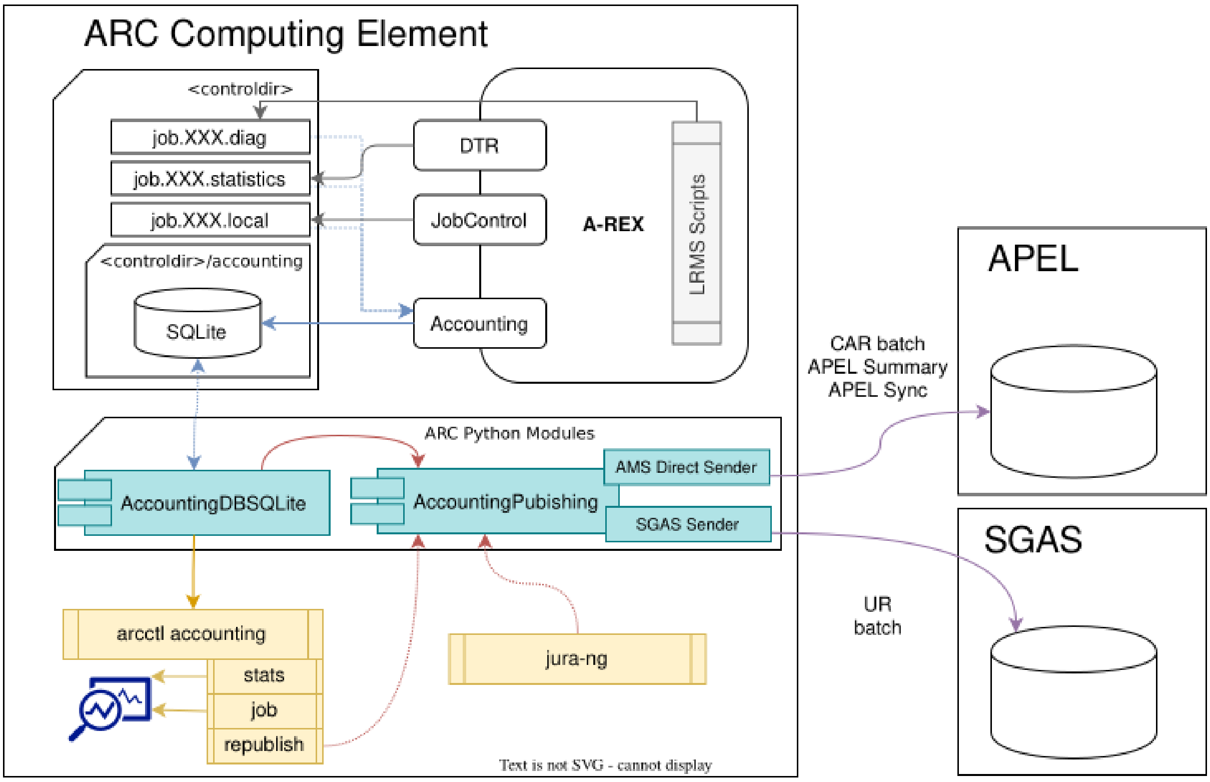


Fig. 5.12: Technical details of ARC CE accounting workflow: information collection, AAR creation, querying and publishing

- `.statistics` file is a dedicated file written by the *DTR* *data transfer framework* that contains data transfer measurements.
- `.diag` file is written by the *LRMS Scripts*: initially by `submit-<lrms>-job.sh`, than *JobScript* during the job execution on the Worker Node and finally by `scan-<lrms>-job.sh` that adds extract from LRMS accounting data. It contains but is not limited to *resource usage* and worker node environment data.

The local SQLite accounting database contains all the *A-Rex Accounting Record (AAR)* data for every ARC CE job.

The initial record about the job is created based on the first ACCEPTED job event. The ID, ownership and submission time is recorded during this step and accounting job `status` is marked as `in-progress`.

Any subsequent job events triggers event data recording in the database, and allow to track data staging time, lrms enqueueing time, etc.

When the FINISHED job event occurs (execution is completed) the A-REX *Accounting* subsystem updates all AAR metrics in the database, storing resource usage, endtime, etc. Such a state is indicated by the `status={completed|failed|aborted}`.

Using the local accounting database

Using the accounting data for statistics lookup and/or publishing to external services is accomplished via the developed `arc.control` Python modules.

The `AccountingDBSQLite` module is responsible for handling all low-level database operations and it hides SQL queries under the API needed for other workflows.

In particular the `accounting` subsystem of *ARC Control Tool* provides a command line interface to the *typical queries* that can get you the accounting data in a flexible manner.

The records publishing is carried out by the `AccountingPublishing` Python module that includes:

- classes for generating usage records in OGF.98 UR, EMI CAR 1.2, APEL Summaries and APEL Sync formats
- classes that handle the records POST-ing to SGAS endpoint
- classes that handle the records sending to APEL via AMS protocol
- general wrapping classes to handle regular publishing and republishing of the data

Both the `arcctl accounting reublish` tool and the `jura-ng` tool (that runs regularly by A-REX) use the same `AccountingPublishing` Python module.

The regular publishing process stores the last published record endtime in the dedicated `Publishing` database. The next round of regular publishing queries the stored time and query the records since then.

5.2.2 Accounting data publishing details

Reporting to SGAS

SGAS has a simple custom web service interface loosely based on WS-ResourceProperties.

`AccountingPublishing` Python module uses the insertion method of this interface to report URs directly to the Python `httplib` library with SSL context wrapping.

To increase communication efficiency the `AccountingPublishing SGASSender` class sends URs in batches. SGAS accepts a batch of URs in a single request. The batch is an XML element called `UsageRecords`, containing elements representing URs. The maximal number of URs in a batch can be set as a *urbatchsize* configuration parameter of SGAS target.

Reporting to APEL

Changed in version 6.12.

APEL curently uses AMS REST protocol for records sending.

The `AccountingPublishing APELAMSDirectSender` class implements the AMS REST communication without external dependencies.

Communication code relies on the same Python `httplib` library with SSL context wrapping.

It connects to AMS endpoint with valid SSL context with client certificate authentication and obtains AMS authentication token.

Messages sent to APEL are S/MIME signed using `openssl` binary tool and than sent to the endpoint using AMS authentication token.

Reporting to APEL also works with sending records in batches. The default *urbatchsize* value is set to 500 according to APEL recommendations but can be lowered if you run into message size issues (e.g. sending large individual records).

Republishing

Republishing simply triggers the same `AccountingPublishing` classes for the defined timeframe that comes from the command line.

All records are regenerated from accounting database data and sent to the target.

5.2.3 Security

The accounting directory `<controldir>/acconting` is by default accessible only by the user running A-REX (root in most cases).

All usage records are submitted with use of the X.509 credentials specified by the value of `x509_` set of configuration options of `arc.conf`. No proxies are used for communication with accounting services.

The only access restriction made by a SGAS service is matching the Distinguished Name of the client (in this context ARC CE) with a set of trusted DNs. When access is granted, policies are then applied by SGAS, allowing either publishing and/or querying rights. Clients with publishing rights can insert any UR, regardless of content. By default, querying rights only allows retrieving URs pertaining to jobs submitted by the querying entity.

Publishing records to APEL requires `glite-APEL` endpoint defined for the grid-site in the GOCDB. The ARC CE certificate DN should be added to the `glite-APEL` endpoint.

5.2.4 Third-party accounting queries

The *ARC Control Tool* accounting stats interface is powerful enough to get custom information from the accounting database as shown in *examples*.

However if you want to get a specific report or integrate ARC accounting database with third-party software you can of cause use SQLite directly.

The SQLite database file location is: `<controldir>/accounting/accounting.db`.

It is worth to be aware of the *ARC Accounting Database Schema* to develop third-party queries.

5.2.5 Definition of the A-REX Accounting Record including attribute mappings to SGAS and APEL

ARC CE is measuring and collecting a lot of accounting information needed but not limited to the data required by common aggregated accounting SGAS and APEL services.

All accounting information stored about a job is defined by what we called *A-REX Accounting Record (AAR)*.

AARs has a representation inside the local accounting database according to schema and representations inside A-REX and Python modules.

Local stats are generated based on the stored AARs information and provides the way for on-site CE operations analyses.

The following tables include a flat list of the properties (NOT the database rendering) included into the AAR:

Table 5.2: Attributes used in current implementation

A-REX Accounting Record (AAR)	SGAS OGF-UR	APEL CAR	Content description
jobid	`JobIdentity.GlobalJobId`, `RecordIdentity` is composed of `jobid` and hostname taken from the `endpointurl`.	`JobIdentity.GlobalJobId`, `RecordIdentity` is composed of `jobid` and hostname taken from the `endpointurl`.	The global unique jobid assigned by AREX.
localid	`JobIdentity.LocalJobId`	`JobIdentity.LocalJobId`	LRMS job ID
jobname	`JobName`	`JobName`	User specified job name
endpointurl	`MachineName`	`MachineName`, `SubmitHost`, `Site`	The A-REX job submission endpoint URL used for this job

continues on next page

Table 5.2 – continued from previous page

A-REX Accounting Record (AAR)	SGAS OGF-UR	APEL CAR	Content description
endpointtype	not used	not used	The A-REX job submission endpoint type used for this job
lrms	not used	`Infrastructure` (used as a part of it)	The LRMS behind A-REX
queue	`Queue`	`Queue`	The name of the LRMS queue of the job
nodename	`Host`	`Host`	WN name(s) as given by LRMS separated by :
clienthost	`SubmitHost` (port removed)	not used	Client connection socket from the client to A-REX
usersn	`UserIdentity.GobalUserName`	`UserIdentity.GobalUserName`	The global user identity, at the moment it is the SN from the certificate
localuser	`UserIdentity.LocalUserId`	`UserIdentity.LocalUserId`	The mapped local userid
authtokenattributes	`UserIdentity.VO` and child structures	`UserIdentity.Group` and `UserIdentity.GroupAttribute`	contains the attributes of auth token (VOMS FAQNs in currect implementation)
projectname	`ProjectName`	`UserIdentity.GroupAttribute`	User-defined name of the project the job belongs to
status	`Status`	`Status`	The terminal state of an A-REX job: aborted, failed, completed
exitcode	not used	`ExitStatus`	The exit code of the payload in the LRMS
submissiontime	`StartTime`	`StartTime`	The timestamp of job acceptance at A-REX
endtime	`EndTime`	`EndTime`	The timestamp when the job reached the terminal state in A-REX
nodecount	`NodeCount`	`NodeCount`	Number of allocated worker nodes
inputfile	`FileTransfers`	not used	Details of downloaded inputfile: url, size, transfer start, transfer end, downloaded from cache
outputfile	`FileTransfers`	not used	Details of uploaded outputfile: url, size, transfer start, transfer end
usedmemory	`Memory`	`Memory`	Maximum virtual memory used by the job
usedmaxresident	`Memory`	`Memory`	Maximum resident memory used by the job
usedaverageresident	`Memory`	`Memory`	To be dropped from the AAR schema
usedwalltime	`WallDuration`	`WallDuration`	The measured clocktime ellapsed during the execution of the job in the LRMS. No matter on how many cores, processors, nodes the user job ran on.

continues on next page

Table 5.2 – continued from previous page

A-REX Accounting Record (AAR)	SGAS OGF-UR	APEL CAR	Content description
usedcputime	`CpuDuration`	`CpuDuration` (with type `all`)	The total CPU time consumed by the job. If the job ran on many cores/processors/nodes, all separate consumptions shall be aggregated in this value.
usedusercputime	`CpuDuration` (with type `user` but should not be there)	`CpuDuration` (with type `user`)	The user part of the usedcputime
usedkernelcputime	`CpuDuration` (with type `system` but should not be there)	`CpuDuration` (with type `system`)	The kernel part of the usedcputime
cores	`Processors`	`Processors`	The number of cores allocated to the job
usedscratchspace	`StorageUsageBlock`		The used size of scratch dir at the end of the job termination in the LRMS.
systemsoftware			The type and version of the system software (i.e. opsys, glibc, compiler, or the entire container wrapping the system software)
wninstance	`ServiceLevel`		Coarse-grain characterization tag for the WorkerNode, e.g. BigMemory or t2.micro (aka Amazon instance type)
RTEs			List of used RTEs, including default ones as well.
data-stagein-volume	`Network` class has something similar		The total volume of downloaded job input data in GBs
data-stagein-time			The time spent by the DTR sysem to download input data for the job
data-stageout-volume	`Network` class has something similar		The total volume of uploaded job output data in GBs
data-stageout-time			The time spent by the DTR sysem to upload output data of the job
lrms-submission-time			The timestamp when the job was handed over to the LRMS system
lrmstarttime			The timestamp when the payload starts in the LRMS
lrmsendtime			The timestamp when the payload completed in the LRMS

continues on next page

Table 5.2 – continued from previous page

A-REX Accounting Record (AAR)	SGAS OGF-UR	APEL CAR	Content description
benchmark	`Benchmark`	`ServiceLevel`	The type and the corresponding benchmark value of the assigned WN

Table 5.3: NOT USED SGAS or APEL attributes

SGAS OGF-UR	APEL CAR
`ProcessID`	
`Charge`	`Charge`
`Swap`	`Swap`

5.3 ARC CE REST interface specification

Note: The current interface version is 1.0

Warning: This is a new REST Intreface specification which implementation is under development! An earlier version of an ARC CE REST Interface implementation called the "The REST technical preview" released with ARC 6.x series does not corespond to this specification.

5.3.1 The REST API endpoint

The various functionalities of the service are accessible through HTTP(S) URL built upon following pattern:

<service endpoint URL>/rest/<version>/<functionality>

- `<service endpoint URL>` represents mounting point of the service and may look like `https://arc.example.org:443/arex`.
- `<version>` is two parts number separated by dot. Current version is `1.0`.
- `<functionality>` is one of keywords defined below.

Further the part `<service endpoint URL>/rest/<version>` is referred as `<base URL>`.

All parts of URL to the right of hostname are case-sensitive.

Depending on `Accept` header in HTTP request (`Accept: application/json`, `Accept: text/xml` or `Accept: application/xml`), information in the response rendered in either JSON or XML format. If not specified it defaults to `text/html` and output is compatible with ordinary web browser.

In the HTTP response headers the HTTP Status-Code (RFC7231) indicates the status of the overal request (e.g. 403 corresponds to the forbidden).

For the operations that support multiple (bulk) requests per single API call, in addition to the Status-Code in HTTP header, the per-request Status-Codes are returned. They are included as a part of the response array in HTTP body using the same RFC2731 values following the syntax defined below.

5.3.2 Description of functionalities and operations

Requesting supported versions

GET <service endpoint URL>/rest

Operations:

- GET - returns list of supported REST API versions
- POST, PUT, DELETE - not supported

Example response:

The XML response is like:

```
<versions>
  <version>1.0</version>
  <version>1.1</version>
  <version>1.2</version>
</versions>
```

The JSON is:

```
{version: [ "1.0", "1.1", "1.2" ]}
```

or

```
{version: "1.0"}
```

Obtaining CE resource information

GET <base URL>/info[?schema=glue2]

Operations:

- GET - retrieve generic information about cluster properties. It accepts the optional schema parameter with the following values: glue2, crr. By the default the information is served as a GLUE2 document. The CRR rendering will be added in future ARC releases. XML or JSON returned according to request headers.
- HEAD - supported
- PUT, POST, DELETE - not supported.

Example QUERY:

```
GET https://host.domain.org:443/arex/rest/1.0/info?schema=glue2 HTTP/1.1
Accept: application/xml
```

The XML response is:

```
<InfoRoot>
  <Domains xmlns="http://schemas.ogf.org/glue/2009/03/spec_2.0_r1"
↪xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation=
↪"https://raw.github.com/OGF-GLUE/XSD/master/schema/GLUE2.xsd">
    <AdminDomain BaseType="Domain" CreationTime="2018-11-06T20:26:46Z"
↪Validity="10800">
      <ID>urn:ad:UNDEFINEDVALUE</ID>
      <Name>UNDEFINEDVALUE</Name>
      <Distributed>false</Distributed>
      <Services>
        <ComputingService BaseType="Service" CreationTime="2018-11-
```

(continues on next page)

(continued from previous page)

```
↪06T20:26:46Z" Validity="10800">
          <ID>urn:ogf:ComputingService:arc.zero:arex</ID>
          <Capability>data.transfer.cepush.srm</Capability>
          <Capability>executionmanagement.jobmanager</Capability>
 ... output omitted ...
```

Operating jobs

GET <base URL>/jobs[?state=<state1>[,<state2>[. . .]]]

POST <base URL>/jobs?action=new

POST <base URL>/jobs?action={info|status|kill|clean|restart|delegations}

Operations:

- `GET` - get list of jobs
- `HEAD` - supported
- `POST` - job submission and management
- `PUT`, `DELETE` - not supported

Get list of jobs

`GET <base URL>/jobs` retrieves list of jobs belonging to authenticated user as `application/xml` or `application/json`. Returned document contains list of job IDs.

It accepts the optional `state` parameters. When defined the returned document contains only jobs in the *requested state(s)*.

Example QUERY:

```
GET https://host.domain.org:443/arex/rest/1.0/jobs HTTP/1.1
Accept: application/xml
```

The XML response is:

```
<jobs>
  <job>
    <id>1234567890abcdef</id>
  </job>
  <job>
    <id>fedcba0987654321</id>
  </job>
</jobs>
```

The JSON is:

```
{
  "job":[
    {"id":"1234567890abcdef"},
    {"id":"fedcba0987654321"}
  ]
}
```

Job submission (create a new job)

`POST <base URL>/jobs?action=new` initiates creation of a new job instance or multiple jobs.

Request body contains job description(s), in one of the supported formats: ADL as `Content-type: application/xml` or XRSL as `Content-type: applicaton/rsl`.

To pass multiple job descriptions in document body of the same type:

- ADL descriptions are enclosed in `<ActivityDescriptions>` element
- XRSL uses + to *merge multiple jobs*.

Response contains 201 code. Response body contains an array of elements corresponding to the sequence of the job descriptions in the requests in the same order. The elemenets of the array in the response contains:

- `status-code`: a 3-digit integer result code of the attempt to understand and satisfy the request (according to RFC7231)
- `reason`: a short textual description of the Status-Code
- `id`: job UUID or None if not assigned (non-successfull submission)
- `state`: the job state according to *state model* or None if not available (non-successfull submission)

  The XML response is:

```xml
<jobs>
  <job>
    <status-code>201</status-code>
    <reason>Created</reason>
    <id>1234567890abcdef</id>
    <state>ACCEPTING</state>
  </job>
  <job>
    <status-code>500</status-code>
    <reason>Requested RTE is missing</reason>
  </job>
</jobs>
```

  The JSON is:

```json
{
  "job":[
    {
      "status-code":"201",
      "reason":"Created",
      "id":"1234567890abcdef",
      "state":"ACCEPTING"
    },
    {
      "status-code":"500",
      "reason":"Requested RTE is missing",
    }
  ]
}
```

Jobs management

`POST <base URL>/jobs?action={info|status|kill|clean|restart|delegations}` - job management operations supporting arrays of jobs.

Request body contains list of jobids as JSON/XML (e.g. output of `GET <base URL>/jobs` can be reused).

Example of the body in XML:

```xml
<jobs>
  <job>
    <id>1234567890abcdef</id>
  </job>
  <job>
    <id>fedcba0987654321</id>
  </job>
</jobs>
```

And in JSON:

```json
{
  "job":[
    {"id":"1234567890abcdef"},
    {"id":"fedcba0987654321"}
  ]
}
```

Response depends on the requested action:

Job info
: `POST <base URL>/jobs?action=info` retrieves full information about job(s) according to the GLUE2 activity information XML document, or in JSON format.

Response contains 201 code. Response body contains an array of elements corresponding to the job IDs in the requests. The elemenets of the array in the response contains:

- `status-code`: a 3-digit integer result code of the attempt to understand and satisfy the request (according to RFC7231). The 200 is only positive response.
- `reason`: a short textual description of the Status-Code
- `id`: job UUID
- `info_document`: GLUE2 activity information about the job or empty documents if not available (request if not satisfiable)

Job status
: `POST <base URL>/jobs?action=status` retrieves information about job(s) current state.

Response body contains an array of elements corresponding to the job IDs in the requests. The elemenets of the array in the response contains:

- `status-code`: a 3-digit integer result code of the attempt to understand and satisfy the request (according to RFC7231). The 200 is only positive response.
- `reason`: a short textual description of the Status-Code
- `id`: job UUID
- `state`: the job state according to *state model* or None if not available

Killing jobs
: `POST <base URL>/jobs?action=kill` send a request to kill job(s).

Response body contains an array of elements corresponding to the job IDs in the requests. The elemenets of the array in the response contains:

- `status-code`: a 3-digit integer result code of the attempt to understand and satisfy the request (according to RFC7231). The response code is 202 to indicate request is queued for later execution and is only positive response.
- `reason`: a short textual description of the Status-Code
- `id`: job UUID

Clean job files
`POST <base URL>/jobs?action=clean` send a request to clean job(s) files.

Response body contains an array of elements corresponding to the job IDs in the requests. The elemenets of the array in the response contains:

- `status-code`: a 3-digit integer result code of the attempt to understand and satisfy the request (according to RFC7231). The response code is 202 to indicate request is queued for later execution and is only positive response.
- `reason`: a short textual description of the Status-Code
- `id`: job UUID

Restart job
`POST <base URL>/jobs?action=restart` send a request to restart job(s).

Response body contains an array of elements corresponding to the job IDs in the requests. The elemenets of the array in the response contains:

- `status-code`: a 3-digit integer result code of the attempt to understand and satisfy the request (according to RFC7231). The response code is 202 to indicate request is queued for later execution.
- `reason`: a short textual description of the Status-Code
- `id`: job UUID

Job delegations
`POST <base URL>/jobs?action=delegations` - retrieves list of delegations associated with the job.

Response body contains an array of elements corresponding to the job IDs in the requests. The elemenets of the array in the response contains:

- `status-code`: a 3-digit integer result code of the attempt to understand and satisfy the request (according to RFC7231), 200 is only positive response
- `reason`: a short textual description of the Status-Code
- `id`: job UUID
- `delegation_id`: an array of assigned delegation IDs

File operations

Files belonging to specific job are operated using `<base URL>/jobs/<job id>` URL.

Working with session directory

GET <base URL>/jobs/<job id>/session/<path>

DELETE <base URL>/jobs/<job id>/session/<path>

PUT <base URL>/jobs/<job id>/session/<path>

Operations:

- `GET`, `HEAD`, `PUT`, `DELETE` - supported for files stored in job's session directory and perform usual actions.
- `GET`, `HEAD` - for directories retrieves list of stored files (consider WebDAV for format)

- DELETE - for directories removes whole directory
- PUT - for directory not supported.
- POST - not supported.

Delegation functionality

GET <base URL>/delegations

POST <base URL>/delegations?action=new

Operations:

- GET - retrieves list of delegations belonging to authenticated user
- HEAD - supported
- POST - create new delegation
- PUT, DELETE - not supported

POST <base URL>/delegations/<delegation id>?action=get,renew,delete

PUT <base URL>/delegations/<delegation id>

Operations:

- GET, HEAD - not supported
- POST - manage particular delegation ID
- PUT - store delegation public part for particular delegation ID

Get list of delegations

`GET <base URL>/delegations` - retrieves list of delegations belonging to authenticated user

QUERY:

```
GET https://host.domain.org:443/arex/rest/1.0/delegations HTTP/1.1
Accept: application/xml
```

The XML response is:

```
<delegations>
  <delegation>
    <id>1234567890abcdef</id>
  </delegation>
  <delegation>
    <id>fedcba0987654321</id>
  </delegation>
</delegations>
```

The JSON formatted response (make consistent across specification):

```
{
  delegation: [
    { "id":"1234567890abcdef"},
    { "id":"fedcba0987654321"}
  ]
}
```

New delegation

Delegation is a 2-step process. Step 1 generates pair of private/public keys on server side and communicates X.509 certificate request to the client. Client sings public key and stores delegated certificate to finish delegation procedure.

1 step
: `POST <base URL>/delegations?action=new` starts a new delegation process (1st step). Response is 201 and contains certificate request of `application/x-pem-file` type and URL of delegation in `Location` HTTP header with assigned delegation id.

2 step
: `PUT <base URL>/delegations/<delegation id>` stores public part (2nd step). Request body contains signed certificate (`Content-type: application/x-pem-file`). Response is 200 on success.

Delegations management

Delegations are managed one-by-one. The same delegation ID can be re-used for multiple jobs (submitted separately or in batch).

The delegation ID to be used in the job context required to be explicitely specified as a part of the job description in a description language defined way (e.g. `DelegationID` in ADL).

`POST <base URL>/delegations/<delegation id>?action=get,renew,delete` used to manage delegation.

Request body is empty and action is defined by `action` value.

Response is structured depending on the action:

Get delegation
: `POST <base URL>/delegations/<delegation id>?action=get` returns public part of the stored delegation as `application/x-pem-file`

Renew delegation
: `POST <base URL>/delegations/<delegation id>?action=renew` initiates renewal of delegation. Response is 200 with certificate request of `application/x-pem-file` type.

Delete delegation
: `POST <base URL>/delegations/<delegation id>?action=delete` removes delegation. Response is 200 with no body expected.

A-REX control directory files access for debugging purposes

GET <base URL>/jobs/<job id>/diagnose/<file type>

Operations:

- `GET` - return the content of file in A-REX control directory for requested jobID
- `HEAD` - supported
- `POST`, `PUT`, `DELETE` - not supported

The `<file type>` matches the controldir file suffix and can be one of the following:

- failed
- local
- errors
- description
- diag

- comment
- status
- acl
- xml
- input
- output
- input_status
- output_status
- statistics

5.3.3 REST Interface Job States

Table 5.4: State identifiers used with ARC REST API

REST API State Name	Description	A-REX Internal State
ACCEPTING	This is the initial job state. The job has reached the cluster, a session directory was created, the submission client can optionally upload files to the sessiondir. The job waits to be detected by the A-REX, the job processing on the CE hasn't started yet	ACCEPTED
ACCEPTED	In the **ACCEPTED** state the newly created job has been detected by A-REX but can't go to the next state due to an internal A-REX limit. The submission client can optionally upload files to the sessiondir.	PENDING:ACCEPTED
PREPARING	The job is undergoing the data stage-in process, input data is being gathered into the session directory (via external downloads or making cached copies available). During this state the submission client still can upload files to the session directory. *This is an I/O heavy job state*.	PREPARING
PREPARED	The job successfully completed the data stage-in process and is being held waiting in A-REX's internal queue before it can be passed over to the batch system	PENDING:PREPARING
SUBMITTING	The job environment (via using RTEs) and the job batch submission script is being prepared to be followed by the submission to the batch system via using the available batch submission client interface	SUBMIT
QUEUING	The job is under the control of the local batch system and is "queuing in the batch system", waiting for a node/available slot	INLRMS
RUNNING	The job is under the control of the local batch system and is "running in the batch system", executing on an allocated node under the control of the batch system	INLRMS
HELD	The job is under the control of the local batch system and is being put on hold or being suspended, for some reason the job is in a "pending state" of the batch system	INLRMS
EXITINGLRMS	The job is under the control of the local batch system and is finishing its execution on the worker node, the job is "exiting" from the batch system either because the job is completed or because it was terminated	INLRMS
OTHER	The job is under the control of the local batch system and is in some "other" native batch system state which can not be mapped to any of the previously described batch systems states.	INLRMS
EXECUTED	The job has successfully completed in the batch system. The job is waiting to be picked up by the A-REX for further processing or waiting for an available data stage-out slot.	PENDING:INLRMS
FINISHING	The job is undergoing the data stage-out process, A-REX is moving output data to the specified output file locations, the session directory is being cleaned up. Note that failed or terminated jobs can also undergo the **FINISHING** state. *This is an I/O heavy job state*	FINISHING
FINISHED	Successful completion of the job on the cluster. The job has finished ALL its activity on the cluster AND no errors occurred during the job's lifetime.	FINISHED
FAILED	Unsuccessful completion of the job. The job failed during one of the processing stages. The job has finished ALL its activity on the cluster and there occurred some problems during the lifetime of the job.	FINISHED
KILLING	The job was requested to be terminated by an authorized user and as a result it is being killed. A-REX is terminating any active process related to the job, e.g. it interacts with the LRMS by running the job-cancel script or stops data staging processes. Once the job has finished ALL its activity on the cluster it will be moved to the **KILLED** state.	CANCELLING
KILLED	The job was terminated as a result of an authorized user request. The job has finished ALL its activity on the cluster.	FINISHED
WIPED	The generated result of jobs are kept available in the session directory on the cluster for a while after the job reaches its final state (**FINISHED**, **FAILED** or **KILLED**). Later, the job's session directory and most of the job related data are going to be deleted from the cluster when an expiration time is exceeded. Jobs with expired session directory lifetime are "deleted" from the cluster in the sense that only a minimal set of info is kept about such a job and their state is changed to **WIPED**	DELETED

5.3.4 Status of This Document

This document provides normative specificsation for the ARC REST Interface version 1.0.

