23/4/2004

The NorduGrid User Guide

Contents

Chapter 1

Introduction

The NorduGrid toolkit is a light-weight Grid solution, designed to support a dynamic, heterogeneous Grid facility, spanning different computing resources and user communities. It provides *middleware* to interface between user applications and distributed resources.

The NorduGrid middleware is almost entirely based on the Globus Toolkit® API, libraries and services [1]. In order to support the original NorduGrid architecture, several innovative approaches were used, such as the Grid Manager [2] and the User Interface (Section 5). Several Globus components were extended and developed further, such as the Information Model [3] and the Extended Resource Specification Language (Section 4).

The NorduGrid User Guide puts together descriptions of user-end parts of the toolkit, serving as a user manual and a reference. It also includes basic examples and the NorduGrid client installation instructions.

This User Guide does not provide detailed description of third-party tools, such as the Globus Toolkit® , giving only minimal necessary information.

The definitive description of the components and services can be found in separate manuals:

User Interface : "The NorduGrid toolkit user interface", user manual [4]

xRSL : "Extended Resource Specification Language" [5]

Grid Manager : "The NorduGrid Grid Manager", description and administrator's manual [2]

Grid Monitor : "The Grid Monitor", usage manual [6]

Detailed installation and configuration instructions are included in the NorduGrid software distribution, available for download at:

`http://www.nordugrid.org`

Any further questions should be addressed to

`nordugrid-support@nordugrid.org`

Chapter 2

NorduGrid Client Installation

The very first step to get acquainted with the NorduGrid facilities is to download and install the client package. There are several ways to do it, you just have to choose which suits you best. Binary distributions are available for several GNU/Linux flavors, such as RedHat, Slackware or Mandrake. Source distributions are available as well. The client package may be installed system-wide by a system administrator, as well as locally by any user.

2.1 System-wide installation

This Section is oriented primarily towards users with system administrator privileges. Local installation by any user is described in Section 2.2.

For a system-wide installation, usage of RPMs is recommended for those Linux platforms which support it. Otherwise, the client can be built from the sources.

2.1.1 Globus Toolkit

Globus Toolkit® must be installed at your machine prior to any system-wide NorduGrid installation. You may get it from the Globus project Web site [1], however, it is recommended to use the distribution available at the NorduGrid download area, as it contains several important bug fixes.

Case A. If you already have a Globus Toolkit® installed on your system, check that the flowing variables are defined and point to the proper locations:

```
echo $GLOBUS_LOCATION $GPT_LOCATION
```

If the variables are not defined, define them according to your installation.

Case B. If there is no Globus installation, or you want to profit from the Globus Toolkit® distribution with most recent bug fixes and patches, download from `http://ftp.nordugrid.org/download` (see Figure 2.1) and install the necessary packages in the following order:

```
rpm -Uvh gpt-<version>.rpm
export GPT_LOCATION=/opt/gpt
rpm -Uvh globus-<version>.rpm
export GLOBUS_LOCATION=/opt/globus
rpm -Uvh globus-config-<version>.rpm
```

You may well want to install Globus Toolkit® from pre-compiled tarballs provided at the NorduGrid download site. In such a case, take care to execute post-installation procedures:

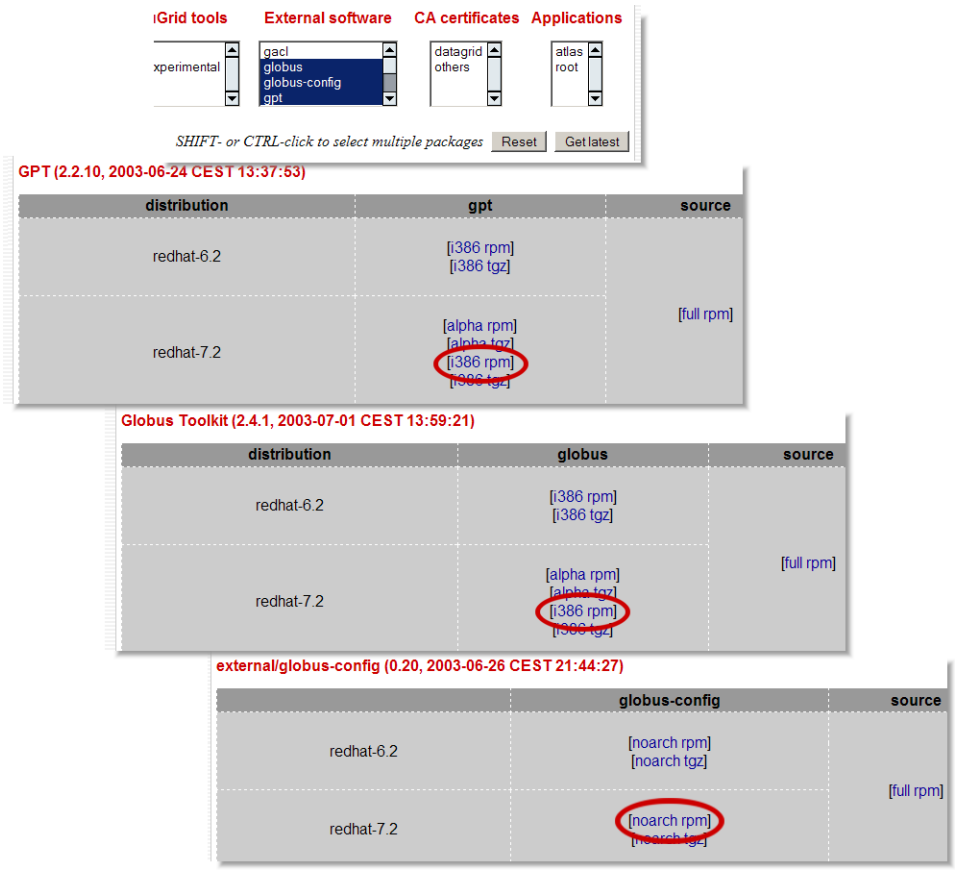


Figure 2.1: Downloading Globus-related packages (GPT, Globus and Globus configuration) from the NorduGrid Web site. This example shows which packages to download for the RedHat 7.2 Linux distribution.

<globus location>`/setup/globus-post-install-script` *<globus location>*

Here *<globus location>* is the directory in which you unpacked Globus (by default `/opt/globus`).

NB: on some systems and distributions, certain additional external packages need to be installed (most notably, Perl components). They are normally available from the NorduGrid downloads area, "External software" section.

2.1.2 Download the client

Download a NorduGrid client package suitable for your operating system from the NorduGrid Downloads area at `http://ftp.nordugrid.org/download` :

- If you are installing on top of an older Globus version, download a **full source** NorduGrid RPM (`nordugrid-`*<version>*`.src.rpm`) or a tarball (`nordugrid-`*<version>*`.tgz`) (see Figure 2.2).
- If you just have installed Globus as described in Section 2.1.1, download just a binary client RPM `nordugrid-client`*<version>*`.rpm` (see Figure 2.2).

2.1.3 Build

If you have installed Globus as described in Section 2.1.1, skip this Section.

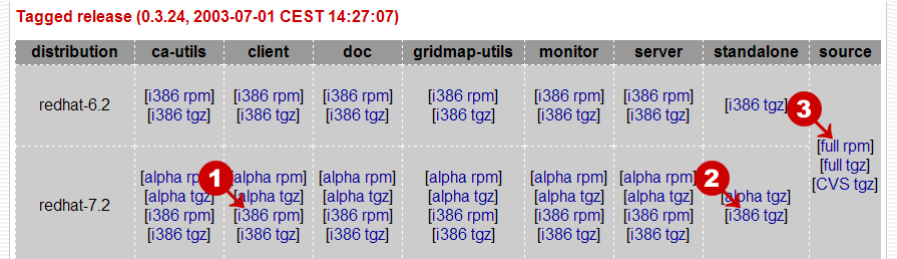


Figure 2.2: Downloading different client packages: (1) binary client RPM, (2) standalone pre-compiled archive, (3) full source RPM.

If you are installing on top of an older Globus version, you have to rebuild the client from source. To do it with a source NorduGrid RPM, do:

```
rpm --rebuild nordugrid-<version>.src.rpm
```

This will create several binary RPMs; you will need only `nordugrid-client`<*version*>`.rpm`.

Alternatively, to build from a tarball, do the following:

```
cat README
tar xvzf nordugrid-<version>.tgz
cd nordugrid-<version>
./configure
make
make install
```

2.1.4 Install the client

Install the NorduGrid client RPM:

```
rpm -Uvh nordugrid-client<version>.rpm
```

Set up the environment variables:

```
source /etc/profile.d/globus.sh
source /etc/profile.d/nordugrid.sh
```

2.1.5 Certificates

Depending on what is your Grid certificate issuing authority (*Certificate Authority, CA*, see Chapter 3) , and which Grid resources you intend to use, you will have to install public certificates and other attributes of the necessary authorities. Table 2.1 presents a short check-list, summarizing which certificates you would need.

The NorduGrid CA **certificate** and the **signing policy** files are distributed from the NorduGrid Web site, "Certificate Authority" section, as well as through the downloads area at `http://ftp.nordugrid.org/download` , see Figure 2.3..

Install all the downloaded CA RPMs:

```
rpm -Uvh ca_NorduGrid*
```

Alternatively, use the provided tarballs or download and install the necessary files one by one from the CA Web page directly. In any case, the obtained CA attributes should be copied into the `$X509_CERT_DIR` directory.