Note that during the testing of the implementation the exact rendering of the responses (especially JSON) might be adjusted.

In order to complement the finalization of this specification, the following actions need to be completed:

- provide more detailed request/response examples
- DelegationID should be added to xRSL. Per-url delegation ID can be specified as on URL-option. Job-wide delegation id requires new `delegationId` option

This specification was designed by the requirements listed below:

1. Support for versioning: via URL paths like https://arc.zero:443/arex/rest/1.0/jobs
2. Usable with simple tools (wget, curl)
3. Friendly to common HTTP REST frameworks
4. Interactive access to session directory content
5. Machine readable error/result codes/messages
6. No drastic changes to information representation and jobs handling
7. Support for different response formats: xml, json

Plans for functionality extension post version 1.0:

1. More effective bulk operations: with HTTP v2, will require HTTP v2 development for HED, this feature is postponed till next versions
2. Resource information functionality: consider filtering through URL options, consider supporting references (relative URLs) to underlying resources.
3. Scalability for many jobs and delegations: consider filtering through URL options
4. Jobs: consider a way to provide list of all jobs per site or per VO to special monitoring agents
5. Add `hold` action for jobs management once it will be implemented
6. For sessiondir access add `PATCH` for files to modifies part of files. Body format need to be defined, all files treated as binary, currently support only non-standard `PUT` with ranges.

5.4 ARCHERY data model and DNS records rendering

ARCHERY implies a minimalistic data model that is targeting mostly static information to address the distributed e-Infrastructure resource grouping, software environments provisioning and service discovery needs.

The ARCHERY objects and their releations are embedded into the DNS infrastructure as a dedicated Resource Record sets according to the rendering described below.

5.4.1 ARCHERY objects

To represent the distributed e-infrastructure concept the ARCHERY data model defines three type of objects: the **Endpoint Object**, the **Service Object** and the **Group Object**.

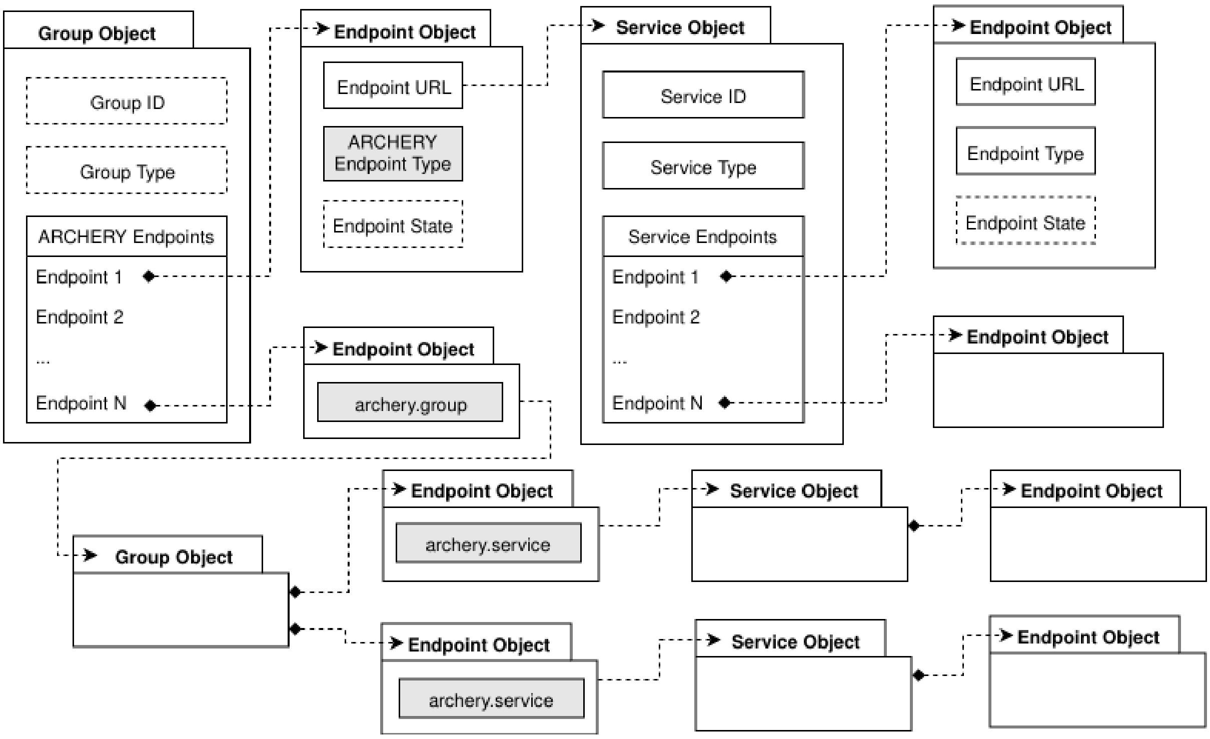


Fig. 5.13: ARCHERY e-Infrastructure topology bjects, their attributes and relations in the data model.

To represent the *community software environments* in the e-infrastructure two additional type of objects are defined in the ARCHERY data model: the **Software Object** and the **RTE Object**.

Endpoint object

Endpoint object is used to capture information about a network location that can be used to access specific service functionality, including accessing information within ARCHERY registry service itself. For the later special endpoint types were defined (see below). The Endpoint object is described with the following set of attributes:

- **Endpoint URL** - defines the network location by which the service functionality is accessible;
- **Endpoint Type** - contains the value from the Endpoint type enumeration defined by the e-Infrastructure operators[1]. In addition to ordinary service endpoint types two special ARCHERY endpoint types `archery.group` and `archery.service` had been introduced for accessing the ARCHERY Group and Service objects within the registry. Another special types includes `archery.software` and `archery.rte` that used to access ARCHERY Software and RTE objects respectively when community software registry is embedded into the DNS.
- **Endpoint State (optional)** - Boolean value that indicates the endpoint availability. In case of an ARCHERY service endpoint, if the endpoint state is false (unavailable) it is still used to represent e-Infrastructure topology, but should not be used during service endpoints discovery. Missing attribute is interpreted as true i.e. the endpoint is available.

[1] The `org.ogf.glue.emies.activiticreation`, `org.nordugrid.ldapglue2`, `org.nordugrid.gridftpd` are examples of endpoint types used in NorduGrid infrastructure.

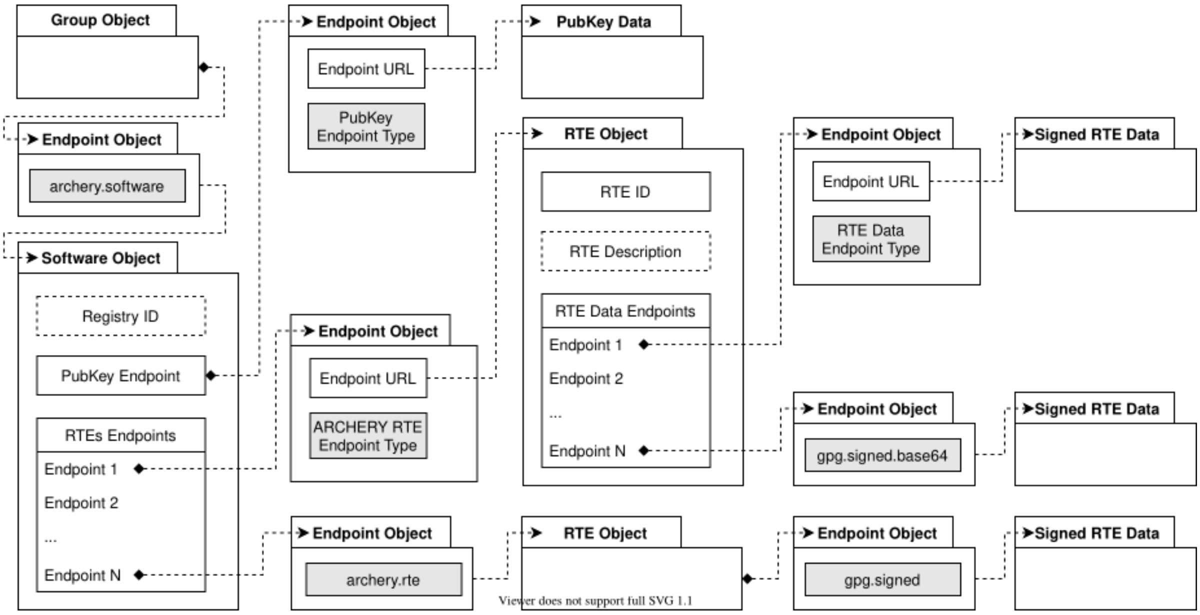


Fig. 5.14: ARCHERY software environment objects, their attributes and relations in the data model.

Service object

Service object represents an e-Infrastructure service, like Computing Element, Storage Element, etc. The Service object is described with the following set of attributes:

- **Service ID** - holds the service identifier as an arbitrary string. For example the ID can be derived from the hostname;
- **Service Type** - contains the value from the Service type enumeration defined by the e-Infrastructure operators;
- **Service Endpoints** - an array of Endpoint objects associated with the Service.

Group object

Group object is used to organize other objects such as ARCHERY Service or Group. The grouping was introduced to the data model so that infrastructure topologies (i.e. hierarchies or federations) can be represented inside the registry in a flexible way. The Group object is described with the following set of attributes:

- **Group ID (optional)** - holds the group identifier as an arbitrary string.
- **Group Type (optional)** - defines the grouping type based on organizational structure; example values could be: Site, Country, Tier, etc.
- **ARCHERY Endpoints** - an array of Endpoint objects of special defined ARCHERY endpoint types (`archery.group`, `archery.service` or `archery.software`). These Endpoint objects describe the optional state and URL of the ARCHERY objects inside the registry.

Software object

Software objects are used to represent a *community-defined RTEs* regisry instances. The Software object provides community public key data and enumerates the available RTEs. It is described with the following set of attributes:

- **Registry ID (optional)** - holds the software registry identifier as an arbitrary string.
- **PubKey Endpoint** - and Endpoint object that contains URL to the public key location. It can points to external URLs or raw data inside the DNS. Technical implementation uses `gpg.pubkey` and `gpg.pubkey.base64` endpoint types depending on Base64 encoding usage.
- **RTEs Endpoints** - and array of Endpoint objects of special defined ARCHERY endpoint type `archery.rte`. These Endpoint objects describe the optional state and URL of the ARCHERY RTE objects inside the registry.

RTE Object

RTE Object is used to represent a particular *community-defined RTE script*. It encapsulates the RTE ID (according to the naming and versioning convention), optional RTE description and contains Endpoints to signed RTE scripts. It is described with the following set of attributes:

- **RTE ID** - holds the RTE identifier according to naming and versioning scheme
- **RTE Description (optional)** - an arbitrary string that contains the human-readable description of RTE
- **RTE Data Endpoints** - and array of Endpoint objects that points to signed RTE data. It can points to external URLs or raw data inside the DNS. Technical implementation uses `gpg.signed` and `gpg.signed.base64` endpoint types depending on Base64 encoding usage.

Entry point group object

Following the data model the service endpoint discovery process can start from the **entry point** Group Object and recursively contacting all the ARCHERY Endpoints.

The Sotfware objects can be attached to any parent Group Object and discovered using the same recursive traversal of the e-Infrastructure topology.

ARCHERY object attributes allow taking into account group, service and/or endpoint types as well as availability status during the recursive discovery process to query only the subset of objects.

5.4.2 DNS Resource Records rendering

Rendering ARCHERY objects

ARCHERY objects are rendered using `TXT` resource records (RRs). These `TXT` RRs contain the space separated `key=value` pairs where keys correspond to the data model object attributes. Boolean object attributes have values specified as 0 or 1.

The **Endpoint Object** is rendered with a single `TXT` RR where Endpoint Object attributes – *Endpoint URL*, *Endpoint Type* and optional *Endpoint State* are represented by the `u`, `t`, `s` keys in this specific order within the space separated key-value pairs.

The **Service Object** of the ARCHERY data model is rendered by a dedicated RRSet identified by unique domain name that can be used to access this object inside the registry.

The service RRSet is composed of single service object identity RR and one RR for every Endpoint objects associated with the service. The service object identity RR has the format of `o=service` followed by `t=<Service Type>` and `id=<Service ID>`:

```
<DNS name> TXT "o=service t=<Service Type> id=<Service ID>"
<DNS name> TXT "u=<Endpoint URL> t=<Endpoint Type> [s={0|1}]"
<DNS name> TXT                   ...
<DNS name> TXT "u=<Endpoint URL> t=<Endpoint Type> [s={0|1}]"
```

The **Group Object** of is rendered by a dedicated RRSet identified by unique domain name that can be used to access this object inside the registry.

The Group RRSet is composed of single Group object identity RR and one RR for every ARCHERY Endpoint objects pointing to other Service or Group objects that are part of this specific Group. The Group object identity RR has the format of `o=group` followed by optional `t=<Group Type>` and optional `id=<Group ID>`:

```
<DNS name> TXT "o=group [t=<Group Type>] [id=<Group ID>]"
<DNS name> TXT "u=<DNS URL> t=archery.{group|service} [s={0|1}]"
<DNS name> TXT              ...
<DNS name> TXT "u=<DNS URL> t=archery.{group|service} [s={0|1}]"
```

The **Software Object** of the ARCHERY data model is rendered by a dedicated RRSet identified by unique domain name that can be used to access this object inside the registry.

The software object RRSet is composed of single object identity RR, PubKey Endpoint object RR and one RR for every RTEs Endpoint objects associated with the registry. The software object identity RR has the format of `o=software` followed by optional `id=<Registry ID>`:

```
<DNS name> TXT "o=software [id=<Registry ID>]"
<DNS name> TXT "u=<PubKey URL> t=gpg.pubkey[.base64]"
<DNS name> TXT "u=<DNS URL> t=archery.rte [s={0|1}]"
<DNS name> TXT                   ...
<DNS name> TXT "u=<DNS URL> t=archery.rte [s={0|1}]"
```

The **RTE Object** of the ARCHERY data model is rendered by a dedicated RRSet identified by unique domain name that can be used to access this object inside the registry.

The RTE object RRSet is composed of single object identity RR, and one RR for every RTE Data Endpoint object pointing to signed RTE data location. The RTE object identity RR has the format of `o=rte` followed by `id=<RTE ID>` and optional `d=<RTE Descition>`. The description should be encoded to eliminate spaces, using the URLEncode is advised. Typicaly there is only one RTE Data Endpoint object that point to either DNS or external URL. Several RTE Data Endpoints imply external URL redundancy:

```
<DNS name> TXT "o=rte id=<RTE ID> [d=<RTE Description>]"
<DNS name> TXT "u=<RTE Data URL> t=gpg.singed[.base64] [s={0|1}]"
```

Embedding ARCHERY object relations

In the registry rendering the entry point Group Object is distinguished by the predefined RRSet DNS name starting with the mandatory `_archery`. Any other objects may have arbitrary RRSet name in any DNS zone. Exact RRSets naming scheme does not affect service discovery process, but should be consistent and transparent from the operational point of view.

The child-parent Endpoint Object to Service Object relation is implicitly defined by means of grouping within the same RRSet. The same applies to "RTE Data Endpoint Object to RTE Object" and "PubKey Endpoint object to Software Object" relations.

The ARCHERY Data model allows grouping of Service, Group and Software Objects into Group objects via the intermediate Endpoint objects. This grouping in the DNS rendering is implemented by using dedicated ARCHERY Endpoint types that are contained as RR text inside the Group object's RRSet. The same applies to "RTEs Endpoint Object to Software Object" relations.

Both Group Object to Group Object, Group Object to Service Object, Group Object to Software Object and Software Object to RTE Object relations rest upon the unique DNS name of the RRSet of the pointed object.

5.5 ARC support for OIDC

5.5.1 Support level

Only tokens conforming to the WLCG profile have been tested. The current validation is not strict: the token is parsed and the signature is checked if present, but no additional requirements are imposed. Tokens are only accepted for client authentication during job submission through the EMI-ES and REST interfaces.

5.5.2 Obtaining and using tokens

The suggested way for obtaining a token is through the oidc-agent utility - https://indigo-dc.gitbook.io/oidc-agent/. Install it following the instructions for your Linux distribution.

Point your browser to https://wlcg.cloud.cnaf.infn.it/ and create an account.

Start the oidc-agent. It will print few lines of shell commands. Copy them to the command line and execute. This will set up environment variables for other oidc-* commands.

Start oidc-gen. It will guide you through the steps to register the OIDC client and create a profile for the oidc-agent. When asked to select the issuer, select `https://wlcg.cloud.cnaf.infn.it/`. When asked about the scope, write `openid profile wlcg`. You only need to run the oidc-gen once. Next time when you use the oidc-agent, you can load an already created profile with the 'oidc-add NAME_YOU_CHOOSE' command.

When the oidc-gen command tells you "To continue and approve the registered client visit the following URL in a Browser of your choice:" point your browser (which must be running on the same machine as the oidc tools do) to the adress given.

Obtain the token and store it in the BEARER_TOKEN variable:

```
export BEARER_TOKEN=`oidc-token NAME_YOU_CHOOSE`
```

If the oidc tools were installed on a different machine than your ARC client, then first obtain the token on the oidc tool machine:

```
oidc-token NAME_YOU_CHOOSE
```

Then copy the token string and on your ARC client machine do:

```
export BEARER_TOKEN=<token-string>
```

Now submit the job to an ARC CE with arcsub through the REST interface. For that, use the option '-T arcrest'. The token stored in the BEARER_TOKEN variable will be used instead of the X.509 certificate for authenticating the user to the ARC CE server. Note that arcsub still requires an X.509 proxy which will be delegated to the CE. Data staging currently will also still use the X.509 proxy credentials.

Note: You can use any other method for obtaining a WLCG compliant OIDC token. Just store it in the BEARER_TOKEN variable before calling arcsub.

5.5.3 Configuring authorization on server

Token processing is enabled by the presence of the [authtokens] configuration block.

The user can be authorized on the server by adding a dedicated command to the authgroup block:

```
authtokens=subject issuer audience scope group
```

The specified parameters must match those in the provided token. Parameters can be '*' to match any value, for example

```
authtokens=e83eec5a-e2e3-43c6-bb67-df8f5ec3e8d0 https://wlcg.cloud.cnaf.infn.it/ * * *
```

matches a user with subject e83eec5a-e2e3-43c6-bb67-df8f5ec3e8d0 in token issued by https://wlcg.cloud.cnaf.infn.it/ .

Note: Until handling of authtokens is integrated with arcproxy you will have to find the subject of the token using a tool like e.g. https://jwt.io/ . Alternatively you can install flaat (https://pypi.org/project/flaat/).

A full example configuration could look like

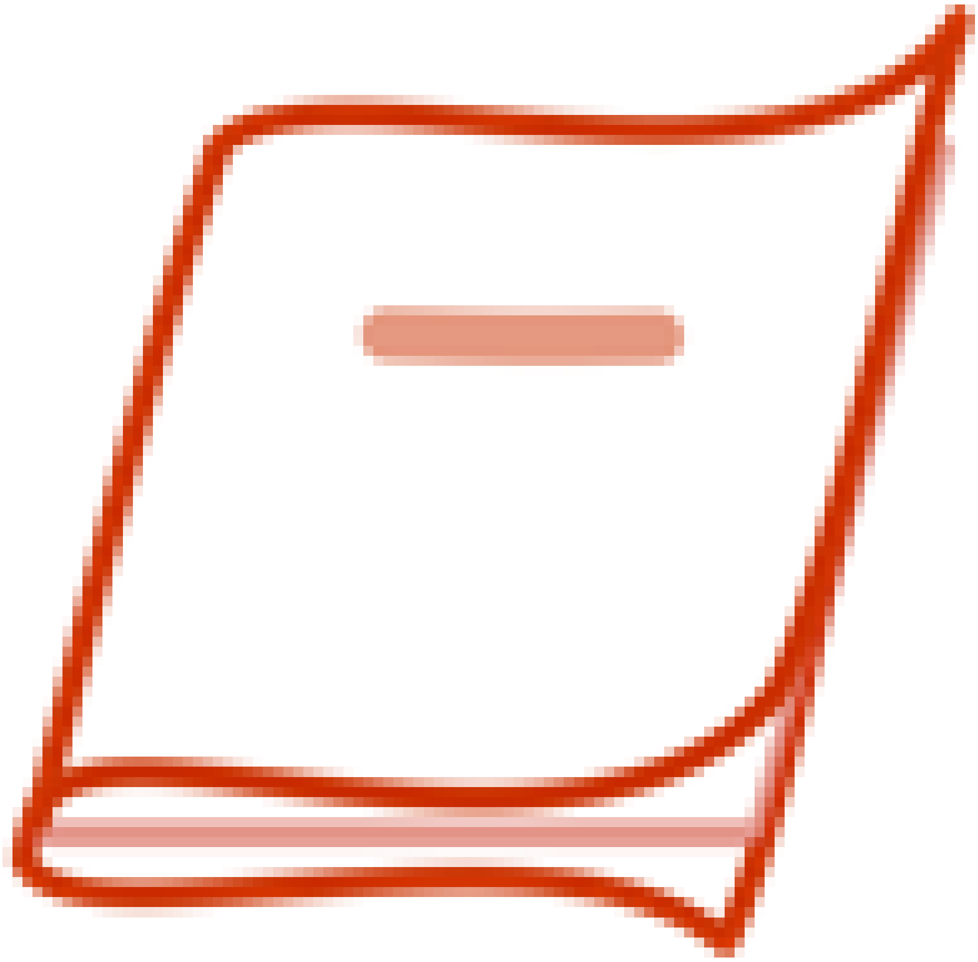
```
[authtokens]

[authgroup: wlcg_iam]
authtokens = * https://wlcg.cloud.cnaf.infn.it/ * compute.create /wlcg/pilots
authtokens = * https://wlcg.cloud.cnaf.infn.it/ * compute.read /wlcg/pilots
authtokens = * https://wlcg.cloud.cnaf.infn.it/ * compute.modify /wlcg/pilots
authtokens = * https://wlcg.cloud.cnaf.infn.it/ * compute.cancel /wlcg/pilots

[mapping]
map_to_user = wlcg_iam wlcg:wlcg
policy_on_nomap=stop

[arex/ws/jobs]
allowaccess=wlcg_iam
```

User mapping to a local account is implemented using a simulated X.509 user subject. The subject provided by an OIDC token is unique only in scope of the identity provider. To generate a globally unique user-identifier, the issuer and the subject are cocatenated as "issuer/subject" to provide an identifier suitable for user mapping. For example, a user with subject e83eec5a-e2e3-43c6-bb67-df8f5ec3e8d0 in the token issued by https://wlcg.cloud.cnaf.infn.it/ is represented by a simulated identifier https://wlcg.cloud.cnaf.infn.it//e83eec5a-e2e3-43c6-bb67-df8f5ec3e8d0

5.6 Old Relevant Technical Documents

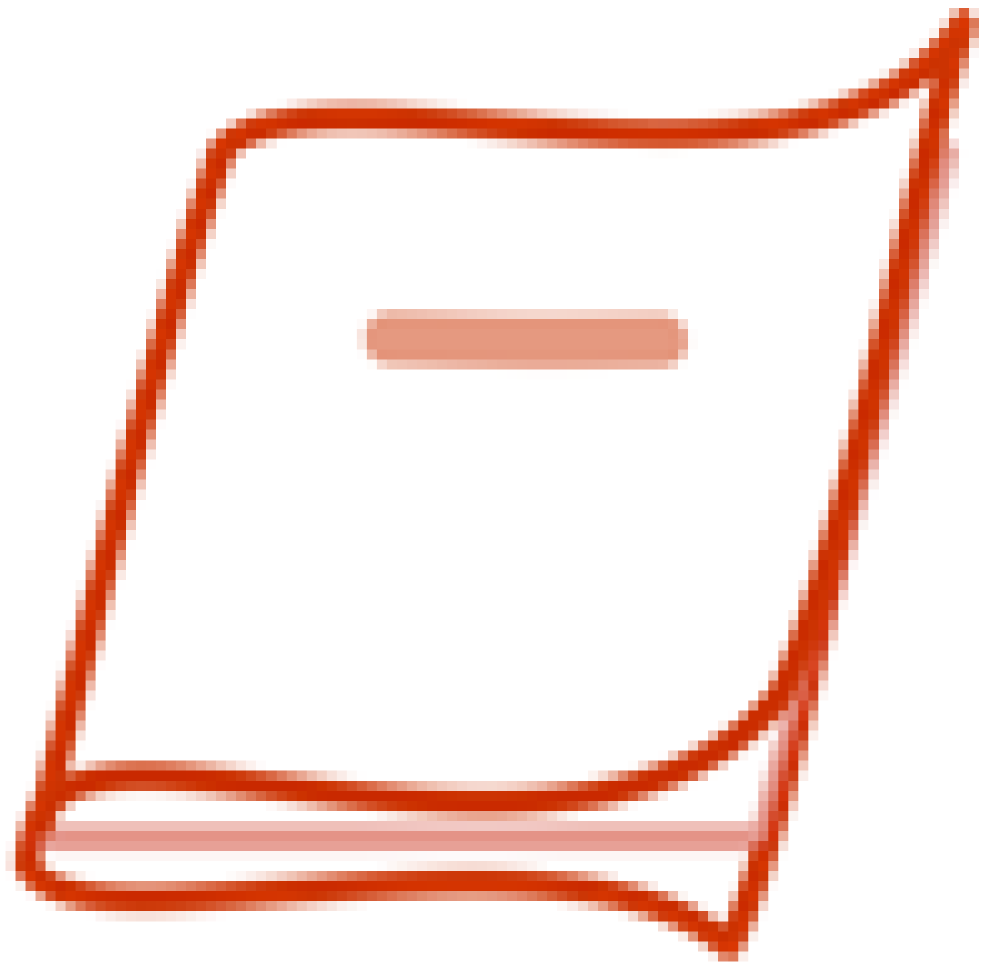
Note: Many of the technical documents exists for ARC5 only. Those that are verified to be relevant for ARC6 will be listed below

5.6.1 Hosting Environment of the Advanced Resource Connector middleware



Document gives a deep technical description of the HED service container.

5.6.2 A Client Library for ARC



Document describes from a technical viewpoint the plugin-based client library of ARC.

5.7 Legacy JURA Accounting Technical Details

WARNING: This component was deprecated in ARC 6.4 and completely removed in ARC 6.8!

> **Warning:** Information in **this chapter is relevant only for 6.0-6.3 ARC releases**.
>
> Starting from ARC 6.4 release the *next generation accounting subsystem* with local accounting database will be introduced. Make sure you are reading the documentation that match your ARC CE release version.

General accounting configuration and operations flows are described in *Accounting with legacy JURA*. This section contains more technical details about implementation of each component of accounting subsystem.

5.7.1 Records processing and publishing

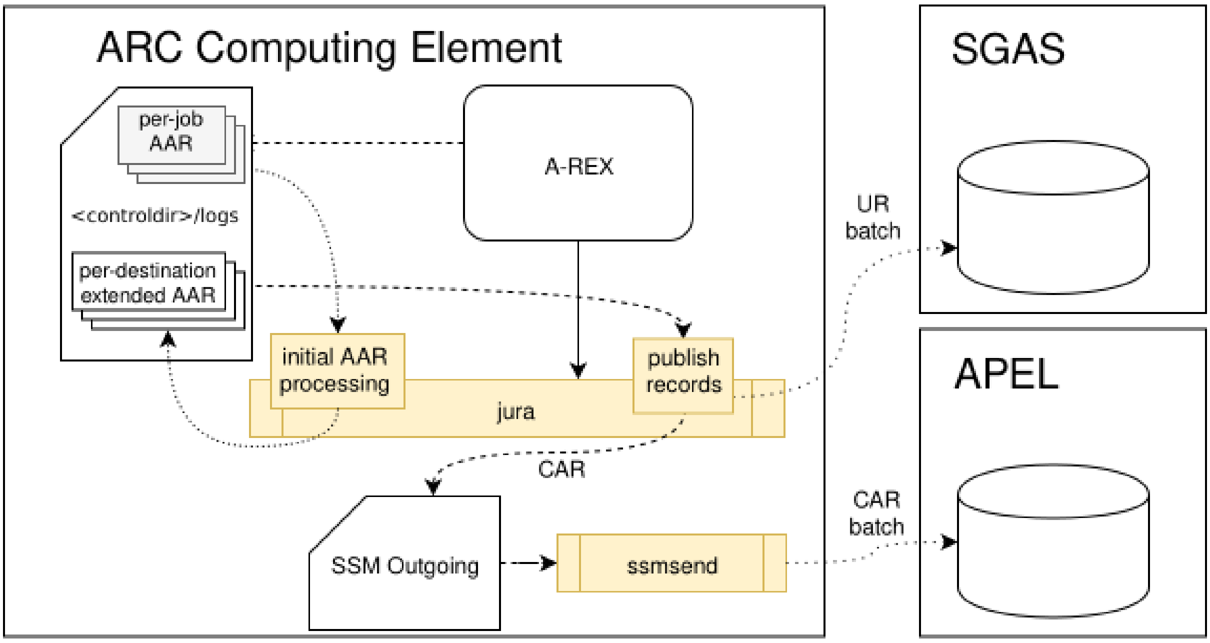


Fig. 5.15: ARC CE accounting: records creation, processing and publishing

AREX Accounting Records (AAR job log files)

The A-REX Accounting Records (AAR) are job log files generated by A-REX. AAR is the only source of accounting information for JURA. AARs are written by A-REX based on job data available, including `.diag` files that backend scripts creates based on batch system data and/or GNU `time` utility measurements. These job log files reside under the `<control_dir>/logs` directory. The name of the AAR job log files consist of the ID of the job and a random string to avoid collision of multiple job log files for the same job: `<jobid>.<random>`.

The AAR job log file consists of `name=value` lines, where `value` is either a job-related resource consumption data or a static info like name, ID or proxy certificate.

A-REX generates at least two job log files for each job: one at the time of job submission, another one after the job finishes, and possibly others at various job events. Please note JURA makes use only one of the AREX generated files belonging to the same job: the one that corresponds to the FINISHED job event (such state is indicated by the `status={completed|failed|aborted}`).

JURA initial AAR processing

A-REX periodically runs `jura` that loop over available A-REX job log records in the `<control_dir>/logs`. JURA opens all the files and processes only those that corresponds to a FINISHED job state.

JURA converts AARs to *per-job per-destnation* extended AARs that contains the target information from `arc.conf` as well. One extended AAR is generated per accounting target. The extended AAR job log files named `<jobid>.<random>_<random2>` where first `<random>` is taken from original AAR.

The original AAR `<jobid>.<random>` file is deleted once *per-destination* extended AAR logs are created by JURA.

Note: JURA as part of the initial processing deletes all files corresponding to non-finished job states.

JURA Publihsing loop

JURA publishing subsystem loop over *extended AAR* logs in the A-REX `<control_dir>/logs` directory. JURA generates records in the Usage Record (UR) format proposed by the Open Grid Forum (OGF) for SGAS or Compute Accounting Record (CAR) XML for APEL.

The *extended AAR* job log file is deleted once record is successfully submitted, thus preventing multiple insertion of same usage records. If submission to destination fails, the extended AAR log files are kept, so another attempt is made upon a subsequent run of JURA. This mechanism will be repeated until the expiration time passes at which point the next execution of JURA removes the file without processing.

Please note that the JURA publishing loop is backward compatible with ARC 5 implementation.

Reporting to SGAS

SGAS has a simple custom web service interface loosely based on WS-ResourceProperties. JURA uses the insertion method of this interface to report URs directly using ARC HTTP client implementation. The corresponding processed extended AAR job log files are deleted after receiving a non-fault response from the service.

To increase communication efficiency JURA sends URs in batches. SGAS accepts a batch of URs in a single request. The batch is an XML element called `UsageRecords`, containing elements representing URs.

The process of handling batches is the following: JURA does not send all usage records immediately after generation, but instead collects them in a batch until reaching the maximal number of records or until running out of job log files. The maximal number of URs in a batch can be set as a *urbatchsize* configuration parameter of SGAS target.

Reporting to APEL

APEL uses the SSM framework for communication.

JURA send records to APEL by means of invoking helper `ssmsend` process that uses SSM python librarires developed by APEL.

ARC ships minimal set of SSM libraries along with A-REX binary packages to allow SSM usage. If SSM binary packages from APEL are availble for your OS (e.g. EL6), you can install this packages and they will be used instead of those shipped with ARC automatically.

JURA prepares the messages to be sent by `ssmsend` and puts them info *SSM Outgoing* directory located in the `/var/spool/arc/ssm/<destination hostname>/outgoing/00000000/`. Generated messages are XML based CAR records with file name format <`YYYYMMDDhhmmss`>.

Reporting to APEL also works with sending records in batches. The default *urbatchsize* value is set to 1000 according to APEL recommendatations.

5.7.2 Accounting archive

After records are archived by `jura` a dedicated `jura-archive-manager` process manages the archive layout and archive database.

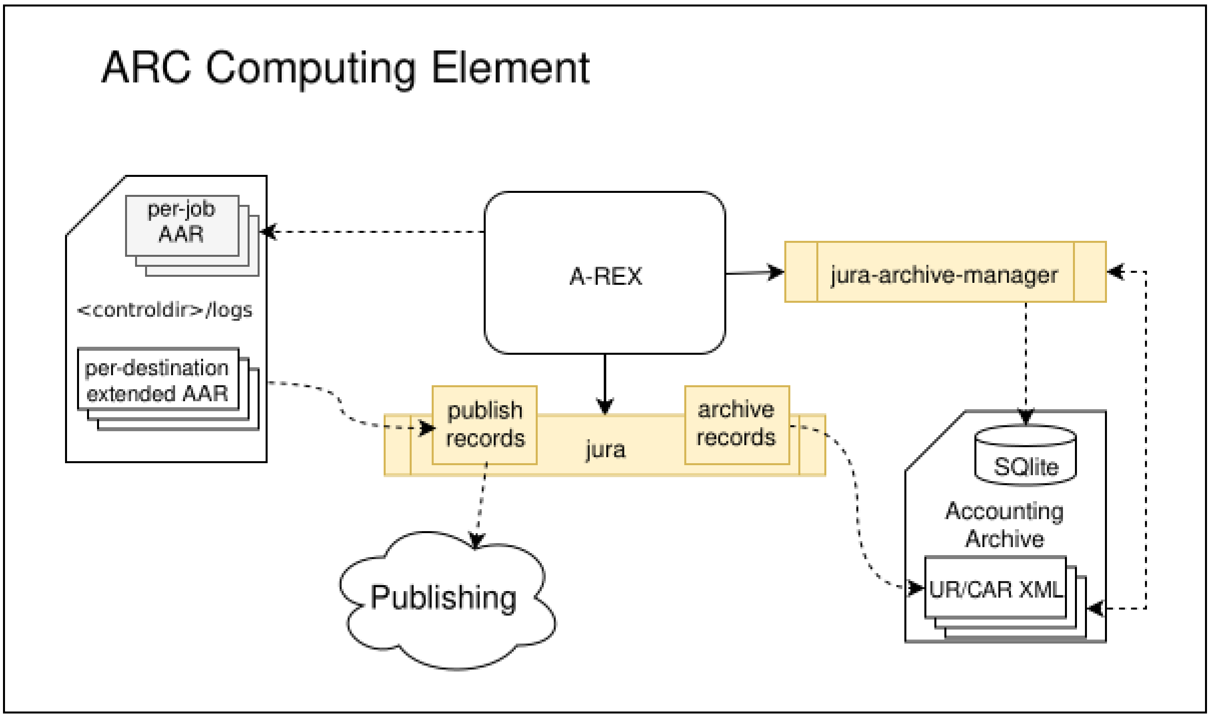


Fig. 5.16: ARC CE accounting: records archiving

5.7.3 Republishing process

Republihsing is a complex workflow that involves:

- archive database to find out which subset of records should be republished
- exporting necessary records from archive structure to temporary flat directory
- trigering `jura` with corresponding configuration to grab records from temporary directory and send them to target
- cleanup after republihsing

This chain is controlled from `arcctl` code.

5.7.4 Security

The JURA executable runs with the same user privileges as the A-REX. The owner of a job log file is the local user mapped for the submitter entity of the corresponding job. Since these files contain confidential data, A-REX restricts access to them allowing only read access for the job owner, thus when JURA is executed by A-REX it is allowed to read and delete job log files.

All usage records are submitted using the X.509 credentials specified by the value of `x509_` set of confiurartion options of `arc.conf`. No proxies are used for communcation with accouting services.

The only access restriction made by a SGAS service is matching the Distinguished Name of the client (in this context JURA) with a set of trusted DNs. When access is granted, policies are then applied by SGAS, allowing either publishing and/or querying rights. Clients with publishing right can insert any UR, regardless of content. By default, querying right only allows retrieving URs pertaining to jobs submitted by the querying entity.

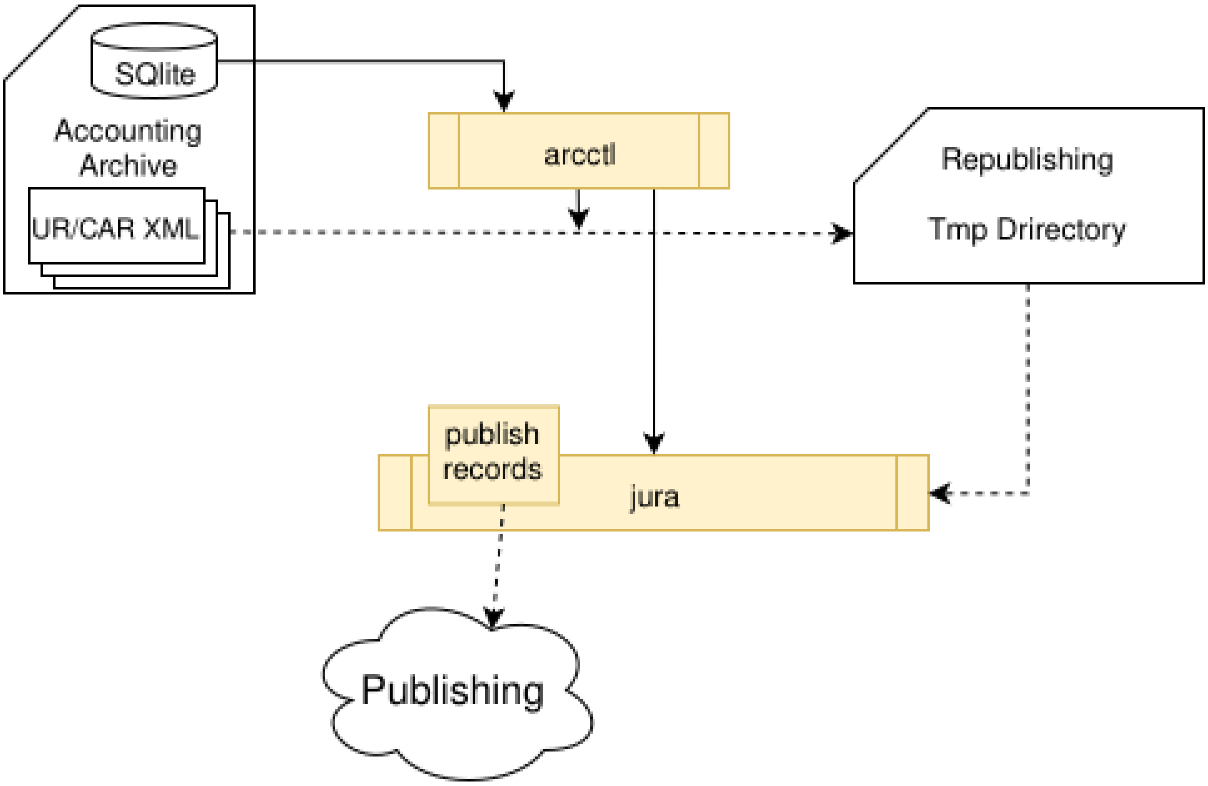


Fig. 5.17: ARC CE accounting: records republishing

5.7.5 Implementation and API

JURA as part of the ARC software stack is written in C++, and utilizes the functionality provided by the ARC libraries, including secure HTTPS communication provided by the ARC plugable TLS and HTTP modules.

The modular design is also present in the usage reporting part of the JURA code, making it possible to extend JURAs support of accounting services. To create a JURA module one should simply write a C++ class which inherits from the abstract `Arc::Destination` class, and it must extend the two methods:

- `static Arc::Destination* Arc::Destination::createDestination(Arc::JobLogFile&)`
- `void Arc::Destination::report(Arc::JobLogFile&)`

The static `createDestination` method should initialize a object of the specialized class, using the configuration options specified in the passed `Arc::LogFile` object, and the memory allocated by the method should be freed by the caller. Then the `report` method should carry out the transfer of the UR, represented by the `JobLogfile` object, to the accounting service.

JURA archive manager is written in Python and share the classes for managing ARC components with *ARC Control Tool*.

5.7.6 Limitations

In the following list some issues which limits the functionality of JURA is described:

- The current implementation of JURA and A-REX supports only one expiration time for all the reporting destinations. Even though the configuration enables the specification of different expiration values per reporting destination, it is not taken into account by the system, the last value is used as the common expiration time value.
- It is not possible to use different credentials per destinations.
- If you are updating from ARC5 with an old jura accounting archive already containing records, the conversion process to index the archive structure will be initiated and will cause serious system load until finished. To avoid old archive conversion, you can move records before update.
- Some *optional UR properties* are not supported.
- Memory can be reported incorrectly with buggy GNU "time" results.

5.8 ARC Accounting Database Schema

CHAPTER

SIX

DOCUMENTATION FOR DEVELOPERS

If you are looking for development internal details of ARC (like how some stuff was coded) this part of documentation is for you. Mainly for those who want to contribute to the project development, advanced troubleshooters or just interested.

6.1 Implementation Details for Developers

6.1.1 General arc.conf python configuration parser

Parsing configuration

Initial configuration parsing

The latest version of `arcconfig-parser` designed to operate with defaults file that holds default values for all possible parameters.

At the time of initial parsing the following chain is executed:

- All blocks and options are parsed from `arc.conf`
- For all blocks defined in `arc.conf` missing options are added from defaults file
- Special constructs in values are substituted (see *Special constructs can be used as values*)

Optional by design parameters that does not have default value (specified with `not set` value) are not included to the parsed configuration.

Runtime configuration

Configuration that includes both `arc.conf` and defaults config called *runtime configuration*.

In some cases it is useful to save and load runtime configuration:

- To supply C++ services (`a-rex`, `gridftpd`) with configuration that includes defaults from common place
- For repetitive operations on config to eliminate full-chain processing of each invocation

To save runtime configuration to the default location (`/var/run/arc/`):

```
arcconfig-parser --save
```

To save runtime configuration to specified location:

```
arcconfig-parser --save -r /var/run/arc/arex.arc.conf
```

To load runtime configuration instead of full-chain processing and e.g. get the value of `x509_host_key` in `[common]` block:

```
arcconfig-parser --load -b common -o x509_host_key
```

Special constructs can be used as values

Defaults includes references to another config parts to be consistent with the implied `arc.conf` structure.

The most obvious example is if `x509_host_key` not found in e.g. `[arex/jura]` block it should be taken from `[common]`.

Config parser is following this logic (especially in respect to defaults) and use special constructs to accomplish this behaviour.

Command substitutions

Configuration option values can contain the construct `$EXEC{<command>}` that substituted to the stdout of `<command>`.

For example:

```
hostname=$EXEC{hostname -f}
```

Option values substitutions

The construct `$VAR{[block]option}` can be used to substitute the values of another option value.

If option is in the same block as referencing option block name can be omitted - `$VAR{option}`.

For example:

```
x509_host_key=$VAR{[common]x509_host_key}
bdii_update_cmd=$VAR{bdii_location}/sbin/bdii-update
```

Evaluation of simple code

For limited number of cases `arc.conf` default values relies on arithmetic operations. For this purpose the `$EVAL{string}` special construct had been introduced.

For example:

```
bdii_read_timeout=$EVAL{$VAR{bdii_provider_timeout} + $VAR{[arex]infoproviders_
↪timelimit} + $VAR{[arex]wakeupperiod}}
```

Getting the configuration values

If `--option` argument is passed to `arcconfig-parser` parser returns the value of the specified option to stdout.

Without `--option` `arcconfig-parser` can be used to operate with configuration blocks:

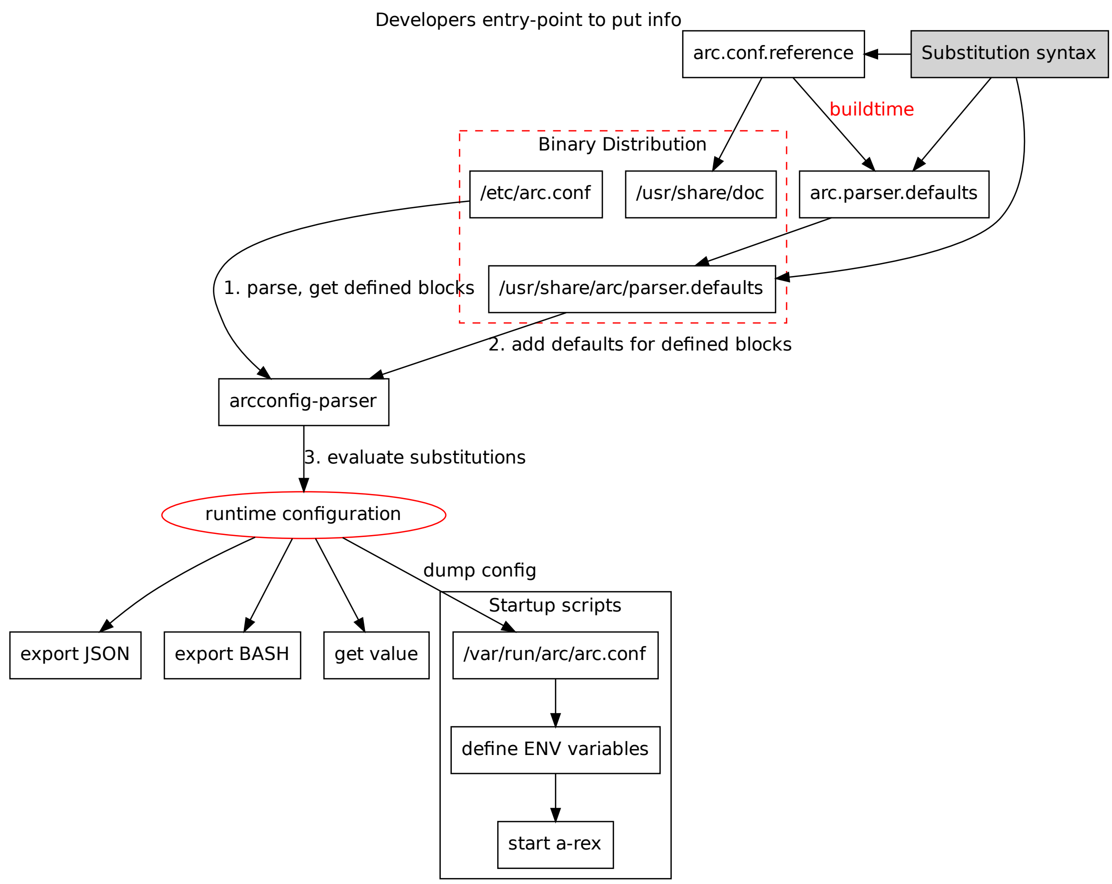
- check blocks existance (exit code used to indicate the status of the check)
- return the list of subblocks

With the `--export` option `arcconfig-parser` allows to export config in the following formats:

- `json` - returns entire configuration or subset of blocks as-is in JSON to stdout

- `bash` - for `[common]` block or specified configuration subset returns `CONFIG_option_name=value` pairs to stdout. Block names ARE NOT included in the exports and option values precedence will be used in the order of passed blocks. If automatic subblocks expansion used with bash export, for every block in sequence - it's subblocks are processed first (in `arc.conf` defined order). It is possible to filter the options that will be exported with additional `--filter` option that can be specified several times.

Common configuration parsing sequence



Options reference

The full list of available options of `arcconfig-parser` is available instantly using `--help` option. Online version is available *here*.

Examples

Get value of option in block:

```
# arcconfig-parser --block infosys --option logfile
/var/log/arc/infoprovider.log
```

Get value of option in blocks in order they are specified (e.g. if not found in `[gridftpd]` look in the `[common]` block[1]):

[1] Block dependencies are now implied by defaults file, so for most cases it is enough to specify only block in question

```
# arcconfig-parser --block gridftpd --block common --option x509_host_key
/etc/grid-security/hostkey.pem
```

Export entire configuration to JSON[2]:

```
# arcconfig-parser --export json
```

Export `[infosys]` block options to JSON (for Perl):

```
# arcconfig-parser --block infosys --export json
{"infosys": {"loglevel": "5"},...
```

Export `[infosys]` block and all their subblocks options to JSON:

```
# arcconfig-parser --block infosys --subblocks --export json
{"infosys/glue2/ldap": {"showactivities": "no"},...
```

Export for BASH (compatible with config representation in shell-based LRMS backends):

```
# arcconfig-parser --block infosys --block arex --block common --export bash
CONFIG_controldir="/var/spool/arc/jobstatus"
CONFIG_defaultttl="1210000"
CONFIG_delegationdb="sqlite"
CONFIG_hostname="ce01.example.org"
CONFIG_maaxrerun="5"
CONFIG_maxjobs="10000 -1"
CONFIG_runtimedir="/home/grid/arc/runtime"
CONFIG_sessiondir="__array__" # <= NEW define for multivalued values that indicate␣
↪indexed vars
CONFIG_sessiondir_0="/mnt/scratch/grid/arc/session"
CONFIG_sessiondir_1="/home/grid/arc/session drain"
...
```

Export for BASH with exported options filtering:

```
# arcconfig-parser -b common -f hostname -f x509_cert_dir -e bash
CONFIG_hostname="ce01.example.org"
CONFIG_x509_cert_dir="/etc/grid-security/certificates"
```

Using BASH export:

```
# eval "$( arcconfig-parser --block infosys --block arex --block common --export bash␣
↪)"
# echo "$CONFIG_gridmap"
```

Check block(s) exists (`[common/perflog]` does not exists in the example):

```
# arcconfig-parser --block common/perflog --block arex
# echo $?
1
```

List block subblocks:

```
# arcconfig-parser --block infosys --subblocks
infosys
infosys/ldap
infosys/nordugrid
```

(continues on next page)

[2] *HINT:* use `arcconfig-parser --export json | jq .` to view highlighted JSON structure in shell

(continued from previous page)

```
infosys/glue2
infosys/glue2/ldap
infosys/cluster
```

Using parser as Python module:

```
from arc.utils import config

# initial parsing with defaults
config.parse_arc_conf('/etc/arc.conf', '/usr/share/arc/arc.parser.defaults')

# get parsed dictionary and list of blocks in the arc.conf order
>>> confdict = config.get_config_dict()
>>> confblocks = config.get_config_blocks()

# get list of all [queue] subblocks sorted by name
>>> sb = config.get_subblocks(['queue'], is_sorted=True)
>>> sb
['queue:grid', 'queue:grid_rt']

# get value of 'x509_host_key' from [arex] block and than from [common] if not found␣
↪in [arex]
>>> a = config.get_value('x509_host_key', ['arex', 'common'])
>>> a
'/etc/grid-security/hostkey.pem'

# get value of 'allowactivedata' option from [gridftpd] block
>>> b = config.get_value('allowactivedata', 'gridftpd')
>>> b
'yes'

# get value of 'allowactivedata' option from [gridftpd] block (always return list)
>>> c = config.get_value('allowactivedata', 'gridftpd', force_list=True)
>>> c
['yes']

# get value of 'allowactivedata' option from [gridftpd] block (return boolean value)
>>> d = config.get_value('allowactivedata', 'gridftpd', bool_yesno=True)
>>> d
True
```

6.1.2 LRMS shell-backends overview for developers

CONFIG variables used in LRMS shell-backend:

lrms_common.sh:

```
$CONFIG_runtimedir            [arex]
$CONFIG_shared_scratch        [arex]
$CONFIG_shared_filesystem     [arex]
$CONFIG_scratchdir            [arex]
$CONFIG_gnu_time          [lrms]
$CONFIG_nodename          [lrms]
$CONFIG_enable_perflog_reporting [common]
$CONFIG_perflogdir        [common]
```

submit_common.sh:

```
$CONFIG_defaultmemory         [queue] [lrms]
$CONFIG_hostname          [common]
$CONFIG_controldir        [arex]
```

lrms=boinc:

```
$CONFIG_boinc_app_id   [lrms]
$CONFIG_boinc_db_host  [lrms]
$CONFIG_boinc_db_port  [lrms]
$CONFIG_boinc_db_user  [lrms]
$CONFIG_boinc_db_pass  [lrms]
$CONFIG_boinc_db_name  [lrms]
```

lrms=condor[3]:

```
# $CONFIG_enable_perflog_reporting    [common] not in reference
# $CONFIG_perflogdir                  [common] not in reference
# $CONFIG_controldir                  [arex] (for perflog)

$CONFIG_condor_requirements   [queue] [lrms]
$CONFIG_condor_rank                   [lrms]
# $CONFIG_shared_filesystem   [arex]
$CONFIG_condor_bin_path               [lrms]
$CONFIG_condor_config                 [lrms]
```

lrms=fork:

```
no variables
```

lrms=ll:

```
# $CONFIG_enable_perflog_reporting    [common] not in reference
# $CONFIG_perflogdir                  [common] not in reference
# $CONFIG_controldir                  [arex] (for perflog)

$CONFIG_ll_bin_path                   [lrms]
$CONFIG_ll_consumable_resources       [lrms]
$CONFIG_ll_parallel_single_jobs       *not in reference
# $CONFIG_scratchdir                  [arex]
```

lrms=lsf:

```
# $CONFIG_enable_perflog_reporting    [common] not in reference
# $CONFIG_perflogdir                  [common] not in reference
# $CONFIG_controldir                  [arex] (for perflog)

$CONFIG_lsf_architecture              [lrms]
$CONFIG_lsf_bin_path                  [lrms]
```

lrms=pbs:

```
# $CONFIG_enable_perflog_reporting    [common] not in reference
# $CONFIG_perflogdir                  [common] not in reference
# $CONFIG_controldir                  [arex] (for perflog)
```

(continues on next page)

[3] Here and following # prefix is for options and are used in `*_common` scripts and not unique to particular backend

(continued from previous page)

```
$CONFIG_pbs_queue_node        [queue]
$CONFIG_pbs_bin_path                  [lrms]
$CONFIG_nodememory                    [queue] ([infosys/cluster] parser substitution␣
↪fallback only)
$CONFIG_pbs_log_path                  [lrms]
# $CONFIG_shared_filesystem           [arex]
```

lrms=sge:

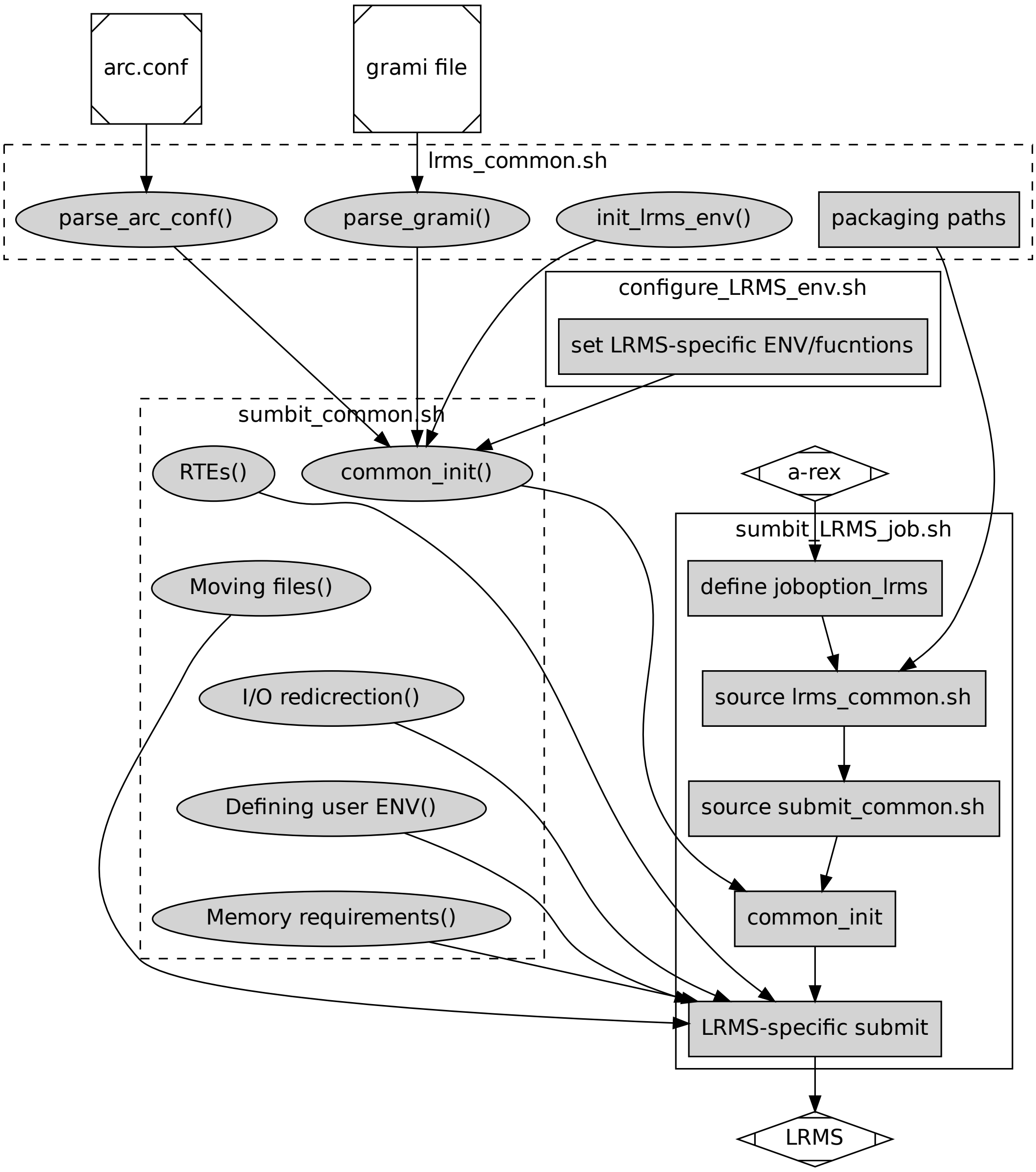
```
# $CONFIG_enable_perflog_reporting    [common] not in reference
# $CONFIG_perflogdir                  [common] not in reference
# $CONFIG_controldir                  [arex] (for perflog)

$CONFIG_sge_root                          [lrms]
$CONFIG_sge_cell                          [lrms]
$CONFIG_sge_qmaster_port              [lrms]
$CONFIG_sge_execd_port                [lrms]
$CONFIG_sge_bin_path                  [lrms]
$CONFIG_sge_jobopts                   [queue] [lrms]
# $CONFIG_scratchdir                  [arex]
```

lrms=slurm:

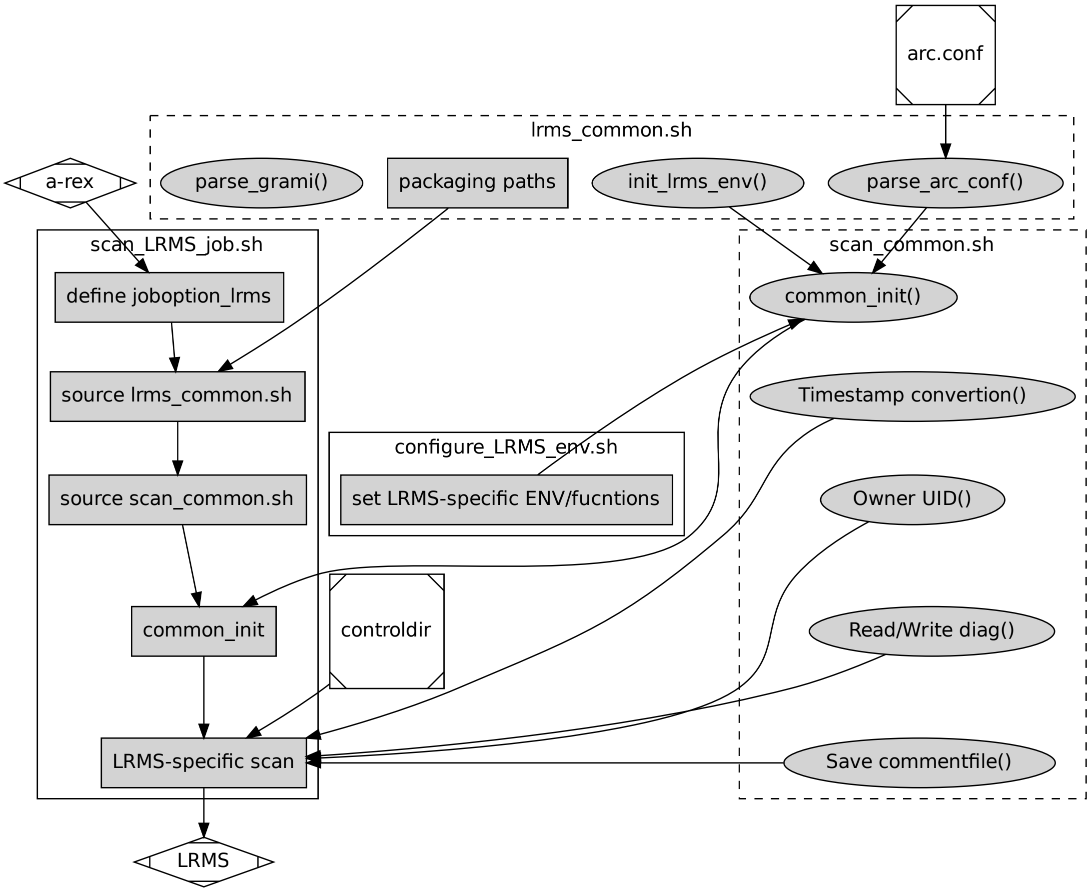
```
# $CONFIG_enable_perflog_reporting    [common] not in reference
# $CONFIG_perflogdir                  [common] not in reference
# $CONFIG_controldir                  [arex] (for perflog)

$CONFIG_slurm_wakeupperiod    [lrms]
$CONFIG_slurm_use_sacct               [lrms]
$CONFIG_slurm_bin_path                [lrms]
# $CONFIG_shared_filesystem           [arex]
```

Call graph

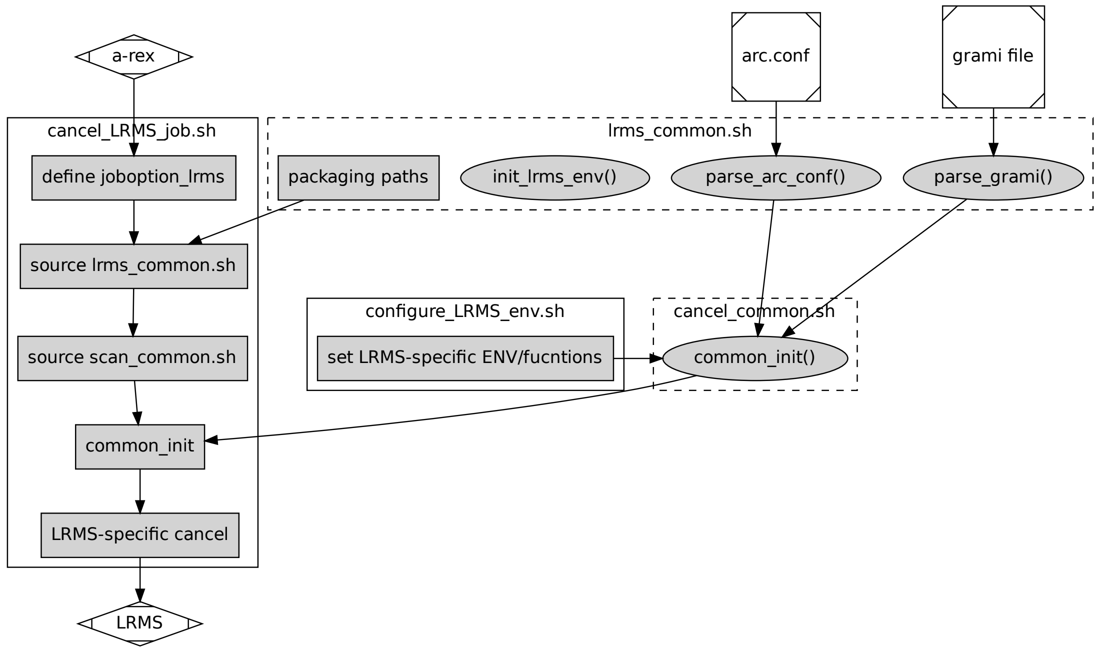
Submitting jobs



Scanning jobs



Canceling jobs



Changes in ARC6 memory limits processing:

Current logic of memory limits processing:

- `nodememory` - advertise memory for matchmaking: max memory on the nodes (in `[infosys/cluster]` block or per-queue)
- `defaultmemory` - enforce during submission if no memory limit specified in the job description (in `[lrms]` block or per-queue)

The ARC6 logic is *no enforcement = no limit*[1]

Backends behaviour with no memory enforcement limit:

- boinc - set to hardcoded 2GB
- condor - no enforcement
- form - no memory handling at all
- ll - no enforcement
- lsf - no enforcement
- pbs - no enforcement[2]
- sge - no enforcement
- slurm - no enforcement

[1] ARC5 logic was *no enforcement = max node memory* or 1GB if `nodememory` is not published (and not used for matchmaking)
[2] exclusivenode is memory-based and `nodememory` value is used in this case

6.2 Contributing to Documentation

NorduGrid ARC6 documentation is mainly written in reStructuredText and build with Sphinx to HTML pages, LaTeX (for printable PDF versions) and ePub.

Recent source tree (`master` branch) build is available instantly via Coderefinery GitLab Pages and deployed to the nordugrid.org on nightly basis.

Everyone is welcome to contribute. For trivial fixes just click on the "Edit in GitLab" button and commmit the changes.

You need to have an account in the Coderefinery GitLab and be a member of the project to be allowed to edit. Contact us if you want to contribute, but are not a member yet.

6.2.1 Commiting and reviewing changes

Contribution is possible by direct push to the repo, no need for merge requests if you are fixing typos or other content that does not requires review!

Note: In the renderend HTML version there is a link at the bottom of each page labeled *Edit page source on GitLab*. Using this link you can edit the page source in your web-browser without the need to checkout source tree

---

In case you want to add something new that requires the review process:

- create the new branch (in the same repository, no need to fork)
- open a merge request to `master` branch
- in-line and general discussion of the particular contribution is inside merge request.

6.2.2 Documentation structure

As reflected in the *main index* page, the documentation logically divided into the following groups:

- *Documentation for Infrastructure Users*
- *Documentation for Infrastructure Admins*
- *Documentation for Developers*
- *Technical Documents Describing ARC Components*

Admins documentation is the most developed part for ARC6. We aimed to have dedicated documents for each particular topic and then bound them in the following way:

- The introductory *Try ARC6: towards distributed computing in a few minutes* that contains a zero-configuration case hands-on instruction to help new users and admins getting started with ARC CE.
- Main *ARC Computing Element Installation and Configuration Guide* that contains general installation and configuration flow for production *Computing Element* deployment. The guide itself should be brief enough and holds only main examples. All detailed instruction for each particular subsystem configuration should be written in the dedicated document that linked to this guide.
- *ARC Configuration Reference Document* automatically rendered from text version in the code source tree. Targeted for web-search. The ultimate configuration options description we have.
- All other dedicated documents goes to *Operating ARC CE Subsystems* section in case someone wants to access them directly instead of following installation guides links.

6.2.3 Source tree directory structure

Sources of documents in rST format are placed into the `source` directory that contains the following structure:

- `_static` - logo and CSS files[1]
- `_templates` - Sphinx HTML theme layout tuning[Page 294, 1]
- `_extensions` - Custom sphinx extensions developed for ARC[1]
- **`common` - directory used to store common for all ARC customers documentation, including admins, users and developes. This documents are hidden in the main TOC tree, instead they are linked from other various documents when needed.**
  - `repos` - this directory holds repository configuration instructions for release, nightlies, etc. Main index page have repos pointers as well as *Try ARC6* and *Install Guide*.
  - `changelog` - documents that describe changes between ARC versions.
- **`admins` - directory used to store *Documentation for Infrastructure Admins*. The main documents that represent the entry points to *ARC Computing Element* installation and configuration resides directly in this directory.**
  - `details` - this subdirectory designed to hold detailed configuration instructions for ARC CE subsystems, like *RunTime Environments in ARC6* that describe all aspects of RTEs in-depth. This documents intentionally moved deeper in the main TOC tree to prevent first levels flooding. If you can find a chapter in main *install guide* that cat point to the in-depth document in question - this is a right place to store it.
  - `commands` - place for automatically generated command line option reference for ARC tools. At the time of writing the Python tools represented there.
  - `archery` - for the documents that not related to *Computing Element* another subdirectory should be created. At the time of writing only ARCHERY documentation of that kind is available.
- `developers` - directory used to store *Documentation for Developers*. As long as we don't have many of this kind of documents everything that contains implementation details (how stuff coded, which variable used, etc) should go there.
- `tech` - directory used to store *Technical Documents Describing ARC Components*. Now it holds only index document that contains references to the available PDFs that is verified to be applicable to ARC6 release. To add more, use template inside the index file.
- `users` - directory used to store *Documentation for Infrastructure Users*. If document is user-oriented - place it here.
- `sdk` - place for Doxygen SDK documentation integration to Sphinx build[1]
- `testing` - directory used to store the documents for *ARC 6 Testing Area*.
- `wip` - "work in progress" area hidden in the TOC tree. You can use it for incomplete documents that should not be publicly advertised yet, but still will be built and available via `/wip/` URL in produced HTML files.

Index files

Each directory in the source tree contains `index.rst` file that is used to link other documents in the same TOC tree. Upper-level `index.rst` file contains references to `index.rst` files in the subdirectories, that in turns contains pointers to the other documents.

Note: When you add a new document, add a reference to the `index.rst` in the same directory

---

Just follow the `index.rst` chain (look for `toctree` keyword) starting from the `source` directory to get familiar with TOC linking structure.

[1] Do not touch unless you are modifying the Sphinx build itself. Nothing there affects documentation writting process.

Storing images

When you need to add images to your document you should upload the image file itself and refer to it from the `.rst`.

ARC6 documentation structure implies that image files are stored inside the `images` subdirectory in the document location. Then referencing is done by relative path, e.g:

```
.. figure:: images/shared_sessiondir_yes.svg
```

6.2.4 Building the docs

The top directory in the source tree contains `build.sh` script that:

- checkout the ARC source code tree and configures it with packaging-like paths
- builds Doxygen SDK documentation
- converts `arc.conf.reference` to rST
- copies documentation parts from ARC source code tree (for *developers* section)
- prepare automatically generated documentation for CLI commands
- builds HTML
- builds PDF (with LaTeX)
- builds ePub

On commit the GitLab CI configured to automatically invoke the `build.sh` to produce rendered documentation. Documentation archive is avaiable as CI job artifacts and for `master` branch is deployed to GitLab Pages.

To build the docs on your local machine you should have at least Sphinx installed. Additionally you should be able to configure ARC source tree and ARC Python command dependencies for auto-generated parts.

Complete and up-to-date list of dependencies defined for CI build and can be found in `.gitlab-ci.yml`.

PDF builds with LaTeX is the most heavy part from the both time and needed additional packages perspective. For HTML rendering local debugging it is recommended to use the `html` script argument the skips PDF and ePUB:

```
[user@localhost doc]$ ./build.sh html
```

Point your browser to `file:///path/to/doc/build/html/index.html` to view the HTML rendering locally.

6.2.5 Writing Documentation in reStructuredText

reStructuredText (reST) is the default plaintext markup language used by Sphinx. There is possible to render other markups but for consistency and better cross-referencing NorduGrid ARC6 documentation written solely in reStructuredText.

General Syntax

reStructuredText markup specification is well documented in the several sources and was designed to be a simple and readable in plain-text. Common text editors (including `vim` and `emacs`) recognize reST markup and provide syntax highlighting out of the box.

Start with reStructuredText Primer on Sphinx docs.

Complete reST Markup Syntax can be found on Docutils starting with Quick reStructuredText document.

Sphinx also uses interpreted text roles to insert semantic markup (cross-referencing, etc) into docuemnts. To get familiar read this document.

Just study the markup following the documentation and readinig the already written documents just here.

Code snippets

There are different options for representing the code, starting with the simple literal blocks identified with `::`.

It is advised to use `code-block::` directive to enable syntax highlighted rendering:

```
.. code-block:: ini

[gridftpd/jobs]
allowaccess = staticdn dnfromfile
```

In this example, the `ini` tag represents the syntax highlighting lexer identfier for Pygments. Look for available lexers to serve your needs. The most common for our docs are: `console`, `bash`, `ini` and `cfg`.

References

Referencing another parts of documentation is necessary to achive usability. The typical referencing cases are:

Using custom label

Create label just above any of the paragraph headers with. The following markup creates `my_label` label.

```
 .. _my_label:

My Heading
==========
```

Refering to a lable is possible from any other doument with:

```
read the :ref:`my_label`
```

In this case paragraph heading will be used for hyperlink text. If you want some custom text for hyperlink text use the following syntax:

```
read this :ref:`text <my_label>`
```

Referring arc.conf.reference

Autogenerater reST rendering of `arc.conf.reference` already contains labels for all configuration options and blocks that can be used.

The label name has the following structure:

- `reference_<block name>[_<sub_block>...]_<option name>` represents configration option inside block
- `reference_<block name>[_<sub_block>...]` represents block itself

For example:

```
In the ``arc.conf`` there is a dedicated :ref:`[lrms] <reference_lrms>` block that␣
↪defines the type of your LRMS.
*Job session directory* is configured with :ref:`reference_arex_sessiondir`␣
↪configurration option.
```

Referring bugz

Custom Sphinx plugin for ARC documentation introduces new reST roles that can be used to mention bugs in NorduGrid Bugzilla or GitLab Issues:

```
More details can be found in :bugz:`3802` and :issue:`57`.
```

Referencing docs

Referencing the the whole document is similar to using labels, but instead of label name the document name (filename without extension) is used with `:doc:` keyword.

Example 1: Refer to the `try_arc6.rst` in the same source tree directory (relative path). Use the document header as hyperlink text:

```
:doc:`try_arc6`
```

Example 2: Refer to the `repository.rst` by absolute path (starting from sources top directory). Use the custom hyperlink text:

```
:doc:`NorduGrid Repositories </common/repos/repository>`
```

Adding notes

To highlight statement visually use notes and warnings:

```
.. note::
   Zero configured A-REX comes with EMI-ES and REST interfaces enabled.

.. warning::
   This information is valid for releases of ARC 6 starting from 6.0.
```

Adding images

In the ARC6 documentation images should be included with the caption. It is accomplished with the figure keyword.

After *storing image* inside `images` subdirectory, include it in reST document as:

```
.. figure:: images/shared_sessiondir_yes.svg
   :align: center
   :alt: Sessiondir is shared between ARC CE and WNs

   Sessiondir is shared between ARC CE and WNs. No local scratchdir defined.
```

Sphinx build had been configured with image format autocoversion feature. So you can use any image format, including vector graphics.

Graphviz

If you want to illustrate some process or structure that can be shown as graph, consider using built-in Graphviz functionality.

Simple usage examples can be found in `admins/details/rtes.rst` document within ARC documentation. Give it a try, it is easy.

Converting from other sources

Converting the old documentation parts from LaTeX or Mediawiki markdown to reST is not a magical process that do everything automatically unfortunalely.

However you can get a good start with a pandoc tool that do the conversion.

E.g. to convert LaTeX source simply run:

```
[user ~]$ pandoc -f latex -t rst acix.tex > acix.rst
```

The ammount of efforts to edit the resulted `.rst` file and fix formatting issues is completely depends on the source itself (e.g. how the console ouput was formatted in the origin document). Images and references in most cases should be fixed separately after the pandoc.

CHAPTER

SEVEN

DOCUMENTATION FOR INFRASTRUCTURE USERS

This part of the documentation targeted to distributed computing infrastructure users that use either clients or SDK to run jobs and handle data transfers.

7.1 Installing ARC Client Tools

Warning: Information in this document is fully applicable to ARC 6.5.0 and above only!

7.1.1 Step 1. Enable NorduGrid ARC6 repos

Prepare your system to install via the *NorduGrid Repositories*.

If you need the latest client tools that are not yet released, configure nightly builds repository following *Using ARC packages from nightly builds* instructions.

Note: On the several operating systems you can install ARC without NorduGrid Repositories (e.g. using EPEL for CentOS/RHEL). But even on that system you will benefit from NorduGrid Repositories as they contains IGTF certificates and gets updated faster on new release.

7.1.2 Step 2. Install packages

Client tools

The main client tools package is `nordugrid-arc-client`. Install it using your OS packet manager[1]:

```
[root ~]# yum -y install nordugrid-arc-client
```

This package brings necessary plugins as dependency and allows you to submit jobs and perform data transfers in the ARC ecosystem, e.g.:

- submit jobs via EMI-ES interface defined by XRSL or ADL descriptions
- query ARCHERY registry and LDAP information system
- support local and HTTP(S) data transfers
- support Rucio, SRM and ACIX

If you need more features, additional plugins are avaialable (see below).

[1] Examples are shown for YUM-based systems. You can use APT or any other packet managers in the similar way.

ARCCTL

ARC Control Tool is one-stop-shop for sysadmins and users running ARC6 that automate many operations.

For the client side it will allows you to easiely deploy CA certificates and VOMS configuration:

```
[root ~]# yum -y install nordugrid-arc-arcctl
```

> **Warning:** The *ARC Control Tool* is available for stand-alone usage without ARC CE since version 6.5. For ealier ARC versions CA certificates and VOMS configuration should be installed manually. You can find some hints in the old client installation instructions

Additional plugins

You can consider installing additional plugins for extra functionality:

- Job submission:
  - `arcrest` - submit jobs via ARC REST interface
  - `gridftpjob` - submit jobs via GRIDFTPJOB interface (installs `gridftp` as a dependency) [globus]
  - `internal` - submit jobs via local filesystem *Internal* interface (requires A-REX on the same host)
- Data transfers:
  - `gridftp` - support for gridftp data transfers [globus]
  - `xrootd` - support for xroot protocol
  - `s3` - support for s3 protocol
  - `gfal` - add support for numerous data transfer protocols and file catalogues via installed GFAL2 plugins

```
[root ~]# yum -y install nordugrid-arc-plugins-<PLUGIN NAME>
```

7.1.3 Step 3. Setting up credentials

Currently users authentication in e-Science distrbuted computing networks (grid) heavily relies on cryptography and uses personal X.509 certificates/keys to identify entities and set of dedicated Certification Athorities (CA).

In the most common workflow you are also required to be a member of some Virtual Organization (VO) to get access to infrastructure resources. On the technical level the VO membership is currently handled by infrastructure VOMS services.

CA certificates bundle

CA certificates used to verify entities (e.g. users as well as compute and storage services) and should be installed on the client host as well for infrastructure security reasons.

In the distributed grid environment set of dedicated CAs used as a part of IGTF.

IGFT CA certificates bundle can be easily installed with *ARC Control Tool*[2]:

```
[root ~]# arcctl deploy igtf-ca classic
```

[2] In case ARC is not installed from the NorduGrid repositories, use `--installrepo` argument to enable third-party repositories with IGTF CA certificates

Personal X.509 certificate

For production infrastructure usage you shoud obtain a personal certificate signed by one of the IGTF accredited CAs.

Typically there is at least one IGTF accreditage CA in each country that you can find on map.

Oganizational and technical procedures varies from CA to CA, so you should read the instructions on a choosen CA web-site.

Once you get your certificte, install it to your client host and ensure the permissions are set correctly. For historical reasons the default location for certificate and key is `.globus` unless redefined in *client configuration file*:

```
[user@client ~]$ ls -l ~/.globus/
total 12
-rw-r--r-- 1 user user 6353 Oct  1 11:55 usercert.pem
-r-------- 1 user user 1854 Oct  1 11:55 userkey.pem
```

Note: For testing and development purposes it is possible to use local testing CA. *ARC Control Tool* provides built-in test CA capabilities that allows you to bootstrap own test CA and generate host and *user certificates*.

Virtual Organization memberhip

In most cases you do have some implicit VO affiliation on your workplace. If not, you can search for VOs using e.g. EGI VO(s) search tool.

Every Virtual Organisation (VO) has own procedures and policies. Please contact VO support team for membership instructions.

To prove that you are a member of the particular VO you need to obtain a special token from the VOMS server as a part of your authentication process.

The `arcproxy` tool will get this token automatically, but requires VOMS server enpoint details to be configured.

With *ARC Control Tool* you only need to know the VO name (for VOs in EGI database) or VOMS URL[3]:

```
[root ~]# arcctl deploy vomses --egi-vo atlas
[root ~]# arcctl deploy vomses --voms  https://voms.ndgf.org:8443 --use-client-cert␣
↪nordugrid.org
```

7.1.4 Step 4. Try it out

When all is set, you can try to submit a job to the grid infrastructure.

Create proxy certificate

To submit a job, or perform any other action you need a so-called *proxy-certificate* which is a Single Sign-On token for distributed grid-infrastructure. It is generated in the following way:

```
[user ~]$ arcproxy
Your identity: /DC=org/DC=AccreditatedCA/O=people/CN=John Smith
Proxy generation succeeded
Your proxy is valid until: 2019-11-28 02:06:57
```

If you need to include you VO membership confirmation token (attribute certificate), specify the VO name as well:

[3] Most of VOMS-Admin services by default prohibit access without client certificate even to configuration page. If this is your case, the `--use-client-cert` option will instruct `arcctl` to use your personal certificate to establish connection.

```
[user ~]$ arcproxy -S area51
Your identity: /DC=org/DC=AccreditatedCA/O=people/CN=John Smith
Contacting VOMS server (named area51): voms.example.org on port: 15001
Proxy generation succeeded
Your proxy is valid until: 2019-11-28 02:08:27
```

Submit test job

ARC client tools includes `arctest` utility that come with several test jobs on board. So you can try to submit job without the need to write a job description.

You can use NorduGrid top-level ARCHERY service registry (`nordugrid.org`) to find available CEs automatically[4]:

```
[user ~]$  arctest -J 2 --registry nordugrid.org
Submitting test-job 2:
&( executable = "/usr/bin/env" )( stdout = "stdout" )( stderr = "stdout" )( gmlog =
↪"gmlog" )( jobname = "arctest2" )( clientxrsl = "&( executable = ""/usr/bin/env""␣
↪)( jobname = ""arctest2"" )( stdout = ""stdout"" )( join = ""yes"" )( gmlog = "
↪"gmlog"" )" )
Client version: nordugrid-arc-6.5.0
Test submitted with jobid: https://arc.example.org:443/arex/
↪oIlKDmiOCuvnjwO5upha6lOqABFKDmABFKDmEFHKDmPBFKDmUYtvNo
Computing service: KNU ARC
```

The job status can be than checked with the `arcstat` tool:

```
[user ~]$ arcstat https://arc.example.org:443/arex/
↪oIlKDmiOCuvnjwO5upha6lOqABFKDmABFKDmEFHKDmPBFKDmUYtvNo
Job: https://arc.example.org:443/arex/
↪oIlKDmiOCuvnjwO5upha6lOqABFKDmABFKDmEFHKDmPBFKDmUYtvNo
 Name: arctest2
 State: Running

Status of 1 jobs was queried, 1 jobs returned information
```

To fetch the job's stdout run `arccat` tool:

```
[user ~]$ arccat https://arc.example.org:443/arex/
↪oIlKDmiOCuvnjwO5upha6lOqABFKDmABFKDmEFHKDmPBFKDmUYtvNo
GRIDMAP=/dev/null
HOSTNAME=arc.example.org
TMPDIR=/tmp
<output omitted>
```

[4] HINT: you can add `-d INFO` to get mode dateails of CE selection process

7.2 Overview of ARC client tools

This document contains general overview of available ARC client tools. For more detailed information about possible tool usage options, please refer to the `--help` messages and corresponding man pages.

7.2.1 Submission endpoint types

Table 7.1: The submission types short, long and legacy style

short	long	legacy
emies	org.nordugrid.emies	org.ogf.glue.emies.activitycreation
gridftpjob	org.nordugrid.gridftpjob	org.nordugrid.gridftpjob
arcrest	org.nordugrid.arcrest	N/A
internal	org.nordugrid.internal	org.nordugrid.internal

Example of use:

```
arcsub -T emies -C some-ce-endpoint
```

7.2.2 Credentials

arcproxy

The `arcproxy` command creates a proxy certificate from a key/certificate pair which can then be used to access resources.

It is all-in-one tool for all kind of operations with proxy certificates, including VOMS extension generation and communications with MyProxy services.

```
[user ~]$ arcproxy -S myVO
Enter pass phrase for private key:
Your identity: /DC=org/DC=AccreditatedCA/O=people/CN=John Smith
Contacting VOMS server (named myVO): voms.example.org on port: 15004
Proxy generation succeeded
Your proxy is valid until: 2020-01-22 22:50:44

[user ~]$ arcproxy -I
Subject: /DC=org/DC=AccreditatedCA/O=people/CN=John Smith/CN=2123720174
Issuer: /DC=org/DC=AccreditatedCA/O=people/CN=John Smith
Identity: /DC=org/DC=AccreditatedCA/O=people/CN=John Smith
Time left for proxy: 11 hours 56 minutes 45 seconds
Proxy path: /tmp/x509up_u1000
Proxy type: X.509 Proxy Certificate Profile RFC compliant impersonation proxy - RFC␣
↪inheritAll proxy
Proxy key length: 2048
Proxy signature: sha256
====== AC extension information for VO myVO ======
VO        : myVO
subject   : /DC=org/DC=AccreditatedCA/O=people/CN=John Smith
issuer    : /DC=org/DC=AccreditatedCA/O=hosts/CN=voms.example.org
uri       : voms.example.org:15004
attribute : /myVO/Role=NULL/Capability=NULL
Time left for AC: 11 hours 56 minutes 47 seconds
```

(continues on next page)

(continued from previous page)

```
[user ~]$ arcproxy -L myproxy.example.org -M put -S myVO -U myproxyuser
Enter pass phrase for private key:
Your identity: /DC=org/DC=AccreditatedCA/O=people/CN=John Smith
Contacting VOMS server (named myVO): voms.example.org on port: 15004
Proxy generation succeeded
Your proxy is valid until: 2020-01-29 10:51:52
Enter pass phrase for MyProxy server (new):
Verifying - Enter pass phrase for MyProxy server (new):
Succeeded to put a proxy onto MyProxy server
```

arcrenew

The `arcrenew` command is used for renewing the proxy of jobs that have been submitted to computing resources.

```
[user ~]$ arcrenew https://arc.example.org:443/arex/
↪7LqLDmKESEwnf5481mks8bjnABFKDmABFKDmhNFKDmHGFKDm9BpmKn
Jobs processed: 1, renewed: 1
```

7.2.3 Job submission and management

arcsub

The `arcsub` command is used for submitting jobs to computing resources.

Note: Submission endpoint selection options are changed in ARC 6.5.0 release

```
[user ~]$ arcsub -T emies -C arc.example.org myJob.xrsl
Job submitted with jobid: https://arc.example.org:443/arex/
↪5SwNDmmwREwnf5481mks8bjnABFKDmABFKDmhNFKDmKFFKDmNPeLgn
```

arcstat

The `arcstat` command is used for obtaining the status of jobs that have been submitted to computing resources.

```
[user ~]$ arcstat https://arc.example.org:443/arex/
↪5SwNDmmwREwnf5481mks8bjnABFKDmABFKDmhNFKDmKFFKDmNPeLgn
Job: https://arc.example.org:443/arex/
↪5SwNDmmwREwnf5481mks8bjnABFKDmABFKDmhNFKDmKFFKDmNPeLgn
 Name: myJob
 State: Running

Status of 1 jobs was queried, 1 jobs returned information
```

arccat

The `arccat` command performs the cat command on the stdout, stderr or A-REX error log of the job.

It can also show the specified file from job's session directory.

```
[user ~]$ arccat https://arc.example.org:443/arex/
↪5SwNDmmwREwnf5481mks8bjnABFKDmABFKDmhNFKDmKFFKDmNPeLgn
SHELL=/bin/bash
HISTSIZE=1000
<output omitted>

[user ~]$ arccat -l https://arc.example.org:443/arex/
↪5SwNDmmwREwnf5481mks8bjnABFKDmABFKDmhNFKDmKFFKDmNPeLgn
2020-01-22T09:57:45Z Job state change UNDEFINED -> ACCEPTED   Reason: (Re)Accepting␣
↪new job
2020-01-22T09:57:45Z Job state change ACCEPTED -> PREPARING   Reason: Starting job␣
↪processing
2020-01-22T09:57:45Z Job state change PREPARING -> SUBMIT   Reason: Pre-staging␣
↪finished, passing job to LRMS
----- starting submit_pbs_job -----
PBS jobname: myJob
<output omitted>
```

arckill

The `arckill` command is used to kill running jobs.

```
[user ~]$ arckill https://arc.example.org:443/arex/
↪wOOKDmT2REwnf5481mks8bjnABFKDmABFKDmhNFKDmWFFKDm1FSSRn
Jobs processed: 1, successfully killed: 1, successfully cleaned: 1
```

arcclean

The `arcclean` command removes a job from the computing resource.

```
[user ~]$ arckill --keep https://arc.example.org:443/arex/
↪UcBLDm32REwnf5481mks8bjnABFKDmABFKDmhNFKDmiFFKDm0zk4un
Jobs processed: 1, successfully killed: 1

[user ~]$ arcclean https://arc.example.org:443/arex/
↪UcBLDm32REwnf5481mks8bjnABFKDmABFKDmhNFKDmiFFKDm0zk4un
Jobs processed: 1, deleted: 1
```

arcresub

The `arcresub` allows to resubmit job to the same or other compute resources.

```
[user ~]$ arcresub --same https://arc.example.org:443/arex/
↪5SwNDmmwREwnf5481mks8bjnABFKDmABFKDmhNFKDmKFFKDmNPeLgn
Job submitted with jobid: https://arc.example.org:443/arex/
↪KSeNDmz5REwnf5481mks8bjnABFKDmABFKDmhNFKDmsFFKDmp4MPWn
```

arcresume

The `arcresume` command is used for resuming a job that was submitted to compute resources and then subsequently failed. The job will be resumed at the last ok state reported by the cluster.

```
[user ~]$ arcresume https://arc.example.org:443/arex/
↪vIGMDm58REwnf5481mks8bjnABFKDmABFKDmhNFKDm6FFKDmOBYWDm
Jobs processed: 1, resumed: 1
```

arcget

The `arcget` command is used for retrieving the results from a job.

```
[user ~]$ arcget --keep https://arc.example.org:443/arex/
↪5SwNDmmwREwnf5481mks8bjnABFKDmABFKDmhNFKDmKFFKDmNPeLgn
Results stored at: 5SwNDmmwREwnf5481mks8bjnABFKDmABFKDmhNFKDmKFFKDmNPeLgn
Jobs processed: 1, successfully retrieved: 1
```

arctest

The arctest command is used for testing resources. It is able to:

- submit several test job
- prints info about installed user- and CA-certificates

Note: Submission endpoint selection options are changed in ARC 6.5.0 release

```
[user ~]$ arctest -J 2 -C arc.example.org
Job submitted with jobid: https://arc.example.org:443/arex/
↪rRVLDmSBSEwnf5481mks8bjnABFKDmABFKDmhNFKDm8FFKDmGoZf3m

[user ~]$ arctest -E
Certificate information:
  Certificate: /home/john/.globus/usercert.pem
  Subject name: /DC=org/DC=AccreditatedCA/O=people/CN=John Smith
  Valid until: 2020-04-21 18:48:15

Proxy certificate information:
  Proxy: /tmp/x509up_u1000
  Proxy-subject: /DC=org/DC=AccreditatedCA/O=people/CN=John Smith/CN=2123720174
  Valid for: 11 hours 37 minutes 33 seconds

Certificate issuer: /DC=org/DC=AccreditatedCA

CA-certificates installed:
  /DC=org/DC=AccreditatedCA
<output omitted>
```

7.2.4 Data manipulation

arcls

The `arcls` command is used for listing files in storage elements and replica catalogs.

```
[user ~]$ arcls srm://se.example.org/dpm/example.org/home/myVO/
data.1
data.2

[user ~]$ arcls -lL rucio://rucio-lb-prod.cern.ch/replicas/mc15_13TeV/EVNT.19785567._
↪000729.pool.root.1
<Name>                           <Type>  <Size>       <Modified>      <CheckSum>     ␣
↪  <Latency>
EVNT.19785567._000729.pool.root.1  file 167468788       (n/a)         ␣
↪adler32:f7332aeb      (n/a)
     root://xrootd.lcg.triumf.ca:1094//atlas/atlasdatadisk/rucio/mc15_13TeV/ee/3b/
↪EVNT.19785567._000729.pool.root.1
     srm://srm.ndgf.org:8443/srm/managerv2?SFN=/atlas/disk/atlasdatadisk/rucio/mc15_
↪13TeV/ee/3b/EVNT.19785567._000729.pool.root.1
     gsiftp://gridftp.echo.stfc.ac.uk:2811/atlas:datadisk/rucio/mc15_13TeV/ee/3b/EVNT.
↪19785567._000729.pool.root.1
```

arccp

The `arccp` command copies files to, from and between storage services.

```
[user ~]$ arccp /mnt/data/data.123 srm://se.example.org/dpm/example.org/home/myVO/
↪data.123

[user ~]$ arccp rucio://rucio-lb-prod.cern.ch/replicas/mc15_13TeV/EVNT.19785567._
↪000729.pool.root.1 /tmp/
```

arcmkdir

The `arcmkdir` command creates directories on storage elements and catalogs.

```
[user ~]$ arcmkdir srm://se.example.org/dpm/example.org/home/myVO/myData
```

arcrename

The `arcrename` command renames files on storage elements.

```
[user ~]$ arcrename srm://se.example.org/dpm/example.org/home/myVO/data.123 srm://se.
↪example.org/dpm/example.org/home/myVO/data.124
```

arcrm

The `arcrm` command deletes files on storage elements.

```
[user ~]$ arcrm srm://se.example.org/dpm/example.org/home/myVO/data.124
```

7.2.5 Information services

arcinfo

The `arcinfo` command is used for obtaining the status of computing resources and detailed information about the resource according to published data in the information system.

The summary optput includes the information endpoint type starting from ARC 6.5.0.

```
[user ~]$ arcinfo arc.example.org
Computing service:  (production)
Information endpoint: ldap://arc.example.org:2135/Mds-Vo-Name=local,o=grid (org.
↪nordugrid.ldapng)
Information endpoint: ldap://arc.example.org:2135/o=glue (org.nordugrid.ldapglue2)
Information endpoint: https://arc.example.org:443/arex (org.nordugrid.arcrest)
Information endpoint: https://arc.example.org:443/arex (org.ogf.glue.emies.
↪resourceinfo)
Submission endpoint: https://arc.example.org:443/arex (status: ok, interface: org.
↪nordugrid.arcrest)
Submission endpoint: https://arc.example.org:443/arex (status: ok, interface: org.ogf.
↪glue.emies.activitycreation)
Submission endpoint: gsiftp://arc.example.org:2811/jobs (status: ok, interface: org.
↪nordugrid.gridftpjob)

[user ~]$ arcinfo -l arc.example.org | grep Implementation
    Implementation name: nordugrid-arc-6.5.0
```

arcsync

The `arcsync` command synchronizes your local jobs database with the information at a given computing element(s).

```
[user ~]$ $ arcsync -C arc.example.org
Synchronizing the local list of active jobs with the information in the
information system can result in some inconsistencies. Very recently submitted
jobs might not yet be present, whereas jobs very recently scheduled for
deletion can still be present.
Are you sure you want to synchronize your local job list? [y/n] y
Total number of new jobs found: 7
```

7.3 How to submit the job?

> **Warning:** Information in this document is fully applicable to ARC 6.5.0 and above only!

The heterogeneous nature of the distributed computing requires the formal *job description* in order to match resources and execution environment of the particular CE to job needs.

The *client tools* than, based on the job description, select submission endpoint and pass job to chosen CE for processing.

7.3.1 Submission sequence

Make sure you have *client tools* installed and credentials configured first.

Than follow the sequence below.

Generate proxy-certificate

To submit a job, or perform any other action towards the ARC server you need a so-called proxy-certificate which is a Single Sign-On token for distributed grid-infrastructure. It is generated using `arcproxy` tool.

Note: In most production cases you need to add VO affiliation information to proxy certificate with `-S <VO>` option.

```
[user ~]$ arcproxy -S myVO
Your identity: /DC=org/DC=nordugrid/DC=ARC/O=TestCA/CN=My Cert
Contacting VOMS server (named myVO): voms.myvo.example.org on port: 15001
Proxy generation succeeded
Your proxy is valid until: 2020-02-29 06:31:54
```

Create job description

ARC6 supports the *Extended Resource Specification Language (xRSL)* and EMI Activity Description Language (ADL)[1] for specifying job descriptions.

According to the syntax you should write xRSL or ADL file, describing your job files and resource requirements.

For example, the xRSL job description for `arctest -J 2` job is:

```
&( executable = "/usr/bin/env" )( jobname = "arctest2" )( stdout = "stdout" )( join =
↪"yes" )( gmlog = "gmlog" )
```

The same description in ADL:

```
<?xml version="1.0"?>
<ActivityDescription xmlns="http://www.eu-emi.eu/es/2010/12/adl" xmlns:emiestypes=
↪"http://www.eu-emi.eu/es/2010/12/types" xmlns:nordugrid-adl="http://www.nordugrid.
↪org/es/2011/12/nordugrid-adl">
  <ActivityIdentification>
    <Name>arctest2</Name>
  </ActivityIdentification>
  <Application>
```

(continues on next page)

[1] See section 9.3 of EMI-ES document for complete ADL specification

(continued from previous page)

```
    <Executable>
      <Path>/usr/bin/env</Path>
    </Executable>
    <Output>stdout</Output>
    <Error>stdout</Error>
    <LoggingDirectory>gmlog</LoggingDirectory>
  </Application>
</ActivityDescription>
```

Run arcsub

The `arcsub` tool is responsible for job submission. This includes selecting (matchmaking) resources based on the job description and sending the job description and all job files to selected submission endpoint.

In the simplest case you can use a top-level *Nordugrid ARCHERY registry* to find CE in entire e-Inftrastrucutre that fits your job and allows you to use it resources:

```
[user ~]$ arcsub --registry nordugrid.org myjobdescription.xrsl
Job submitted with jobid: https://arc.example.org:443/arex/
↪R7pMDmXI8Dwnf5481mks8bjnABFKDmABFKDmhNFKDmVEFKDmQ0taDn
```

To specify the CEs you are aiming to submit jobs to more precisely, please read the next section for details.

7.3.2 Specify CEs for job submission

The `arcsub` accepts a set of options that allow to select target CE for the job submission.

There CEs can be passed manually (using `-C, --computing-element=ce` option) or fetched from the registries (`-Y, --registry=registry`).

The submission endpoint type (actual job submission interface) and information endpoint type (interface to query the information for matchmaking and brokering) can be defined as well, if you want to use some interface explicitely.

Following is a detailed description of how options works together to select submission endpoint, including the examples of usage.

> **Warning:** It is not possible to mix old set of ARC5 options for CE selection with ARC6 options described below

Comuputing Element

The Computing Element (`-C, --computing-element=CE`) used to specify computing element name or a a complete endpoint URL directly.

Supported options:

- CE name (given as FQDN of the cluster frontend)
- URL of info or jobsubmission endpoint on the CE

The URL is one of the valid info or jobsubmission URLs of the computing element. The usage of URL is needed when the services are ruining on non-default ports or path.

Except the case when *info-endpoint-type* is `NONE`, the URL is interpreted as info-endpoint URL. In case of `--info-endpoint-type=NONE`, the URL represents the jobsubmission endpoint URL.

If only the `--computing-element` is specified without any preferred info-endpoint-type or submission-endpoint-type then `arcsub` will fetch resource info from all the available info endpoints parallel AND select from the discovered and supported (via available plugin) jobsubmission interfaces.

Examples:

```
[user ~]$ arcsub --computing-element my.cluster.org
```

will trigger a submission to a randomly selected jobsubmission-endpoint-type advertised via local infosys and supported by installed submission plugins.

```
[user ~]$ arcsub --computing-element ldap://my.cluster.org:389/o=glue
```

will trigger a submission to a available jobsubmission endpoint discovered from the info interface running on the ldap port 389.

```
[user ~]$ arcsub --submission-endpoint-type emies --computing-element https://my.
↪cluster.org:60000/emies
```

will triger a submission to the emies jobsubmission endpoint after the infosys discovery completed against the specified emies info endpoint URL. The URL above is interpreted as info URL.

```
[user ~]$ arcsub --submission-endpoint-type emies --info-endpoint-type NONE --
↪computing-element https://my.cluster.org:60000/emies
```

will triger a submission to the emies jobsubmission endpoint specified via the given URL. Info discovery is turned off. The URL is interpreted as jobsubmission endpoint URL.

Registry

The `-Y, --registry=registry` option used to specify service endpoint URL registries.

It takes FQDN's of ARCHERY or EGIIS servers with optional specification of protocol.

Supported options:

- ARCHERY: `arcsub --registry nordugrid.org` or `arcsub --registry dns://nordugrid.org`
- EGIIS: `arcsub --registry index1.nordugrid.org` or `arcsub --registry ldap://index1.nordugrid.org:2135/mds-vo-name=ATLAS,o=grid`

Support for EMIR and BDII service catalogues are REMOVED and the corresponding plugins are not part of ARC6.

In case only a short hostname given with `--registry` without specifying the full protocol URL (ldap or dns), then `arcsub` will attempt to contact both ldap://hostname and dns://hostname.

Examples using `--registry` together with `--info-endpoint-type` and `--submission-endpoint-type`:

```
[user ~]$ arcsub --submission-endpoint-type emies --registry dns://nordugrid.org
```

will fetch all information endpoint of emies info type (matching emies submission type) from the registry, query these endpoints for available emies submission endpoints and choose one of them (via brokering, default is random) to submit job.

```
[user ~]$ arcsub --info-endpoint-type ldap.nordugrid --registry dns://nordugrid.org
```

will fetch all information endpoint of specified `ldap.nordugrid` type from the registry, query them for available submission endpoints and try to submit to one of them, prioritizing matching `gridftpjob` submission endpoint type first.

```
[user ~]$ arcsub --info-endpoint-type ldap.glue2 --submission-endpoint-type emies --
↪registry dns://nordugrid.org
```

will fetch all information endpoint of specified `ldap.glue2` type from the registry, query them for available submission endpoints of specified `emies` type and choose one of them (via brokering, default is random) to submit job.

```
[user ~]$ arcsub --info-endpoint-type NONE --submission-endpoint-type emies --
↪registry dns://nordugrid.org
```

will fetch all SUBMISSION endpoints of specified `emies` type from the registry and select one of them to directly submit job without info endpoint querying. There are no other info to do matchmaking or prioritizing in this case, so the random broker will be used to select one of the available submission endpoints.

Info endpoint type

Use the `-Q, --info-endpoint-type=type` for choosing the cluster/resource information retrieval interface type.

Supported options:

```
org.nordugrid.internal
org.nordugrid.ldap.nordugrid
org.nordugrid.ldap.glue2
org.nordugrid.emies
org.nordugrid.arcrest
NONE
```

The special value `NONE` will disable the local infosys queries during job submission.

Support for BES, WSRFGLUE2, GLUE1 are REMOVED and the corresponding plugins deleted from ARC6.

Note: `org.nordugrid` prefix can be omitted, e.g. both `org.nordugrid.emies` and `emies` should be accepted.

The `arcsub` client in case NO submission-endpoint-type is specified and ONLY the info-endpoint-type given will first try to use a predefined submission-endpoint-type following the rules given below.

Table 7.2: The submission types to try corresponding to a given info-endpoint-type

info-endpoint-type	submission-endpoint-type
emies	emies
arcrest	arcrest
internal	internal
ldap.nordugrid	gridftpjob
ldap.glue2	first emies then gridftpjob
NONE	first emies then gridftpjob

The `arcsub` command will guess the job submission endpoint URLs based on the default values corresponding to a specific submission-endpoint-type.

Examples:

```
[user ~]$ arcsub --info-endpoint-type emies --computing-element my.cluster.org
```

will trigger a jobsubmission where cluster info is obtained from `emies` infosource and jobsubmission is done via `emies` using a guessed submission URL of https://my.cluster.org:443/arex

```
[user ~]$ arcsub --info-endpoint-type NONE --computing-element my.cluster.org
```

will trigger a jobsubmission where cluster info is NOT collected, infosys is completely ignored and the client tries to submit via `emies` submission-endpoint-type again guessing jobsubmision endpoint URLs (https://my.cluster.org:443/arex). IN case emies submission failed the gridftpjob endpoint will be tried.

```
[user ~]$ arcsub --info-endpoint-type org.nordugrid.ldap.glue2 --computing-element my.
→glue2.cluster
```

will trigger a jobsubmission where cluster info containing jobsubmission URLs are obtained from GLUE2 ldap rendering infosource (guessed as ldap://my.glue2.cluster:2135/o=glue) and jobsubmission is first tried via emies jobinterface then gridftpjob (in case emies failed) URLS obtained from infosys query.

```
[user ~]$ arcsub -- info-endpoint-type org.nordugrid.ldap.glue2 --submission-endpoint-
→type gridftpjob --computing-element my.glue2.cluster
```

will trigger a jobsubmission where cluster info containing the jobsubmission URLs are obtained from GLUE2 ldap rendering infosource and jobsubmission is performed via gridftpjobs channel.

Submission endpoint type

Use `-T, --submission-endpoint-type=type` for choosing the jobsubmission endpoint type.

Supported options:

```
org.nordugrid.internal
org.nordugrid.emies
org.nordugrid.gridftpjob
org.nordugrid.arcrest
```

Note: `org.nordugrid` prefix can be omitted, e.g. both `org.nordugrid.emies` and `emies` should be accepted.

support for BES, CREAM, ARC1, unicore, ARC0 (now called gridftpjob) submission interface types are REMOVED and the corresponding plugins deleted from ARC6.

Proposed arc client behavior in case of ONLY submission-endpoint-type specified and NO info-endpoint-type is given: the client USE ONLY the following matching info interfaces:

Table 7.3: The info-endpoint-type corresponding to a given submission-endpoint-type

submission-endpoint-type	info-endpoint-type
emies	emies
arcrest	arcrest
internal	internal
gridftpjob	both ldap.glue2 and ldap.nordugrid

Examples:

```
[user ~]$ arcsub --submission-endpoint-type emies --computing-element my.cluster.org
```

will trigger a jobsubmission where cluster info is obtained from emies infosource (via the guessed info endpoint https://my.cluster.org:443/arex) and jobsubmission is done via emies jobsubmission interface

```
[user ~]$ arcsub --submission-endpoint-type emies --info-endpoint-type org.nordugrid.
→glue2 --computing-element my.emies.cluster.org
```

will trigger a jobsubmission process where cluster info is obtained from glue2 ldap info endpoint guessed URL (ldap://my.cluster.org:2135/o=glue) and job is submitted via emies

```
[user ~]$ arcsub --info-endpoint-type NONE --submission-endpoint-type gridftpjob --
↪computing-element my.cluster.org
```

will trigger a direct jobsubmission via the gridftpjob submission interface without any query to the local LDAP or whatever else infosys. The jobsubmission URL (gsiftp://my.cluster.org:2811/jobs) is guessed using defaults.

7.4 How to work with data?

ARC libraries are very flexible in terms of supported data transfer protocols. It is designed to be extendable via a set of plugable Data Management Components (DMC).

The availble protocols to work with depends on the DMCs *installed on the system* (look for *Additional plugins* in particular).

Note: Notice the special plugin to integrate GFAL2 aditional plugins for data transfer protocols in addition to ARC native DMCs. For example, to add support for legacy LFC file catalogue protocol to ARC you have to install `nordugrid-arc-plugins-gfal` and `gfal2-plugin-lfc`.

This applies both to client tools and *ARC Data Staging*.

7.4.1 Data transfer URLs

File locations in ARC can be specified both as local file names, and as Internet standard *Uniform Resource Locators (URL)*. There are also some additional URL *options* that can be used.

Depending on the installed ARC components some or all of the following transfer protocols and metadata services are supported:

Table 7.4: List of main supported protocols

Protocol	Description
`ftp`	ordinary *File Transfer Protocol (FTP)*
`gsiftp`	GridFTP, the Globus -enhanced FTP protocol with security, encryption, etc. developed by The Globus Alliance
`http`	ordinary *Hyper-Text Transfer Protocol (HTTP)* with PUT and GET methods using multiple streams
`https`	HTTP with SSL
`httpg`	HTTP with Globus GSI
`dav`	WebDAV
`davs`	WebDAV with SSL
`ldap`	ordinary *Lightweight Data Access Protocol (LDAP)*
`srm`	Storage Resource Manager (SRM) service
`root`	Xrootd protocol
`rucio`	Rucio – a data management system used by ATLAS and other scientific experiments
`acix`	ARC Cache Index
`s3`	Amazon S3
`file`	local to the host file name with a full path

An URL can be used in a standard form, i.e.

```
protocol://[host[:port]]/file
```

Or, to enhance the performance or take advantage of various features, it can have additional options:

```
protocol://[host[:port]][;option[;option[...]]]/file[:metadataoption[:metadataoption[.
↪..]]
```

For a metadata service URL, construction is the following:

```
protocol://[url[|url[...]]@]host[:port][;option[;option[...]]]
  /lfn[:metadataoption[:metadataoption[...]]]
```

where the nested URL(s) are physical replicas. Options are passed on to all replicas, but if it is desired to use the same option with a different value for all replicas, the option can be specified as a common option using the following syntax:

```
protocol://[;commonoption[;commonoption]|][url[|url[...]]@]host[:port]
  [;option[;option[...]]/lfn[:metadataoption[:metadataoption[...]]]
```

In user-level tools, URLs may be expressed using this syntax, or there may be simpler ways to construct complex URLs. In particular, command line tools such as `arccp`, and the xRSL languages provide methods to express URLs and options in a simpler way.

For the SRM service, the syntax is:

```
srm://host[:port][;options]/[service_path?SFN=]file[:metadataoptions]
```

Versions 1.1 and 2.2 of the SRM protocol are supported. The default *service_path* is `srm/managerv2` when the server supports v2.2, `srm/managerv1` otherwise.

For Rucio the following URL is used to look up replicas of the given scope and name:

```
rucio://rucio-lb-prod.cern.ch/replicas/scope/name
```

The Rucio authorisation URL can be specified with the environment variable `$RUCIO_AUTH_URL`. The Rucio account to use can be specified either through the `rucioaccount` URL option or `$RUCIO_ACCOUNT` environment variable. If neither are specified the account is taken from the VOMS `nickname` attribute.

For ACIX the URLs look like:

```
acix://cacheindex.ndgf.org:6443/data/index?url=http://host.org/file1
```

S3 authentication is done through keys which must be set by the environment variables `$S3_ACCESS_KEY` and `$S3_SECRET_KEY`.

The URL components are:

`host[:port]`	Hostname or IP address [and port] of a server
`lfn`	Logical File Name
`url`	URL of the file as registered in indexing service
`service_path`	End-point path of the web service
`file`	File name with full path
`option`	URL option
`commonoption`	URL option for all replicas
`metadataoption`	Metadata option

The following URL options are supported:

<table>
<tr><td>threads=<number></td><td>specifies number of parallel streams to be used by GridFTP or HTTP(s,g); default value is 1, maximal value is 10</td></tr>
<tr><td>exec=yes|no</td><td>means the file should be treated as executable</td></tr>
<tr><td>preserve=yes|no</td><td>specify if file must be uploaded to this destination even if job processing failed (default is no)</td></tr>
<tr><td>cache=yes|no|renew|copy|c</td><td>indicates whether the file should be cached; default for input files in A-REX is yes. renew forces a download of the file, even if the cached copy is still valid. copy forces the cached file to be copied (rather than linked) to the session directory, this is useful if for example the file is to be modified. check forces a check of the permission and modification time against the original source. invariant disables checking the original source modification time.</td></tr>
<tr><td>readonly=yes|no</td><td>for transfers to file:// destinations, specifies whether the file should be read-only (unmodifiable) or not; default is yes</td></tr>
<tr><td>secure=yes|no</td><td>indicates whether the GridFTP data channel should be encrypted; default is no</td></tr>
<tr><td>blocksize=<number></td><td>specifies size of chunks/blocks/buffers used in GridFTP or HTTP(s,g) transactions; default is protocol dependent</td></tr>
<tr><td>checksum=cksum|md5|adler3</td><td>specifies the algorithm for checksum to be computed (for transfer verification or provided to the indexing server). This is overridden by any metadata options specified (see below). If this option is not provided, the default for the protocol is used. checksum=no disables checksum calculation.</td></tr>
<tr><td>overwrite=yes|no</td><td>makes software trying (or not) to overwrite existing file(s); if yes, the tool will try to remove any information/content associated with the specified URL before writing to the destination.</td></tr>
<tr><td>protocol=gsi|gssapi|ssl|t</td><td>distinguishes between different kinds of HTTPS/HTTPG and SRM protocols. Here gssapi stands for HTTPG implementation using only GSSAPI functions to wrap data and gsi uses additional headers as implemented in Globus IO. The ssl and tls options stand for the usual HTTPS and are specifically usable only if used with the SRM protocol. The ssl3 option is mostly the same as the ssl one but uses SSLv3 handshakes while establishing HTTPS connections. The default is gssapi for SRM connections, tls for HTTPS and``gssapi`` for HTTPG. In the case of SRM, if default fails, gsi is tried.</td></tr>
<tr><td>spacetoken=<pattern></td><td>specifies a space token to be used for uploads to SRM storage elements supporting SRM version 2.2 or higher</td></tr>
<tr><td>autodir=yes|no</td><td>specifies whether before writing to the specified location the software should try to create all directories mentioned in the specified URL. Currently this applies to FTP and GridFTP only. Default value for these protocols is yes</td></tr>
<tr><td>tcpnodelay=yes|no</td><td>controls the use of the TCP_NODELAY socket option (which disables Nagle's algorithm). Applies to HTTP(S) only. Default is no (supported only in arcls and other arc* tools)</td></tr>
<tr><td>transferprotocol=protocol</td><td>specifies transfer protocols for meta-URLs such as SRM. Multiple protocols can be specified as a comma-separated list in order of preference.</td></tr>
<tr><td>rucioaccount=account</td><td>specifies the Rucio account to use when authenticating with Rucio.</td></tr>
<tr><td>httpputpartial=yes|no</td><td>while storing a file on a HTTP(S) server, the software will try to send it in chunks/parts. If the server reports error for the partial PUT command, the software will fall back to transferring the file in a single piece. This behavior is non-standard and not all servers report errors properly. Hence the default is a safer no.</td></tr>
<tr><td>httpgetpartial=yes|no</td><td>while retrieving a file from a HTTP(S) server, the software will try to read it in chunks/parts. If the server does not support the partial GET command, it usually ignores requests for partial transfer range and the file is transfered in one piece. Default is yes.</td></tr>
<tr><td>failureallowed=yes|no</td><td>if set to yes for a job input or output file, then a failure to transfer this file will not cause a failure of the job. Default is no.</td></tr>
<tr><td>relativeuri=yes|no</td><td>if set to yes, HTTP operations will use the path instead of the full URL. Default is no.</td></tr>
<tr><td>accesslatency=disk|tape</td><td>filter replicas returned from an index service based on their access latency. (available from version 6.12)</td></tr>
</table>

Local files are referred to by specifying either a location relative to the job submission working directory, or by an absolute path (the one that starts with /), preceded with a `file://` prefix.

URLs also support metadata options which can be used for registering additional metadata attributes or querying the service using metadata attributes. These options are specified at the end of the LFN and consist of name and value pairs separated by colons. The following attributes are supported:

`checksumtype`	Type of checksum. Supported values are `cksum` (default), `md5` and `adler32`
`checksumvalue`	The checksum of the file

The checksum attributes may also be used to validate files that were uploaded to remote storage.

Examples of URLs are:

- `http://grid.domain.org/dir/script.sh`
- `gsiftp://grid.domain.org:2811;threads=10;secure=yes/dir/input_12378.dat`
- `ldap://grid.domain.org:389/lc=collection1,rc=Nordugrid,dc=nordugrid,dc=org`
- `file:///home/auser/griddir/steer.cra`
- `srm://srm.domain.org/griddir/user/file1:checksumtype=adler32:checksumvalue=12345678`[1]
- `srm://srm.domain.org;transferprotocol=https/data/file2`[2]
- `rucio://rucio-lb-prod.cern.ch/replicas/user.grid/data.root`

7.4.2 Stage-in during submission

During the job submission to computing element, data can be moved from the client machine along with the job description.

The *inputFiles* directive of the job description automatically activates this kind of data movement if `source` of the data is a local path or empty (`""`). An empty source value means that input `filename` is taken from the current working directory on the submission machine.

For example:

```
(inputFiles=("data1" "/mnt/data/analyses/data11.22.33")
            ("data2" ""))
```

During the job submission process the `/mnt/data/analyses/data11.22.33` file will be uploaded to the job session directoty on the CE as `data1`. The `data2` file from the current working directory will be uploaded as `data2`.

7.4.3 Stage-in on ARC CE

Instead of copying data from submission machine ARC CE can download it from the available storage services specified by *URL* in the *inputFiles* source value.

In this case all the power of the *A-REX Data Caching* can be used as well.

For example:

```
(inputfiles = "data.root.1" "srm://srm.mystorage.example.org/atlas/disk/data11_7TeV/
↪data.root.1")
```

During job submission the data WILL NOT be uploaded. Instead, A-REX will analyze the job description and starts a Stage-In process. During Stage-in the `data.root.1` file will be downloaded from provided SRM URL.

[1] This is a destination URL. The file will be copied to `srm.domain.org` at the path `griddir/user/file1` and the checksum will be compared to what is reported by the SRM service after the transfer.

[2] This is a source or destination URL. When getting a transport URL from SRM, the HTTPS transfer protocol will be requested.

7.4.4 Manual data movement with arc* tools

There is a set of *data management tools* that can be used to manipulate data manually, out of jobs processing context.

`arcls`, `arccp`, `arcrm` and `arcmkdir` work similar to classic Unix data movement commands but accept local or remote URLs.

Again, *any URL* supported by installed ARC data management plugins can be passed to the tools as an argument.

```
[user ~]$ arccp srm://srm.mystorage.example.org/atlas/disk/data11_7TeV/data.root.1 /
↪mnt/data/data.root.1

[user ~]$ arcls http://download.nordugrid.org/repos/6
centos/
debian/
fedora/
ubuntu/
```

7.5 Job Description Language (xRSL)

In the distributed computing job are submitted to a large number of very different computing resources, which are often widely distributed geographically.

In order to abstract from the heterogeneous nature of these resources, a high-level job description language is necessary. This document introduces the xRSL job description language.

Note: The EMI Activity Description Language (ADL) is also supported by ARC (refer to section 9.3 of EMI-ES document for complete ADL specification)

The diversity of resources implies special requirements for a proper description of a job, introducing many new options as opposed to a conventional high-performance computing center use case.

The NorduGrid project introduced the **extended Resource Specification Language (xRSL)**. xRSL adopts the general syntax of RSL language developed by Globus Alliance but extend and redesign some attributes and introduce the two levels of job option specifications:

- **User-side xRSL**, i.e., the set of attributes specified by a user in a job-specific file. This file is interpreted by a *Client* and after the necessary modifications is passed to the ARC computing service – A-REX
- **Server-side xRSL**, i.e., the set of attributes pre-processed by a client, and ready to be interpreted by the A-REX. Effectively, this is an *internal* job representation of ARC.

A user only has to know the user-side part, and utilize it to describe the grid jobs. A-REX, however, uses slightly different notations, supplied by the client tools; therefore developers of such tools must take care of converting user-submitted xRSL to the internal job description (Server-side xRSL).

In what follows, description of the xRSL is given, using the following notations:

<xxxx>	parameter to be substituted with a corresponding string or a number
[xxxx]	optional parameter
xxx|yyy|zzz	list of possible values of a parameter
-"-	same as above

7.5.1 xRSL syntax and rules

A job is described by means of xRSL attributes, which can be either passed via a command-line, or, more conveniently, be collected in a so-called xRSL-file (`*.xrsl`).

Such a file contains a plain list of attribute-value pairs and boolean operators `&` (for AND) and `|` (for OR). Attribute names are **case-insensitive**.

If the attributes are specified from the command line, the entire description must be enclosed either in single or in double quotes. Single quotes enclosure is completely analogous to xRSL-file usage, while double quotes imply standard shell expansion of the enclosed string. This has implications when strings contain local shell variables: they **will not be expanded** unless the task description is entered from the command line and is enclosed in double quotes.

Attribute value

An attribute-value pair is a key element of specification. It consists of an expression that assigns one or more values to an attribute, and is enclosed in round brackets:

```
(attribute="value")
```

For multi-valued attributes:

```
(attribute="value1" "value2")
```

List of values

Certain attributes do not have assigned value; instead, they contain a list of values that should be put in proper correspondence with each other:

```
(attribute=("value1" "value2")("value3" "value4"))
```

In the example above, `value1` and `value3` are put in correspondence to `value2` and `value4` respectively, according to the context of the attribute.

Quoting

Values should be enclosed in quotes if they contain blank spaces or special characters.

The special characters are:

```
+   &   |   (   )   =   <   >   !   "   '   ^   #   $
```

To quote a string containing special characters, you can use either single or double quotes. If your string, however, contains both such quotes, you can define any character as an own delimiter, by preceding it with the "carat" (`^`) character: `attribute=^*My "good" value^*` makes use of a carat-escaped asterisk as a delimiter.

Implicit conjunction

An xRSL job description starts with an ampersand (&) , to indicate implicit conjunction of all the attributes:

```
&(attribute1=value1)(attribute2="value 2")...
```

Whenever a disjunct-request of two or more attributes is needed, the following construction can be used:

```
(|(attribute="value1")(attribute="value2")...)
```

Note: Only few selected attributes (indicated further in the document) can be requested by the user multiple times, like in the disjunct request example above. Most attributes **must be unique**, i.e., appear only once in the job description document.

Operators

In expressions, the following relational operators are allowed, in general:

```
=    !=    >    <    >=    <=
```

However, most attributes can *only be used with equality operator* `=`. For few attributes (as indicated in the document), some other operators can be used in client-side job description as well.

Blank spaces

The xRSL attributes can be written in a single string, or split in lines arbitrary; blank spaces between and inside `(attribute="value")` relations are ignored.

Comments

Commented lines should start with `(*` and be closed with `*)`:

```
(*attribute="value1"*)
```

Comments can not be nested.

Multible jobs in one description

Multiple job description in one file can be specified via multi-request operator `+`, which should precede multiple job descriptions:

```
+(&(...))(&(...))(&(...))
```

7.5.2 User-side xRSL attributes

The following attributes can be specified in a user's xRSL script. Some have to be modified by the client tool before being passed to the A-REX.

Attribute names are case-insensitive, although assigned values may well be case-sensitive, if they represent file names, environment variables etc.

Note: It is possible to use unsupported attributes in job description. Standard ARC client submission commands (`arcsub` and `arcresub`) must be used with a command line option `-U` in order to accept unknown attributes.

Without this command line option, the client tool will concider job description invalid if it contains unsupported attributes.

---

executable

Unique:	yes
Operators:	=
User xRSL:	`(executable=<string>)`
A-REX xRSL:	`-"-`
Example:	`(executable="local_to_job.exe")`

The executable to be submitted as a main task to a Local Resource Management System (LRMS).

- `string` - file name (including path), local to the computing element (CE)

Executable is a file that has to be executed as the main process of the task. It could be either a pre-compiled binary, or a script. Users may transfer their own executables, or use the ones known to be already installed on the remote system (CE).

If an executable has to be transferred to the destination site (CE) from some source, it has to be specified in the `inputFiles` list. If it is not specified in `inputFiles`, the source is expected to be local to the user (client) and will be added as such to the `inputFiles` list by the ARC Client.

If the file name starts with a leading slash (/), it is considered to be **the full path to the executable at the destination site (CE)**; otherwise the location of the file is **relative** to the session directory (where job input and files are stored).

If the xRSL string is entered from the command line and is enclosed in double quotes, standard shell expansion of variables takes place. That is, if the file name contains an environment variable (`$...`), the value of this variable is resolved locally, but if the name itself is also enclosed in double quotes, it will be resolved at the remote computing element:
`(executable=$ROOT_DIR/myprog.exe)` – `$ROOT_DIR` is resolved locally (*will cause errors if the path does not exist at the execution machine*)
`(executable="$ROOT_DIR/myprog.exe")` – `$ROOT_DIR` will be resolved remotely

To use executable at the destination site without specifying absolute path a trivial wrapper script can be used as the executable input file:

```
#!/bin/sh

# load local environment
module load openmpi/latest

# pass-through all the arguments
mpirun "$@"
```

Other option is to use dummy environment variable:
`(executable="${PROG_PATH}${PROG_PATH:+/}myprog.exe")` – `$PROG_PATH` will be resolved remotely and possibly provided by relevant RTE or be an empty string when left undefined. In the latter case `myprog.exe` will be searched in the `$PATH` list on the destination site (CE).

arguments

Unique:	yes
Operators:	=
User xRSL:	`(arguments=<string> [string] ... )`
A-REX xRSL:	`(arguments=<executable> <string> [string] ... )`
Example:	`(arguments="10000" $(ATLAS)/input.dat)`

List of the arguments for the executable.

- `string` - an argument
- `executable` - the executable to be run by LRMS, taken by the ARC Client from the user-specified `executable` attribute

inputFiles

Unique:	yes
Operators:	=
User xRSL:	`(inputFiles=(<filename> <source> [option] ... ) ... )`
A-REX xRSL:	`(inputFiles=(<filename> <URL> [option] ... )` `(<filename> [size][.checksum]) ... )`

Example:	`(inputFiles=("local_to_job" "gsiftp://se1.lu.se/p1/remote.1" "threads=5")`
	`("local_to_job.dat" "/scratch/local_to_me.dat")`
	`("same_name_as_in_my_current_dir" ""))`

List of files to be copied to the computing element before job execution.

- `filename` - destination file name, local to the computing element and always relative to the session directory
- `source` - source of the file: (remote URLs, or a path, local to the submission node). If void (`""`, use the quotes!), the input file is taken from the submission directory.
- `option` - URL options for source. The ARC Client converts `source` and any options given here to a URL with the syntax described *here*. Additionally `delegationid=<id>` can be specified here to assign X.509 delegation identifier to be used for corresponding URL.
- `URL` - URL of the file
- `size` - file size in bytes
- `checksum` - file checksum (as returned by `cksum`)

If the `inputFiles` list does not contain the standard input file (as specified by `stdin`) and/or the executable file (as specified by `executable`), an ARC client must append these files to the list. If the `<source>` is a URL, any options given by `option` are added to it, then it is passed by the ARC Client to the A-REX as shown in the example above. A-REX recognizes all URLs except `file:///`.

Internally, the client must forward the `(<filename> <source> [option] ...)` request to the execution service without changes, unless `<source>` is a local path, void (`""`) or `file:///`. In case `<source>` is a local path, void (`""`) or `file:///`, the client must extract file size and checksum, and substitute the `<source>` string with `[size][.checksum]`. In the unlikely case when it is impossible to extract file size, the `<source>` string must be substituted by a void one (`""`).

Note: Please note that the `inputFiles` attribute is not meant to operate with directories, for reasons of access control and checksum verifications. You must specify a pair `("<local_to_job>" "<source>")` for each file.

executables

Unique:	yes
Operators:	=
User xRSL:	`(executables=<string> [string] ...)`
A-REX xRSL:	`-"-`
Example:	`(executables="myscript.sh" "myjob.exe")`

List of files from the `inputFiles` set, which will be given executable permissions.

- `string` - file name, local to the computing element and relative to the session directory

If the executable file (as specified in `executable` and if relative to the session directory) is not in the `executables` list, it will be added to the list by the ARC Client.

cache

Unique:	yes
Operators:	=
User xRSL:	`(cache="yes"|"no")`
A-REX xRSL:	`-"-`
Example:	`(cache="yes")`

Specifies whether input files specified in the `inputFiles` should be placed by default in the cache or not. This affects all input files, even those described by `executables`.

If not specified, default value is `yes`.

Note: Cached files can not be modified by jobs by default. If your job has to modify input files, please use the `(readonly=no)` URL option for those files. This option does not affect whether or not the file is cached.

outputFiles

Unique:	yes
Operators:	=
User xRSL:	`(outputFiles=(<string> <URL> [option] ... ) ... )`
A-REX xRSL:	`(outputFiles=(<string> <URL>) ... )`
Example:	`(outputFiles=("local_to_job.dat" "gsiftp://se1.uio.no/stored.dat") ("local_to_job_dir/" ""))`

List of files to be retrieved by the user or uploaded by the A-REX and optionally indexed (registered) in a data indexing service.

- `string` - file name, local to the *Computing Element (CE)*. If this string ends with a backslash `/` and `<URL>` is empty, the entire directory will be kept at the execution site. If however this string ends with a backslash `/` but the `<URL>` is a remote location, the contents of the directory are transferred to the destination.
- `URL` - destination URL of the remote file; if void (`""`, use the quotes!), the file is kept for manual retrieval. Note that this can not be a local `file://` URL.
- `option` - URL options for destination URL. See *URL Options* for possible values. When the destination is an indexing service, a physical file location may be specified by the additional option `location`. This option can be given multiple times. The CE will attempt to upload the file to the specified locations in the order they are given until one succeeds. Options specified after a location option only affect that location. Another additional `delegationid=<id>` option can be specified here to assign X.509 delegation identifier to be used for corresponding URL.Before passing to the A-REX, the ARC Client adds to `URL` any options and locations given here, using the syntax described in *URL Options*.

If the list does not contain standard output, standard error file names and A-REX log-files directory name (as specified by `stdout`, `stderr` and `gmlog`), the ARC Client appends these items to the `outputFiles` list. If the `<URL>` is not specified (void, `""`, use the quotes!), files will be kept on the CE and should be downloaded by the user via the ARC Client. If specified name of file ends with `/`, the entire directory is kept.

A convenient way to keep the entire job directory at the remote site for a manual retrieval is to specify `(outputfiles=(/ ""))`.

In some cases, the list of output files may only be known after the job has completed. ARC allows a user to specify a list of output files dynamically in a file or files in the session directory as part of their job. The file(s) containing the output file information can be defined in the xRSL script as the path to the file relative to the session directory preceeded by '@'. The format of these files is lines of 2 values separated by a space. The first value contains name of the output file relative to the session directory and the second value is a URL to which the file will be uploaded.

Example:	`(outputFiles=("@output.files" ""))`
	output.files is generated by the user and contains
	`file1 gsiftp://grid.domain.org/file1`
	`file2 gsiftp://grid.domain.org/file2`

After the job completes, the file output.files in the session directory will be read and any files described within will be uploaded to the given URLs.

cpuTime

Unique:	yes
Operators:	=
User xRSL:	`(cpuTime=<time>)`
A-REX xRSL:	`(cpuTime=<tttt>)`
Example:	`(cpuTime="240")`

Maximal CPU time request for the job. For a multi-processor job, this is a sum over all requested processors.

- `time` - time (in minutes if no unit is specified)
- `tttt` - time converted by the ARC Client from `time` to seconds.

The client converts time specified in the user-side XRSL file to seconds. If no time unit is specified, the client assumes the time given in minutes. Otherwise, a text format is accepted, i.e., any of the following will be interpreted properly (make sure to enclose such strings in quotes!):

```
1 week
3 days
2 days, 12 hours
1 hour, 30 minutes
36 hours
9 days
240 minutes
```

If both `cpuTime` and `wallTime` are specified, the ARC Client converts them both. `cpuTime` can not be specified together with `gridTime` or `benchmarks`.

Note: This attribute should be used to direct the job to a system with sufficient CPU resources, typically, a batch queue with the sufficient upper time limit. Jobs exceeding this maximum most likely will be **terminated** by remote systems! If time limits are not specified, the limit is not set and jobs can run as long as the system settings allow (note that in this case you can not avoid queues with too short time limits).

wallTime

Unique:	yes
Operators:	=
User xRSL:	`(wallTime=<time>)`
A-REX xRSL:	`(wallTime=<tttt>)`
Example:	`(wallTime="240")`

Maximal wall clock time request for the job.

- `time` - time (in minutes if no unit is specified)
- `tttt` - time converted by the ARC Client to seconds

The client converts time specified in the user-side XRSL file seconds. If no time unit is specified, the client assumes the time given in minutes. Otherwise, a text format is accepted, i.e., any of the following will be interpreted properly (make sure to enclose such strings in quotes!):

```
1 week
```

```
3 days
2 days, 12 hours
1 hour, 30 minutes
36 hours
9 days
240 minutes
```

If both `cpuTime` and `wallTime` are specified, the ARC Client converts them both. `wallTime` can not be specified together with `gridTime` or `benchmarks`. If only `wallTime` is specified, but not `cpuTime`, the corresponding `cpuTime` value is evaluated by the ARC Client and added to the job description.

Note: This attribute should be used to direct the job to a system with sufficient CPU resources, typically, a batch queue with the sufficient upper time limit. Jobs exceeding this maximum most likely will be **terminated** by remote systems! If time limits are not specified, the limit is not set and jobs can run as long as the system settings allow (note that in this case you can not avoid queues with too short time limits).

gridTime

Unique:	yes
Operators:	=
User xRSL:	`(gridTime=<time>)`
A-REX xRSL:	none
Example:	`(gridTime="2 h")`

Maximal CPU time request for the job scaled to the 2.8 GHz Intel Pentium 4 processor.

- `time` - time (in minutes if no unit is specified)

The attribute is completely analogous to `cpuTime`, except that it will be recalculated to the actual CPU time request for each queue, depending on the published processor clock speed.

`gridTime` can not be specified together with `cpuTime` or `wallTime`. If only `gridTime` is specified, but not `cpuTime`, the corresponding `cpuTime` value is evaluated by the ARC Client and added to the job description.

benchmarks

Unique:	yes
Operators:	=
User xRSL:	`(benchmarks=(<string> <value> <time>) ... )`
A-REX xRSL:	
Example:	`(benchmarks=("mybenchmark" "10" "1 hour, 30 minutes"))`

Evaluate a job's `cpuTime` based on benchmark values.

- `string` - benchmark name
- `value` - benchmark value of reference machine
- `time` - the `cpuTime` the job requires on the reference machine

`benchmarks` can not be specified together with `cpuTime` or `wallTime`. If only `benchmarks` is specified, but not `cpuTime`, the corresponding `cpuTime` value is evaluated by the ARC Client and added to the job description.

memory

Unique:	yes
Operators:	=
User xRSL:	`(memory=<integer>)`
A-REX xRSL:	`-"-`
Example:	`(memory>="500")`

Memory required for the job, per count for parallel jobs.

- `integer` - size (Mbytes)

Note: Similarly to `cpuTime`, this attribute should be used to direct a job to a resource with a sufficient capacity. Jobs exceeding this memory limit will most likely be **terminated** by the remote system.

disk

Unique:	no
Operators:	`= != > < >= <=`
User xRSL:	`(disk=<integer>)`
A-REX xRSL:	none
Example:	`(disk="500")`

Disk space required for the job.

- `integer` - disk space, Mbytes

Note: This attribute is used at the job submission time to find a system with sufficient disk space. However, it **does not guarantee** that this space will be available at the end of the job, as most known systems do not allow for disk space allocation. Eventually, a remote system can terminate a job that exceeds the requested disk space.

runTimeEnvironment

Required runtime environment.

- `string` - environment name

The site to submit the job to will be chosen by the ARC Client among those advertising specified runtime environments. Before starting the job, the A-REX will set up environment variables and paths according to those requested. Runtime environment names are defined by Virtual Organizations, and tend to be organized in name spaces.

To request several environments, repeat the attribute string:
`(runTimeEnvironment="ENV1")(runTimeEnvironment="ENV2")` etc.

To make a disjunct-request, use a boolean expression:
`(|(runTimeEnvironment="env1")(runTimeEnvironment="env2"))`.

You can use >= or <= operators: job will be submitted to any suitable site that satisfies such requirements, and among the available at the sites runtime environments, the highest version satisfying a requirement will be requested in the pre-processed xRSL script.

Runtime environment string interpretation is case-insensitive. If a runtime environment string consists of a name and a version number, a partial specification is possible: it is sufficient to request only the name and use > or >= operators to select the highest version.

The optional arguments can be supplied as additional strings after runtime environment string. On the server side they are passed to software responsible for preparing corresponding infrastructure.

middleware

Unique:	no
Operators:	`= != > < >= <=`
User xRSL:	`(middleware=<string>)`
A-REX xRSL:	only = is allowed
Example:	`(middleware="nordugrid-arc-0.5.99")`

Required middleware version. Make sure to specify full name and version number.

`string`	Grid middleware name.

The site to submit the job to will be chosen by the ARC Client among those advertising specified middleware. Usage is identical to that of the `runTimeEnvironment`. Use the >= operator to request a version “equal or higher”.

opsys

Unique:	no
Operators:	`= != > < >= <=`
User xRSL:	`(opsys=<string>)`
A-REX xRSL:	only = is allowed
Example:	`(opsys="FC3")`

Required operating system.

`string`	Operating system name and version.

The site to submit the job to will be chosen by the ARC Client among those advertising specified operating system. Usage is identical to that of `runTimeEnvironment` and `middleware`. Use the >= operator to request a version “equal or higher”.

stdin

Unique:	yes
Operators:	=
User xRSL:	`(stdin=<string>)`
A-REX xRSL:	`-"-`
Example:	`(stdin="myinput.dat")`

The standard input file.

`string`	file name, local to the computing element.

The standard input file should be listed in the `inputFiles` attribute; otherwise it will be forced to that list by the ARC Client.

stdout

Unique:	yes
Operators:	=
User xRSL:	`(stdout=<string>)`
A-REX xRSL:	`-"-`
Example:	`(stdout="myoutput.txt")`

The standard output file.

`string`	file name, local to the computing element and relative to the session directory.

The standard output file should be listed in the `outputFiles` attribute; otherwise it will be forced to that list by the ARC Client. If the standard output is not defined, ARC Client assigns a name.

stderr

Unique:	yes
Operators:	=
User xRSL:	`(stderr=<string>)`
A-REX xRSL:	`-"-`
Example:	`(stderr="myjob.err")`

The standard error file.

`string`	file name, local to the computing element and relative to the session directory.

The standard error file should be listed as an `outputFiles` attribute; otherwise it will be forced to that list by the ARC Client. If the standard error is not defined, ARC Client assigns a name. If `join` is specified with value `yes`, ARC Client adds `stderr` to the pre-processed xRSL script with the same value as `stdout`.

join

Unique:	yes
Operators:	=
User xRSL:	`(join="yes"|"no")`
A-REX xRSL:	none
Example:	`(join="yes")`

If `yes`, joins `stderr` and `stdout` files into the `stdout` one. Default is `no`.

gmlog

Unique:	yes
Operators:	=
User xRSL:	`(gmlog=<string>)`
A-REX xRSL:	`-"-`
Example:	`(gmlog="myjob.log")`

A name of the directory containing grid-specific diagnostics per job.

`string`	a directory, local to the computing element and relative to the session directory

This directory is kept in the session directory to be available for retrieval (ARC Client forces it to the list if `outputFiles`)

jobName

Unique:	yes
Operators:	=
User xRSL:	`(jobName=<string>)`
A-REX xRSL:	`-"-`
Example:	`(jobName="My Job nr. 1")`

User-specified job name.

`string`	job name

This name is meant for convenience of the user. It can be used to select the job while using the ARC Client. It is also available through the Information System.

ftpThreads

Unique:	yes
Operators:	=
User xRSL:	`(ftpThreads=<integer>)`
A-REX xRSL:	`-"-`
Example:	`(ftpThreads="4")`

Defines how many parallel streams will be used by the A-REX during `gsiftp` and `http(s|g)`transfers of files.

`integer`	a number from 1 to 10

If not specified, parallelism is not used.

acl

Unique:	no
Operators:	=
User xRSL:	`(acl=<xml>)`
A-REX xRSL:	`-"-`
Example:	`(acl="<?xml version=""1.0""?> <gacl version=""0.0.1""><entry><any-user></any-user> <allow><write/><read/><list/><admin/></allow></entry></gacl>")`

Makes use of GACL rules to list users who are allowed to access and control job in addition to job's owner. Access and control levels are specified per user. `any-user` tag refers to any user authorized at the execution cluster. To get more information about GACL please refer to http://www.gridsite.org.

`xml`	a GACL-compliant XML string defining access control list

Following job control levels can be specified via `acl`:

`write`	allows to modify contents of job data (job directory) and control job flow (cancel, clean, etc.)
`read`	allows to read content of job data (contents of job directory)
`list`	allows to list files available for the job (contents of job directory)
`admin`	allows to do everything – full equivalence to job ownership

queue

Unique:	yes
Operators:	`=` `!=`
User xRSL:	`(queue=<string>)`
A-REX xRSL:	only = is allowed
Example:	`(queue="pclong")`

The name of the remote batch queue.

Use only when you are sure that the queue by this name does exist.

`string`	known queue name

While users are not expected to specify `queue` in job descriptions, this attribute **must** be present in the Server-side xRSL. In fact, this is primarily an internal attribute, added to the job description by client tools after resource discovery and matchmaking. Still, users can specify this attribute to explicitly force job submission to a queue: when specified explicitly by the user, this value will not be overwritten by the ARC Client, and an attempt will be made to submit the job to the specified queue.

If for some reason (e.g. due to a client tool error) `queue` is absent from the Server-side xRSL, A-REX on the selected cluster will attempt to submit the job to the default queue if such is specified in the A-REX configuration.

startTime

Unique:	yes
Operators:	=
User xRSL:	`(startTime=<time>)`
A-REX xRSL:	`(startTime=<tttt>)`
Example:	`(startTime="2002-05-25 21:30:00")`

Time to start job processing by the Grid Manager, such as e.g. start downloading input files.

`time`	time string, YYYY-MM-DD hh:mm:ss
`tttt`	time string, YYYYMMDDhhmmss[Z] (converted by the ARC Client from `time`)

Actual job processing on a worker node starts depending on local scheduling mechanisms, but not sooner than `startTime`.

lifeTime

Unique:	yes
Operators:	=
User xRSL:	`(lifeTime=<time>)`
A-REX xRSL:	`(lifeTime=<tttt>)`
Example:	`(lifeTime="2 weeks")`

Maximal time to keep job files (the session directory) on the gatekeeper upon job completion.

`time`	time (in minutes if no unit is specified)
`tttt`	time (seconds, converted by the ARC Client from `time`)

Typical life time is 1 day (24 hours). Specified life time can not exceed local settings.

notify

Unique:	yes
Operators:	=
User xRSL:	`(notify=<string> [string] ... )`
A-REX xRSL:	`-"-`
Example:	`(notify="be your.name@your.domain.com")`

Request e-mail notifications on job status change.

`string`	string of the format: `[b][q][f][e][c][d] user1@domain1 [user2@domain2] ...`

Here flags indicating the job status are:

`b`	begin (PREPARING)
`q`	queued (INLRMS)
`f`	finalizing (FINISHING)
`e`	end (FINISHED)
`c`	cancellation (CANCELLED)
`d`	deleted (DELETED)

When no notification flags are specified, default value of `eb` will be used, i.e., notifications will be sent at the job's beginning and at its end.

No more than 3 e-mail addresses per status change accepted.

rerun

Unique:	yes
Operators:	=
User xRSL:	`(rerun=<integer>)`
A-REX xRSL:	`-"-`
Example:	`(rerun="2")`

Number of reruns (if a system failure occurs).

`integer`	an integer number

If not specified, the default is 0. Default maximal allowed value is 5. The job may be rerun after failure in any state for which reruning has sense. To initiate rerun user has to use the `arcresume` command.

architecture

Unique:	no
Operators:	`=` `!=`
User xRSL:	`(architecture=<string>)`
A-REX xRSL:	none
Example:	`(architecture="i686")`

Request a specific architecture.

`string`	architecture (e.g., as produced by uname -a)

nodeAccess

Unique:	yes
Operators:	`=`
User xRSL:	`(nodeAccess="inbound"|"outbound")`
A-REX xRSL:	none
Example:	`(nodeAccess="inbound")`

Request cluster nodes with inbound or outbound IP connectivity. If both are needed, a conjunct request should be specified.

dryRun

Unique:	yes
Operators:	`=`
User xRSL:	`(dryRun="yes"|"no")`
A-REX xRSL:	`-"-`
Example:	`(dryRun="yes")`

If `yes`, do dry-run: job description is sent to the optimal destination, input files are transferred, but no actual job submission to LRMS is made. Typically used for xRSL and communication validation.

rsl_substitution

Unique:	no
Operators:	`=`
User xRSL:	`(rsl_substitution=(<string1> <string2>))`
A-REX xRSL:	`-"-`
Example:	`(rsl_substitution=("ATLAS" "/opt/atlas"))`

Substitutes <`string2`> with <`string1`> for **internal** RSL use.

`string1`	new internal RSL variable
`string2`	any string, e.g., existing combination of variables or a path

Use this attribute to define variables that simplify xRSL editing, e.g. when same path is used in several values, typically in `inputFiles`. Only one pair per substitution is allowed. To request several substitution, concatenate such requests. Bear in mind that substitution must be defined **prior** to actual use of a new variable `string1`.

After the substitution is defined, it should be used in a way similar to shell variables in scripts: enclosed in round brackets, preceded with a dollar sign, **without quotes**:
`(inputfiles=("myfile" $(ATLAS)/data/somefile))`

Unlike the `environment` attribute, `rsl_substitution` definition is only used by the client and is valid inside xRSL script. It can not be used to define environment or shell variable at the execution site.

environment

Unique:	no
Operators:	=
User xRSL:	`(environment=(<VAR> <string>) [(<VAR> <string>)] ... )`
A-REX xRSL:	`-"-`
Example:	`(environment=("ATLSRC" "/opt/atlas/src") ("ALISRC" "/opt/alice/src"))`

Defines execution shell environment variables.

`VAR`	new variable name
`string`	any string, e.g., existing combination of variables or a path

Use this to define variables at an execution site. Unlike the `rsl_substitution` attribute, it can not be used to define variables on the client side.

count

Unique:	yes
Operators:	=
User xRSL:	`(count=<integer>)`
A-REX xRSL:	`-"-`
Example:	`(count="4")`

Specifies amount of sub-jobs to be submitted for parallel tasks.

countpernode

Unique:	yes
Operators:	=
User xRSL:	`(countpernode=<integer>)`
A-REX xRSL:	`-"-`
Example:	`(countpernode="2")`

Specifies amount of sub-jobs per node to be submitted for parallel tasks. Note: The `count` attribute must be specified when this attribute is specified.

exclusiveexecution

Unique:	yes
Operators:	=
User xRSL:	`(exclusiveexecution="yes"|"no")`
A-REX xRSL:	`-"-`
Example:	`(exclusiveexecution="yes")`

Specifies whether the node should be allocated for exclusive use by the job.

Note: for the setting to be valid and take effect one must in addition specify count and countpernode.

jobreport

Unique:	yes
Operators:	=
User xRSL:	`(jobreport=<URL>)`
A-REX xRSL:	`-"-`
Example:	`(jobreport="https://grid.uio.no:8001/logger")`

Specifies an URL for an accounting service to send reports about job to. The default is set up in the cluster configuration.

`URL`	URL

It is up to users to make sure the requested accounting service accepts reports from the set of clusters they intend to use.

credentialserver

Unique:	yes
Operators:	=
User xRSL:	`(credentialserver=<URL>)`
A-REX xRSL:	`-"-`
Example:	`(credentialserver="myproxy://myproxy.nordugrid.org;username=user")`

Specifies an URL which Grid Manager may contact to renew/extend delegated proxy of job. Only MyProxy servers are supported.

`URL`	URL of MyProxy server

It is up to a user to make sure the specified MyProxy server will accept requests from Grid Manager to renew expired credentials. `URL` may contain options `username` and `credname` to specify user name and credentials name which Grid Manager should pass to MyProxy server. If `username` is not specified DN of user credentials is used instead.

priority

Unique:	yes
Operators:	=
User xRSL:	`(priority=<integer>)`
A-REX xRSL:	`-"-`
Example:	`(priority="80")`

Specifies priority given to this job during staging of input and output files when the new data staging framework is used by A-REX. Values are limited to between 1 (lowest priority) and 100 (highest priority). Default if this attribute is not specified is 50.

7.5.3 Server-side attributes

Note: It is strongly advised to aviod specifying following attributes in the client xRSL.

The following attributes are a part of the internal ARC job representation, and must be provided by ARC client tools and passed to the A-REX. Developers of new ARC client tools and utilities must make sure these attributes are added to the user job description before it is submitted to a A-REX.

sstdin

A-REX xRSL:	`(sstdin=<filename>)`
Example:	`(sstdin="myinput.dat")`

Internal attribute for the standard input. Can also be spelled `stdinput`. Only needed for GRAM compatibility, not used by ARC as such.

- `filename` - standard input file name

action

A-REX xRSL:	`(action="request"|"cancel"|"clean"|"renew"|"restart")`
Example:	`(action="request")`

Action to be taken by the gatekeeper: submit the job, cancel job execution, clear the results of the job (also cancels the job), renew the proxy of the job, or restart the job from a previous failed state.

savestate

A-REX xRSL:	`(savestate="yes"|"no")`
Example:	`(savestate="yes")`

If `yes`, input RSL is stored in a temporary file at the gatekeeper. Must be always set as `yes` in the current implementation. Only needed for GRAM compatibility, not used by ARC as such.

lrmstype

A-REX xRSL:	`(lrmstype=<string>)`
Example:	`(lrmstype="pbs")`

LRMS type, indicating which submission script is to be invoked.

- `string` - LRMS type

hostName

A-REX xRSL:	`(hostname=<string>)`
Example:	`(hostName="grid.quark.lu.se")`

Name (e.g. as returned by the Linux `hostname` command) of the client machine from which the submission was made.

- `string` - client host name, as passed by the ARC client

jobid

A-REX xRSL:	`(jobid=<string>)`
Example:	`(jobid="grid.quark.lu.se:2119/jobmanager-ng/157111017133827")`

Unique job identification string, needed for cancellation and clean-up.

- `string` - global job ID

It can also be provided during submission of the job and should be unique to a computing element (cluster).

clientxrsl

A-REX xRSL:	`(clientxrsl=<string>)`
Example:	`(clientxrsl="&(executable=/bin/echo)(arguments=boo)")`

Job description xRSL string as submitted by the user, before being pre-processed by the client.

- `string` - original xRSL description submitted by the user

This attribute is added by the User Interface during pre-processing, and is used for job re-submission in order to repeat brokering and matchmaking.

clientsoftware

A-REX xRSL:	`(clientsoftware=<string>)`
Example:	`(clientsoftware="nordugrid-arc-0.5.39")`

Version of ARC client used to submit the job.

- `string` - version string

This attribute is added by the User Interface during pre-processing.

delegationid

A-REX xRSL:	`(delegationid=<id>)`
Example:	`(delegationid="a6f3e8fc920de023fbb432ea520")`

X.509 delegation identifier for all data staging operations and job itself.

- `id` - delegation identifier obtained during delegation procedure (for applicable job submission interfaces)

This attribute is added by the User Interface during pre-processing.

7.5.4 Examples

User-side xRSL script

```
&
(* test run: if "yes", only submits RSL without actual job start *)
     (dryRun="no")
(* some local variables defined for further convenience *)
     (rsl_substitution=("TOPDIR" "/home/johndoe"))
     (rsl_substitution=("NGTEST" $(TOPDIR)/ngtest))
     (rsl_substitution=("BIGFILE" "/scratch/johndoe/100mb.tmp"))
(* some environment variables, to be used by the job *)
     (environment=("ATLAS" "/opt/atlas") ("CERN" "/cern"))
(* the main executable file to be staged in and submitted to the PBS *)
     (executable="checkall.sh")
(* the arguments for the executable above *)
     (arguments="pal")
(* files to be staged in before the execution *)
     (inputFiles = ("be_kaons" "")
     ("file1" gsiftp://grid.uio.no$(TOPDIR)/remfile.txt)
      ("bigfile.dat" $(BIGFILE) )    )
(* files to be given executable permissions after staging in *)
     (executables="be_kaons")
(* files to be staged out after the execution *)
     (outputFiles=
        ("file1" "gsiftp://grid.tsl.uu.se/tmp/file1.tmp")
        ("100mb.tmp" "rls://rls.nordugrid.org:39281/test/bigfile")
        ("be_kaons.hbook"  gsiftp://ce1.grid.org$(NGTEST)/kaons.hbook)   )
(* user-specified job name *)
     (jobName="NGtest")
(* standard input file *)
     (stdin="myinput.dat")
(* standard output file *)
     (stdout="myoutput.dat")
(* standard error file *)
     (stderr="myerror.dat")
(* A-REX logs directory name *)
     (gmlog="gmlog")
(* flag whether to merge stdout and stderr *)
     (join="no")
(* request e-mail notification on status change *)
     (notify="bqfe john.doe@gmail.com jane.doe@mail.org")
(* maximal CPU time required for the job, minutes for PBS*)
     (CpuTime="60")
```

(continues on next page)

(continued from previous page)

```
(* maximal time for the session directory to exist on the remote node, days *)
     (lifeTime="7")
 (* memory required for the job, per count, Mbytes *)
     (Memory="200")
(* wall time to start job processing *)
     (startTime="2002-04-28 17:15:00")
(* disk space required for the job, Mbytes *)
     (Disk="500")
(* required architecture of the execution node *)
     (architecture="i686")
(* required run-time environment *)
     (runTimeEnvironment="APPS/HEP/Atlas-1.1")
(* number of re-runs, in case of a system failure *)
     (rerun="2")
```

Server-side xRSL script

Note that a client tool must do matchmaking and modify correspondingly the job document before submitting it to the matching resource. Specifically, a client tool has to:

- expand all the `rsl_substitution` values
- add double quotes to **all** strings
- insert queue attribute in case such is missing
- make sure every logical or comparison operator is expanded and replaced with a deterministic = statement
- streamline blank spaces

Note: Comment lines are removed from the Server-side xRSL! Below they are shown to explain details.

```
&
(* saves RSL in a temporary file if "yes" *)
     ("savestate" = "yes" )
(* job submission to be performed if action is "request" *)
     ("action" = "request" )
(* submission host name *)
     ("hostname" = "ce1.grid.org" )
(* client software version *)
     ("clientsoftware" = "nordugrid-arc-0.6.0.3" )
(* walltime value added by the client, in seconds *)
     ("walltime" = "3600" )
(* test run: if "yes", only submits RSL without actual job start *)
     ("dryRun" = "no" )
(* some local variables defined for further convenience *)
     ("rsl_substitution" = ("TOPDIR" "/home/johndoe" ) )
     ("rsl_substitution" = ("NGTEST" "/home/johndoe/ngtest" ) )
     ("rsl_substitution" = ("BIGFILE" "/scratch/johndoe/100mb.tmp" ) )
(* some environment variables, to be used by the job *)
     ("environment" = ("ATLAS" "/opt/atlas" ) ("CERN" "/cern" ) )
(* executable *)
     ("executable" = "checkall.sh" )
(* arguments *)
     ("arguments" = "pal" )
(* files to be staged in before the execution *)
```

(continues on next page)

(continued from previous page)

```
     ("inputfiles" = ("checkall.sh" "279320" )
        ("myinput.dat" "39806" )
        ("be_kaons" "8807" )
        ("file1" "gsiftp://grid.uio.no/home/johndoe/remfile.txt" )
        ("bigfile.dat" "104857600" )
     )
(* files to be given executable permissions after staging in *)
     ("executables" = "checkall.sh" "be_kaons" )
(* files to be staged out after the execution *)
     ("outputfiles" = ("file1" "gsiftp://grid.tsl.uu.se/tmp/file1.tmp" )
        ("100mb.tmp" "rls://rls.nordugrid.org:39281/test/bigfile" )
        ("be_kaons.hbook" "gsiftp://ce1.grid.org/home/johndoe/ngtest/kaons.hbook" )
        ("myoutput.dat" "" )
        ("myerror.dat" "" )
    )
(* user-specified job name *)
     ("jobName" = "NGtest" )
(* standard input file *)
     ("stdin" = "myinput.dat" )
(* standard output file *)
     ("stdout"="myoutput.dat" )
(* standard error file *)
     ("stderr" = "myerror.dat" )
(* flag whether to merge stdout and stderr *)
     ("join" = "no" )
(* request e-mail notification on status change *)
     ("notify" = "bqfe john.doe@gmail.com jane.doe@mail.org" )
(* specific queue to submit the job *)
     ("queue" = "atlas" )
(* CPU time required for the job, converted into seconds *)
     ("cputime" = "3600" )
(* maximal time for the session directory to exist on the remote node, seconds *)
     ("lifetime" = "604800" )
(* memory required for the job, per count, Mbytes *)
     ("memory" = "200" )
(* wall time to start job processing *)
     ("startTime" = "20020128171500" )
(* disk space required for the job, Mbytes *)
     ("disk" = "500" )
(* required architecture of the execution node *)
     ("architecture" = "i686" )
(* required run-time environment *)
     ("runtimeenvironment" = "APPS/HEP/Atlas-1.1" )
(* number of re-runs, in case of a system failure *)
     ("rerun" = "2" )
(* original client xRSL with expanded string substitutions; shortened here *)
     ("clientxrsl" = "&(""dryrun"" = ""no"" )(... )(""rerun"" = ""2"" )" )
```

7.6 ARC Client Config Reference

Some default valued used by *client tools* can be refedined in the client configuration file located in `~/.arc/client.conf`.

All currently supported client configuration options should be defined in the `[common]` block.

> **Warning:** Other blocks (that was previously used to define target options in ARC5) are deprecated from ARC 6.5.0 release!
>
> You can still use such blocks with legacy submission endpoint selection options for backward compatibility. But they are completely ignored with *new ARC6 options set*.

7.6.1 certificatepath

Synopsis: `certificatepath = path`

Description: Specify the location of client certificate file. Environmental variable X509_USER_CERT redefines this value.

Default: `$HOME/.globus/usercert.pem`

Example:

```
certificatepath=/home/user/credentials/cert.pem
```

7.6.2 keypath

Synopsis: `keypath = path`

Description: Specify the location of client secret key file. Environmental variable X509_USER_KEY redefines this value.

Default: `$HOME/.globus/userkey.pem`

Example:

```
keypath=/home/user/credentials/key.pem
```

7.6.3 cacertificatesdirectory

Synopsis: `cacertificatesdirectory = path`

Description: Specify the location of CA certificates directory Environmental variable X509_CERT_DIR redefines this value.

Default: `/etc/grid-security/certificates`

Example:

```
cacertificatesdirectory=/home/user/.globus/certificates
```

7.6.4 proxypath

Synopsis: `proxypath = path`

Description: Specify the location of proxy certificate (both for generation and usage) Environmental variable X509_USER_PROXY redefines this value.

Default: `/tmp/x509up_u${UID}`

Example:

```
proxypath=/home/user/credentials/proxy.pem
```

7.6.5 vomsespath

Synopsis: `vomsespath = path`

Description: Path to file or directory that holds client VOMS configuration to generate proxy certificates

Environmental variables `X509_VOMS_FILE` and `X509_VOMSES` redefine this value If missing arcproxy will search for vomses in the following locations:

- `~/.arc/vomses`
- `~/.voms/vomses`
- `/etc/vomses`
- `/etc/grid-security/vomses`

Default: `undefined`

Example:

```
vomsespath=/home/user/credentials/vomses
```

7.6.6 defaultvoms

Synopsis: `defaultvoms = vo[:command]`

Description: Default value for `--voms` (-S) arcproxy option that is used to define VO and optionaly FQANs used during proxy certificate generation

This option in **multivalued**.

Default: `undefined`

Example:

```
defaultvoms=atlas:/atlas/Role=pilot
defaultvoms=nordugrid.org:all
defaultvoms=ops.ndgf.org
```

7.6.7 rejectdiscovery

Synopsis: `rejectdiscovery = service`

Description: Specify the FQDN or URLs of the services that should be rejected during service discovery process by CLI tools (arcsub, arctest, arcresub)

This option in **multivalued**.

Default: `undefined`

Example:

```
rejectdiscovery=bad.service.org
rejectdiscovery=bad2.service.org
```

7.6.8 rejectmanagement

Synopsis: `rejectmanagement = service`

Description: Specify the FQDN or URLs of the CEs that should be skipped during the job management (e.g. `arcstat`, `arckill`)

This option in **multivalued**.

Default: `undefined`

Example:

```
rejectmanagement=bad3.service.org
rejectmanagement=bad4.service.org
```

7.6.9 brokername

Synopsis: `brokername = broker`

Description: Specify the broker used in resource discovery. The full list of installed brokers can be obtained running `arcsub -P`

Default: `Random`

Example:

```
brokername=FastestQueue
```

7.6.10 brokerarguments

Synopsis: `brokerarguments = args`

Description: Specify broker arguments (if applicable to specified broker)

Default: `undefined`

Example:

```
brokername=PythonBroker
brokerarguments=ACIXBroker.ACIXBroker:https://cacheindex.ndgf.org:6443/data/index
```

7.6.11 timeout

Synopsis: `timeout = seconds`

Description: Amount of time to wait for a service to respond before considering it dead.

Default: `20`

Example:

```
timeout=60
```

7.6.12 joblist

Synopsis: `joblist = path`

Description: Path to the jobs database that holds all extra data about submitted jobs to be used during further job management

Default: `$HOME/.arc/jobs.dat`

Example:

```
joblist=/home/user/arcjobs.dat
```

7.6.13 joblisttype

Synopsis: `joblisttype = type`

Description: Type of the backend used for jobs database.

> **Warning:** IT IS STRONGLY advised to keep default SQLITE type of backend.

Allowed values: `SQLITE`, , `BDB`, , `XML`

Default: `SQLITE`

Example:

```
joblisttype=XML
```

7.7 ARC SDK Documentation

CHAPTER

EIGHT

ARC MISCELLANEOUS PAGES

These pages contain information that does not fit in the formal ARC documentation. That can be for instance more dynamic contents, like overview of testing campaigns related to releases, or how-to pages.

8.1 About the Nordugrid ARC Releases

This page collects general information concerning the releases of the Nordugrid ARC software.

An ARC release is defined as the ARC source package AND the corresponding binary packages on the *supported platforms*.

8.1.1 Release categories

The source package is identified by its `major` and `minor` version number. Due to historical reasons a "bugfix" number is also included, but for scheduled releases (see below) this is always 0. For scheduled releases we do as of March 2018 not distinguish between bugfix and minor releases. The third digit is bumped only in the case of an express release (see below).

An example of a release number is NorduGrid ARC 6.1.0 where 6 refers to the major release series, and 1 to the minor. This is a scheduled release since the last digit is 0.

An ARC release 6.1.1 or 6.1.2 signifies an express release since the last digit is different than 0.

The properties of a major are:

- may break backward compatibility
- introduces new components, features
- obsoletes components
- has longer term planning
- 3-6 months preparation
- can include alpha, beta, rcx test releases
- bumps the major number in the version number
- the contents mainly follow a high level ARC development roadmap

The properties of a minor release are:

- includes bugfixes
- can include new features and/or enhancements
- can include new components
- does not include anything that is backward incompatible
- the release bumps the minor number in the version number

The Nordugrid ARC releases can come in three types: scheduled, express and binary update of a release.

Scheduled releases:

- Minor releases are scheduled regularly monthly or bi-montly
- Major releases are scheduled irregularly as needed, with frequency ranging from 1 to three years.
- All developments that at the time of the release are already merged into master are included in the scheduled release
  - this means that backward incompatible changes are kept in separate development branches, and only merged before a major release is imminent.

Express releases:

- is an unplanned urgent release to fix a security issue or a critical bug
- maximum two weeks preparation
- only development needed to fix the issue is included into the release in order to not delay the release or introduce new bugs
- a separate branch is created from the point on master of the last release, and the release tag is created on this branch
- bumps the third digit in the ARC release number by one: 6.1.0 -> 6.1.1.

Binary update of a release:

- is sometimes needed due to a change in the external dependencies and ARC releases have to be rebuilt
- this "semi release" does not affect the ARC source code, no re-tagging takes place, only new binaries are built
- can happen for both major, minor or express releases

8.1.2 Testing

Testing needs to be done on several Linux distributions as ARC depends on a lot of software that has different versions on different distributions with different file structure.

The level of importance of the different Linux distributions is evaluated as follows:

1. RHEL/CentOS/Scientific Linux version 7.x architecture x86_64 is The Most Important version.
2. RHEL 6, Latest Ubuntu LTS, Latest Ubuntu, any architecture
3. Fedora and other Ubuntu
4. The rest.

Some OSes are more used server-side (RHEL) and some client-side (Fedora).

The testing of Nordugrid ARC is performed in levels, where level 1 and 2 are always performed, and level 3 is performed on the best-effort or as-needed -basis.

Level 1

All supported platforms are built on a nightly basis and publically available.

In addition ARC is built and deployed for the most important platforms on the GitLab CI platform. The deployment includes a basic functional test. Both supported platforms and tests are subject to ongoing development. Currently the build and deploy is performed on each commit to master.

Level 2

The release candidate is deployed on voulenteer sites (minially 1) and on testing infrastructure available at the time. In particular the functionality affected by the development in the current release should be tested.

The test-site tracks his contribution to the test on in the GitLab Wiki.

Level 3

If necessary and possible, larger testing campaigns are organized using the release candidate in question (involving users and sysadmins outside the testing team). This campaign will follow a plan organized centrally by the Nordugrid ARC team. The sites report back on any issues seem, and may be asked to perform tests, and give a simple report back on the test results.

The tests and their reults are recorded on a dedicated pages in the *Testing area of the ARC documentation*

8.1.3 Release notes

Release notes are published on http://www.nordugrid.org/arc/releases/ and are distributed in the following channels:

- News section at http://www.nordugrid.org
- RSS feed: http://www.nordugrid.org/news.xml
- RSS feed: http://www.nordugrid.org/arc/release-news.xml
- LinkedIn NorduGrid page: http://www.linkedin.com/company/nordugrid

The following Web pages are updated upon a release:

- ARC Web page: http://www.nordugrid.org/arc/
- ARC releases: http://www.nordugrid.org/arc/releases/ (updated automatically when release notes are uploaded)
- NorduGrid repositories, in case new one is created: *repository document within this documentation* and http://download.nordugrid.org/repos-6.html
- Wikipedia article: http://en.wikipedia.org/wiki/Advanced_Resource_Connector

8.2 Security Operations

This page collects general information about security operations.

8.2.1 Finding jobs submitted by DN

In case it is necessary to find all the jobs belonging to a specific user, the following command may be run by the local CE admin:

```
[root ~]# arcctl job list --owner <DN>
```

8.2.2 Killing jobs submitted by DN

In case you need to kill all jobs owned by a specific user, identified by a certain certificate DN:

```
[root ~]# arcctl job killall --owner <DN>
```

8.2.3 Removing jobs submitted by DN

Warning: Do not do this before any needed forensics have been performed

In case you want to remove all files related to a jobs run by a specific user:

```
[root ~]# arcctl job cleanall --owner <DN>
```

8.3 ARC 6 Testing Area

Note: This is the home for all ARC6 related testing activities, infrastructures and plans. Please add text, links, anything here

8.3.1 Sites with ARC 6 alpha or release candidate installed

Early installers of ARC 6 - Testing sites

Table 8.1: Testers

Site	OS	First installed	Current version	Mode	Issues reported
Oslo Abel (ce01, ce02, ce03) (T1)	Centos 6	June 2018	Nightlies 201905290207	Production	BUGZ-3785, BUGZ-3737, BUGZ-3736, BUGZ-3781, BUGZ-3782, BUGZ-3780, BUGZ-3779, BUGZ-3735, BUGZ-3785, BUGZ-3802, BUGZ-3796.
David				Testing	BUGZ-3786, BUGZ-3787, BUGZ-3784, BUGZ-3797, BUGZ-3800, BUGZ-3799, BUGZ-3798, BUGZ-3789, BUGZ-3777, BUGZ-3771, BUGZ-3770, BUGZ-3755, BUGZ-3725, BUGZ-3714, BUGZ-3723, BUGZ-3724.
Linkoping Bluegrass		October 2018	ARC RC 6	Production	
Copenhagen T3	Centos 7	End-february	Nightlies: 201905070207	Production	Issue #57
Slovenia - SIGNET	Gentoo	March 2019	ARC RC 6	Production	
Slovenia - SIGNET_NSC	Fedora 28	March 2019	Nightly installation equiv to RC6	Test	
Ukraine - KNU	Centos 7	June 2018	ARC RC 5 (6?)	Production	

List of bugs reported for ARC 6

Bugzilla search

Issues in GitLab

8.3.2 Release testing

- ARC-6.2.0
- ARC-6.3.0

8.4 Release management

8.4.1 People involved in preparing for a release

Table 8.2: People

Name	Experties/responsibility
Anders Waananen	Packaging, autotools, admin access to Bugzilla and nordugrid.org (package downloads http://download.nordugrid.org/repos/6/)
Balazs Konya	Overall coordination
Mattias Ellers	packaging and builds, EPEL and Debian uploads (Globus)
Maiken Pedersen	Release manager
Oxana Smirnova	Bugzilla, Web site, documentation

8.4.2 Release workflow

- Decide on release date and timeline on Nordugrid technical coordination weekly meeting
  - The aim is a monthly scheduled minor release
  - The timeline contains code-freeze date. From this date on no new merge requests will as a rule be added to master.
  - Together with the developers a set of minimal tests are defined for the current release in planning.
  - Clarify if any documenation changes are needed.
  - The release manager enters the test-descriptions on the GitLab wiki test-page: https://source.coderefinery.org/nordugrid/arc/wikis/home
- Release manager sends out a heads-up email announcing the release in preparation and its timeline right after the decision of the release is made
  - to: nordugrid-disucss
  - Inform that documentation updates should be ready by code-freeze date
- The day after code-freeze, the agreed test-sites should install the nightly build from the nightly build repository: http://download.nordugrid.org/builds/index.php?pkgname=nordugrid-arc&type=master and report on the tests and any issues encountered
  - If there are any issues the release date might be postponed.
    - If a postponement is necessary a new timeline is drawn
    - TO-DO: Should a new email go out to nordugrid-discuss or will that just be noise?
  - If no issues are found the release is ready for the next step
- After the definite code-freeze the translations are run
  - The release manager gives a green light to run translations
  - Translations are run. Responsible: Oxana Smirnova

- The release manager creates a tag on master
    - Informs the build-manager about the commit-hash either on skype or by email to nordugrid-core mail list
- The builds are created and repositories populated. Responsible: Anders Waananen
- The list of supported platforms is checked for updates. Responsible: Anders Waanaen
- Once the source tarballs are available in the nordugrid repo, the build in the Fedora build system and for Debian is started. Responsible: Mattias Ellert.
    - The builds should not be made available before the release is announced. TO-DO: Is this possible?
- Prepare the release announcement (see more below). Responsible: Maiken Pedersen
- Notify release manager that the builds are ready to be pushed to the repo. Responsible: Anders Waananen
- Notify responsible in order to publish the release annoucement on the web. Responsible: Maiken Pedersen
    - Publish the release announcement. Responsible: Oxana Smirnova
- Notify responsible in order to push the packages to the nordugrid repo. Responsible: Maiken Pedersen
    - Push packages to nordugrid repo. Responsible: Anders Waananen
- Notify responsible in order to push builds to Fedora and Debian repo. Responsible: Maiken Pedersen
    - Push packages to Fedora and Debian repos. Responsible: Mattias Ellers
- Ensure there are consistent binary packages in agreed repositories and linked from the release page. Responsible: Maiken Pedersen
- Ensure there is corresponding release version in Bugzilla. Responsible: Anders Wannanen

8.4.3 The release announcement should contain

- Most important changes/highlights
- List of fixed bugs
- List of new features (if relevant)
- List of backward incompatible changes (if relevant - only for major releases)
- Link to documentation in addition to installation links
- Link to the GitLab ARC repo
- Getting-in-touch information

8.5 Changelogs/list of bugs

8.5.1 Main changes in ARC 6 compared to ARC 5

There are many improvements and changes in the new ARC release, and with these we hope to have greatly enhanced the ease of setting up, configuring and managing an ARC production site, and to have improved the reliability and scalability of the ARC CE by the internal restructuring ARC has undergone.

> **Note:** Despite all the new features and code changes, the supported ARC 6 CE interfaces are unchanged - the latest ARC 5 clients are compatible with an ARC 6 CE and vice versa.

- **Complete overhaul of server-side *ARC CE configuration*:**
    - Configuration has been completely reworked, cleaned up, regrouped and made more streamlined.

- In particular, the way authorization and user mapping is handled has been greatly improved. Worth mentioning is the support of higher granularity vo-based authorization, and that mapping via gridmap files is no longer required nor recommended.
- ARC now comes with a so-called zero configuration - a preconfigured minimal ARC setup automatically installed with ARC, including test-CA and test-host certificate for immediate test submission.
- Default values throughout ARC have had a big tidy up and are now handled consistently through the ARC runtime configuration.
- Configuration blocks are now used to enable (turn on) or disable a functionality, a service or an interface. For example, the NorduGrid schema is no longer published unless it is enabled via the corresponding configuration block.
- Validation of the ARC configuration is enforced by A-REX startup scripts. If you have configuration errors then A-REX will not start, and you will be pointed to the error.

- **Scalability and manageability improvements:**
  - The internal job-loop in A-Rex has been re-engineered to be event-driven
  - xrootd plugin has been re-written to improve data transfer performance
  - Consistent handling of server-side logging and improved logfile structure
  - Reworked startup scripts
  - Streamlined package name scheme, where e.g. ldap services have been separated out
- *The RTE framework* has got a redesigned architecture and a largely extended functionality (default, installed and enabled RTEs, introduction of RTE parameters)
- **A new framework for *RunTimeEnvironments (RTE)*:**
  - The RTE framework has gotten a redesigned architecture and largely extended functionality (default, installed, enabled RTEs, introduction of RTE parameters).

    Note: RTEs must in ARC 6 be *explicitly enabled through the new arcctl tool* as a separate step after installation.

  - **There are several system RTEs installed together with ARC which you can enable on demand. These are:**
    * ENV/CANDYPOND: makes ARC Candypond ("Cache and deliver your pilot on-demand data") client available on the Worker Nodes and ready to be used
    * ENV/CONDOR/DOCKER: enables submission to Docker universe in HTCondor backend
    * ENV/LRMS-SCRATCH: enables the usage of local WN scratch directory defined by LRMS
    * ENV/PROXY: copies proxy certificate to the job session directory
    * ENV/RTE: copies RunTimeEnvironment scripts to the job session directory
- ***ARCCTL*, a new server-side management and control tool for sysadmins has been developed.**
  - arcctl is meant to be the central one-stop-shop tool for administrating an ARC CE
  - With this tool you can handle RTEs, the jobs, accounting, ARC services and many other things related to an ARC CE.
  - arcctl also offers help with integrating 3rd party services and components with an ARC CE such as handling CAs, VOMS, etc..
- ***JURA Accounting* subsystem improvements:**

  - Accounting configuration, archive and operations has been restructured to improve typical operations.
  - Improved republishing via arcctl.
  - Better integration with APEL SSM.
  - Archive structure is relying on a database for republishing and gathering stats locally.
- ***ARCHERY*, the new DNS-based service endpoint catalogue for ARC**
  - archery-manage package to populate the ARCHERY registry with ARC CE endpoint info
  - Official top-level DNS registry under the nordugrid.org domain
  - ARCHERY-compatible clients to submit and monitor jobs (arc cli, ldap monitor)
  - ARCHERY replaces the OBSOLETED EGIIS service of the ARC 5 release series.
- **TECHNOLOGY PREVIEW components:**
  - The REST interface - enabled together with the A-Rex Web-Service.
  - The INTERNAL interface - a special interface aimed for restrictive HPC sites, to be used with a local installation of the ARC Control Tower.
  - *Candypond* - "Cache and deliver your pilot on-demand data" service to bring the power of ARC Cache to the pilot job universe.
  - Experimental Python LRMS for SLURM with SSH access support.
- **The ARC client and SDK:**
  - The client has undergone a major internal cleanup, large number of submission, target retrieval and job description plugins got removed as a result of OBSOLETING third-party interfaces and technologies in ARC.
  - New plugins for ARCHERY and the REST And INTERNAL interfaces were implemented.
  - The arcstat cli of the ARC6 client and the underlying SDK now handles walltime as (walltime)*(number of cores) for multicore jobs.

REMOVED and obsoleted components

- Large number of configuration options of arc.conf got DELETED during the server-side configuration rework. A detailed list of those options are available in the arc.conf.DELETED file.
- The separate gangliarc component is discontinued, instead we now offer ganglia integrated into AREX.
- Nordugrid no longer distributes a source code bundle including documentation, gangliarc and nagios.
- CREAM related plugins and server-side components.
- UNICORE related plugins and server-side components.
- All the windows & solaris related ARC code and build.
- JAVA bindings for ARC SDK.
- Support for BES, WSRF and other non-EMIES WS flavours as job management interfaces.
- Support for JDL, ARCJSDL, RSL job description dialects: ARC keeps only XRSL and ADL as supported job description languages.
- EMIR indexing service including ARC CE registration to EMIR and ARC SDK plugins.
- Server-side EGIIS, including ARC CE registration to EGIIS. BUT: keep EGIIS plugins in the client.
- GLUE1 support from the ARC client SDK, server-side support for GLUE1 temporarily kept but labelled as OBSOLETE.
- ARC SDK support to obtain information from top-BDII.

- Publishing nordugrid-authuser objects in the NorduGrid LDAP schema

List of bugs fixed since ARC 5

A rather long (but not exhaustive) list of bugs fixed since ARC 5 can be found here: http://www.nordugrid.org/arc/arc6/common/changelog/bugs-6.0.0.html

8.5.2 List of bugs fixed since ARC 5.4.4

Table 8.3: bugs_600

Bug ID	Summary
BUGZ-895	ARC Administrative tools
BUGZ-1421	No log of backend script problems
BUGZ-3034	arcstat -s is case-sensitive
BUGZ-3106	arcproxy could be more expressive when it finds a problem
BUGZ-3360	SGE and LL backends fail to report correct node OS/system information in GLUE2 infosystem
BUGZ-3384	Support for per-queue authorisation configuration and publishing
BUGZ-3451	Configure number of cpus manually
BUGZ-3476	Crashes in multiple ARC components
BUGZ-3545	Patch for Correct Cores Parsing
BUGZ-3557	undetected job submission in case of heavy filesystem load
BUGZ-3565	Allow setting default VO in ~/.arc/client.conf to be used in arcproxy
BUGZ-3566	Implement RTEs processing without shared directory
BUGZ-3569	Exporting CPU/Wall time limits to Glue2/BDII
BUGZ-3570	Arcsub used 100GB memory
BUGZ-3584	JURA: create one log file per job, not per submission
BUGZ-3624	Data delivery service can only listen to one network interface.
BUGZ-3626	Force GLUE2ComputingManagerTotalLogicalCPUs to be totalcpus when this value is defined in arc.conf
BUGZ-3632	arcproxy fails in Ubunt 16.04, 16.10 and recent Debian systems
BUGZ-3637	arcget with multiple jobs crashes
BUGZ-3643	The watchdog crashes
BUGZ-3662	arcsub crashes
BUGZ-3667	JSON output for arcstat
BUGZ-3674	settings in client.conf ignored
BUGZ-3675	Problems retrieving jobs with arcget 1
BUGZ-3676	Problems retrieving jobs with arcget 2 (first byte missing)
BUGZ-3677	arcproxy fails retrieving attributes from voms
BUGZ-3682	Better error message when DN not in gridmap file
BUGZ-3690	Warnings about missing information on deleted jobs
BUGZ-3695	Slowness with arccp and the xrootd protocol
BUGZ-3700	ARC1ClusterInfo.pm uses netstat
BUGZ-3702	Cannot use arc data commands without certificates
BUGZ-3707	Seg fault Triolith - related or not to the data-staging problems on Triolith
BUGZ-3713	Malformed jobs.dat entries
BUGZ-3722	Xenial repo for the (nordugrid) ARC source is not correct
BUGZ-3756	package update failed to restart A-REX
BUGZ-3772	Can't use different credentials within one submission process when jobs require user input files
BUGZ-3773	Enabling arex-ganglia breaks controldir access
BUGZ-3778	arcctl not working if arc code configured with –disable-swig flag and installed with make install
BUGZ-3788	Poor performance with arccp and HTTPS

continues on next page

Table 8.3 – continued from previous page

Bug ID	Summary
BUGZ-3812	A-REX hangs inside XRootd after fork

8.6 Using ARC packages from nightly builds

Recent development version of ARC is available as a nightly builds packages for many Linux distribution.

To use these nightlies smoothly for installation and updates, the repository should be added to your system.

8.6.1 Enabling nightlies repo for RHEL-based distributions

Setup a CRON job (e.g. `/etc/cron.daily/update-arc-nightly-time.sh`) to fetch the latest nightlies date:

```
#!/bin/bash
curl -s http://builds.nordugrid.org/nightlies/nordugrid-arc/master/ | sed -n 's/^.*<a.
↪*>\(.*\)\/<\/a>.*$/\1/p' | sort  | tail -1 > /etc/yum/vars/arcnightly
```

Run the CRON script once manually to have the `arcnightly` variable initialized. Also, make sure the script has executable permissions.

To add the nightlies repository to your RHEL-based system, create a `/etc/yum.repos.d/nordugrid-nightly.repo` with the following content[1]:

```
[nordugrid-nightly]
name=Nordugrid ARC Master Nightly Builds - $basearch
baseurl=http://builds.nordugrid.org/nightlies/nordugrid-arc/master/$arcnightly/centos/
↪el7/$basearch
enabled=1
gpgcheck=0
```

Check if it works running `yum` (or `dnf`), e.g.:

```
[root ~]# yum makecache
```

8.6.2 Install required packages

The NorduGrid repositories for RedHat Enterprise Linux/CentOS depends on the EPEL Repositories which must also be part of the YUM configuration:

For RHEL7 flavour:

```
yum install -y epel-release
```

[1] NOTE that you should modify OS release version to match your case. For Fedora releases, replace `centos/el` with `fedora/`.

For RHEL8 flavour:

```
dnf config-manager --set-enabled powertools
```

For RHEL9 flavour:

```
dnf config-manager --set-enabled crb
```

8.6.3 Enabling nightlies repo for Debian-based distributions

Nightlies for Debian/Ubuntu are available as standalone packages without repository index files generated.

The suggested approach is to:

- download packages locally
- create the necessary repository package index
- use the repository on the local filesystem

To accomplish this, install the necessary tools for making repository files:

```
[root ~]# apt-get -y install dpkg-dev
```

Regularly fetch latest nightly packages and create the repository index with the following daily CRON job:

```
#!/bin/bash

arcrelease='ubuntu/16.04/amd64'
latestdate=$( wget -q -O - http://builds.nordugrid.org/nightlies/nordugrid-arc/master/
↪ | sed -n 's/^.*<a.*>\(.*\)\/<\/a>.*$/\1/p' | sort | tail -1 )
repodir=/srv/nordugrid-nightlies

rm -rf $repodir; mkdir -p $repodir; cd $repodir

wget -q -r -nH --cut-dirs=8 --no-parent -l1 -A "*.deb, *.ddeb" http://builds.
↪nordugrid.org/nightlies/nordugrid-arc/master/$latestdate/$arcrelease/

dpkg-scanpackages . /dev/null 2>/dev/null | gzip -9c > Packages.gz
```

Create the local repository file `/etc/apt/sources.list.d/nordugrid-nightlies.list` and add the contents:

```
deb [trusted=yes] file:/srv/nordugrid-nightlies ./
```

Check it works running `apt`, e.g:

```
[root ~]# apt-get update
```

8.7 NorduGrid repository information for ARC 6

The NorduGrid ARC packages are available through YUM and APT repositories for several systems. We have release-based repositories that you can follow. This will keep your install to a particular release of NorduGrid ARC with only minor and bug-fixing updates. You can also choose to follow the repository "latest" which will always point to the latest stable release. For each distribution there are 3 channels (repositories) available:

- *base* - Base packages (mandatory)
- *updates* - Updates to the base release (strongly recommended)
- *testing* - Packages almost ready to go into the updates repository (optional). The alpha, beta and release candidate releases can be found here.

8.7.1 Repository security

The NorduGrid RPM packages and DEB repositories are signed, and in order for the repository tools APT and YUM to verify them you must install the NorduGrid GPG key:

For `rpm` based distributions like Red Hat Enterprise Linux and Fedora:

```
[root ~]# rpm --import http://download.nordugrid.org/RPM-GPG-KEY-nordugrid-6
```

For Ubuntu distributions with `sudo`:

```
[user ~]$ wget -q http://download.nordugrid.org/DEB-GPG-KEY-nordugrid-6.asc -O- |␣
↪sudo apt-key add -
```

For Debian without `sudo`:

```
[root ~]# wget -q http://download.nordugrid.org/DEB-GPG-KEY-nordugrid-6.asc -O- | apt-
↪key add -
```

8.7.2 Repository configuration - Red Hat Enterprise Linux

The NorduGrid ARC repositories for RedHat Enterprise Linux / CentOS packaging utility `yum` or `dnf` can be configured through:

```
/etc/yum/nordugrid.repo
```

The repository configuration can be set up automatically by means of installing `nordugrid-release` package or creating the configuration file manually.

Install nordugrid-release package with YUM/DNF

The easiest way to configure YUM/DNF to use the NorduGrid repository for Red Hat Enterprise Linux, CentOS and similar distributions is to install the `nordugrid-release` package which can be found in the NorduGrid package repository for the appropriate RHEL/EPEL release.

Example packages are shown below for `x86_64` architectures, they also exist for `i386` when applicable. In that case exchange the `x86_64` in the links below with `i386`.

Rocky Linux: 9 8

CentOS Linux: EL7 EL6

CentOS Stream: 9 8

Fedora: 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12

Install with yum (CentOS Linux 6+7) or dnf (Fedora, CentOS Stream, Rocky Linux, CentOS Linux 8), here shown for CentOS Linux:

```
[root ~]# yum install <rhel-repo link>
```

This creates the appropriate repo files in /etc/yum.repos.d/.

Manual YUM repository setup - NorduGrid repository

For manual YUM repository setup, create a file /etc/yum.repos.d/nordugrid.repo with the following contents (here using CentOS as example, if you are on Fedora, replace centos with fedora)

If you are installing an alpha, beta or release candiate, please set the nordugrid-testing to enabled=1.

```
[nordugrid]
name=NorduGrid - $basearch - base
baseurl=http://download.nordugrid.org/repos/6/centos/$releasever/$basearch/base
enabled=1
gpgcheck=1
gpgkey=http://download.nordugrid.org/RPM-GPG-KEY-nordugrid-6

[nordugrid-updates]
name=NorduGrid - $basearch - updates
baseurl=http://download.nordugrid.org/repos/6/centos/$releasever/$basearch/updates
enabled=1
gpgcheck=1
gpgkey=http://download.nordugrid.org/RPM-GPG-KEY-nordugrid-6

[nordugrid-testing]
name=NorduGrid - $basearch - testing
baseurl=http://download.nordugrid.org/repos/6/centos/$releasever/$basearch/testing
enabled=0
gpgcheck=1
gpgkey=http://download.nordugrid.org/RPM-GPG-KEY-nordugrid-6
```

Check if it works running yum (or dnf), e.g.:

```
[root ~]# yum makecache
```

Install required packages

The NorduGrid repositories for RedHat Enterprise Linux/CentOS depends on the EPEL Repositories which must also be part of the YUM configuration:

For RHEL7 flavour:

```
yum install -y epel-release
```

For RHEL8 flavour:

```
dnf config-manager --set-enabled powertools
```

For RHEL9 flavour:

```
dnf config-manager --set-enabled crb
```

Once the NorduGrid repositories are configured, install the packages with:

```
[root~]#  yum install <list of package names>
```

If you are installing an alpha, beta or release candiate, you must install by enabling the nordugrid-testing repo.

```
[root~]#  yum install --enablerepo nordugrid-testing <list-of-packages>
```

Are you on recent version of Fedora, use `dnf` instead of `yum`.

Please refer to the *ARC Computing Element Installation and Configuration Guide* for package selection and configuration.

8.7.3 Repository configuration - Debian and Ubuntu

The NorduGrid ARC repositories for Debian and Ubuntu packaging utility APT can be configured through:

```
/etc/apt/sources.list
```

or when supported through a repo specific file:

```
/etc/apt/sources.list.d/nordugrid.list
```

The configurations for the varios APT based distributions can be found in the following sections. To enable a specific repository, remove the “#” from the beginning of the line, before the “deb” as shown for the Base Channel.

The repository configuration can be set up automatically by means of installing `nordugrid-release` package or creating the configuration file manually.

Install nordugrid-release package for Debian/Ubuntu through dpkg

The examples below give you the link for most recent Debian/Ubuntu releases. Packages are shown below for `amd64` architecture. Replace `amd64` for `i386` if required for your architecture.

Debian: 12 11 10 9 8 7

Ubuntu: 24.10 24.04 23.10 23.04 22.10 22.04 21.10 21.04 20.10 20.04 19.10 19.04 18.10 18.04 17.10 17.04 16.10 16.04 15.10 15.04 14.10 14.04

Install the source file with `dpkg`, example shown for Debian 10:

```
[root ~]# wget -q https://download.nordugrid.org/packages/nordugrid-release/releases/
↪6.2/debian/10/amd64/nordugrid-release_6.2~bpo10+1_all.deb
[root ~]# dpkg -i nordugrid-release_6.2~bpo10+1_all.deb
```

For a different version of Debian or Ubuntu, change the version names appropriately.

Manual APT repository setup - NorduGrid repository

For manual APT repository setup for Debian, the APT sources file should contain the following (here shown for Debian 10 buster):

```
# Base channel - must be enabled
deb http://download.nordugrid.org/repos/6/debian/ buster main
deb-src http://download.nordugrid.org/repos/6/debian/ buster main

# Updates to the base release - should be enabled
deb http://download.nordugrid.org/repos/6/debian/ buster-updates main
deb-src http://download.nordugrid.org/repos/6/debian/ buster-updates main

# Scheduled package updates - optional
#deb http://download.nordugrid.org/repos/6/debian/ buster-experimental main
#deb-src http://download.nordugrid.org/repos/6/debian/ buster-experimental main
```

For manual APT repository setup for Ubuntu, the APT sources file should contain the following (here shown for Ubuntu 20.04 focal):

```
# Base channel - must be enabled
deb http://download.nordugrid.org/repos/6/ubuntu/ focal main
deb-src http://download.nordugrid.org/repos/6/ubuntu/ focal main

# Updates to the base release - should be enabled
deb http://download.nordugrid.org/repos/6/ubuntu/ focal-updates main
deb-src http://download.nordugrid.org/repos/6/ubuntu/ focal-updates main

# Scheduled package updates - optional
#deb http://download.nordugrid.org/repos/6/ubuntu/ focal-experimental main
#deb-src http://download.nordugrid.org/repos/6/ubuntu/ focal-experimental main
```

For a different release version, change the version name accordingly.

For Debian:

- 12: bookworm
- 11: bullseye
- 10: buster
- 9: stretch
- 8: jessie
- 7: wheezy

For Ubuntu:

- 24.10: oracular
- 24.04: noble
- 23.10: mantic
- 23.04: lunar
- 22.10: kinetic
- 22.04: jammy
- 21.10: impish
- 21.04: hirsute
- 20.10: groovy

- 20.04: focal
- 19.10: eoan
- 19.04: disco
- 18.10: cosmic
- 18.04: bionic
- 17.10: artful
- 17.04: zesty
- 16.10: yakkety
- 16.04: xenial
- 15.10: wily
- 15.04: vivid
- 14.10: utopic
- 14.04: trusty
- 13.10: saucy
- 13.04: raring
- 12.10: quantal
- 12.04: precise

Install required packages

Run the following command to update the packages database:

```
[root~]# apt-get update
```

Install the packages with (showing example for nordugrid-arc-arex):

```
[root~]# apt-get <list-of-packages>
```

Please refer to the *ARC Computing Element Installation and Configuration Guide* for package selection and configuration.

8.8 Work-in-progress Docs

Hidden area that holds unfinished documents to be build and available in the doc tree, but not yet linked to the right place.

8.8.1 INTERNAL interface - ARC 6

NB! WIP [TODO] Fix references, now in tex style ref to arc-ce sys admin guide

The INTERNAL submission interface is aimed for restrictive HPC sites. When ARC runs in the internal mode, it should do so alongside a local instance of the ARC Control Tower (aCT). aCT pulls jobs from the central job server (like PaNDA for ATLAS), and and feeds ARC with new jobs internally. Since there is no external access, there is no need for a web-service, gridftp server or ldap, as the purpose of these components are to facilitate external access. There is neither a need for a host-certificate on such a machine. The only service running on ARC is A-REX. Therefore a stripped-down version of aCT and ARC-CE can be used which is beneficial for installation, configuration and maintenance. Furthermore, in the INTERNAL mode ARC should be installed as a normal user.

No user mapping (See Section myref{sub:access_control}) is needed in this case, as the default behaviour in ARC 6 is to map the user submitting the job to the A-REX user.

Implementation overview

The INTERNAL submission plugin which is part of the ARC client, interacts with the parent plugin classes using the same API as the other plugins such as gridftp plugin or the emi-es plugin. However, the INTERNAL plugin interacts directly with the A-REX memory and methods, and therefore is integrated as part of the A-REX service which belongs to the ARC-CE code-base. Therefore, both the ARC client and the ARC CE must be installed and on the same machine for the INTERNAL submission plugin interface to function.

All interaction between the client and A-REX happens directly via files in the controldir or via A-REX memory.

Actions

Retrieving Service Information

As a site running in the INTERNAL mode is not accessible from the outside, any service retrieval information can only be done from within the site. You may inspect the service information as usual by issuing the arcinfo command. The information is extracted by direct access to the info.xml file in the controldir. The sstat method of the INTERNALClient reads the info.xml and outputs information in xml-format to the client. An example of the output of arcinfo for localhost

```
[root ~]# arcinfo -c localhost
Computing service:
  Information endpoint: file://localhost
  Submission endpoint: file://localhost (status: ok, interface: org.nordugrid.
↪internal)
```

When arcinfo is called, the INTERNAL submission interface extracts the site information by directly accessing the info.xml file in the controldir. The INTERNAL plugin reads the info.xml and outputs information in xml-format to the client, which in turn displays it to the user.

Job submission

When a job is submitted via the INTERNAL submission interface the plugin creates an A-REX job object, which in turn takes care of creating all necessary files (like for instance the ARC job description) and folders (sessiondir) for the job, in addition to creating a job ID. The INTERNAL plugin then places any input files local to the client in the newly created sessiondir. Remaining remote input files are downloaded by the DTR (See Section myref{sub:datastaging}). Once these files are present in the controldir A-REX adds the job to its joblist, and takes over the handling of the job from there.

Accessing Information About Job

Job information evoked by calling arcstat is extracted from a combination of information stored in A-REX memory (job state) and the job.ID.local file in the controldir (session, stagein and stageout directories).

Controlling Execution Of Job

Killing, cleaning and resubmitting jobs is initiated by direct call the existing ARexJob methods: Kill(), Clean(), Resume(). These methods all place files in the controldir that the grid-manager acts upon, such as job.jobid.clean mark or job.jobid.cancel mark.

Delegation Interface [TODO]

BIBLIOGRAPHY

[6.2] New in version 6.2.

[globus] In the ARC < 6.5 there was an all-in-one plugin `globus` instead of `gridftp`, `gridftpjob` and some other components.