Your certificate	*Will use NorduGrid resources*	*Will use other Grid resources*
NorduGrid certificate	NorduGrid CA certificate and signing policy	Certificates, signing policies and other attributes (if any) of all the CAs that certified the resources
Other Grid certificate	Your issuing CA certificate and signing policy file (if any) and the NorduGrid CA certificate and signing policy	
No Grid certificate	NorduGrid CA certificate and signing policy, and certificate request configuration files	

Table 2.1: Check-list of needed certificates

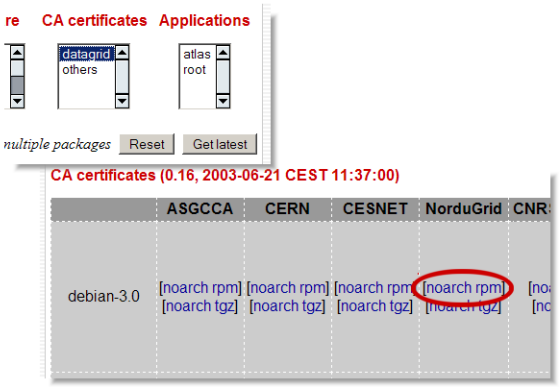


Figure 2.3: Downloading NorduGrid CA attributes: certificates are platform- and system-independent.

By default, the `X509_CERT_DIR` variable is not set up. The standard location for Globus keys is `/etc/grid-security/certificates`. However, it is always possible to point the `X509_CERT_DIR` variable to any directory where you prefer to store the certificates. This could be useful if you assume that different users may trust different CAs, not necessarily those centrally accepted.

The recommended procedure to obtain other CA certificates and attributes is to fetch them directly from the CAs in question. They are typically available from their respective Web sites (for a non-authoritative list, see Appendix B) .

If for some reason it is impossible to fetch your CA attributes (e.g., Web site unavailable, holiday season *etc.*), you may download the necessary files from the NorduGrid downloads area at `http://ftp.nordugrid.org/download` . The attributes are platform- and system-independent. Install the CA attributes either from RPMs:

`rpm -Uvh` *<your CA name>*`*`

or from the provided tarballs.

IMPORTANT! The rule of thumb when installing the NorduGrid middleware is **a**lways to check the contents of the `$X509_CERT_DIR` folder and to make sure that it does contain certificates and signing policy files of your certificate issuer CA. A typical listing of the folder should look as follows:

```
> ls -1 $X509_CERT_DIR
1f0e8352.0
1f0e8352.signing_policy
bc870044.0
```

```
bc870044.signing_policy
d64ccb53.0
```

Here 8-symbol file names with extension ".0" are the certificates containing public keys of different CAs.

2.1.6 Request a Certificate

To request a personal or a host certificate, you must decide to which authority you want to submit such a request. Appendix B lists some known authorities and their contact addresses. Each CA has its own procedure, which you have to follow.

If you have no certificates and are a resident of a Nordic country*, you should request the certificate from the NorduGrid Certificate authority. To do this, you should first download at `http://ftp.nordugrid.org/download` the corresponding `certrequest-config` configuration package, residing in section "CA certificates" (see Figure 2.4). Install the RPM package using

```
rpm -Uvh ca_NorduGrid-certrequest*
```

or use the provided tarballs. The following files will be installed in `/etc/grid-security/certificates`:

```
globus-host-ssl.conf.1f0e8352
globus-user-ssl.conf.1f0e8352
grid-security.conf.1f0e8352
```

You may well prefer installing the files in any other location defined by the `$X509_CERT_DIR` variable.

Despite appearance, this is not a part of a general installation and is specific only for the Nordic countries residents.

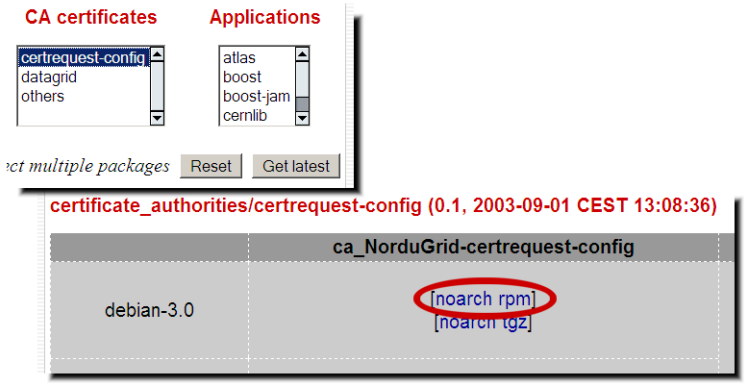


Figure 2.4: Downloading NorduGrid CA certificate request configuration package.

The certificate request then should be prepared by doing the following:

```
grid-cert-request -int -ca
```

From the presented list, select the NorduGrid CA, and then enter the requested information at each interactive prompt. Typically, you should use the suggested default values, except of the case of the Organisational

*The term "Nordic countries" refers to Denmark, Finland, Iceland, Norway and Sweden.

Unit ("OU", your domain, default is `nbi.dk`) and your proper name. Upon request, type and confirm your password. Follow the printed suggestions strictly, i.e. do the following:

```
cat $HOME/.globus/usercert_request.pem | mail ca@nbi.dk
```

Alternatively, simply send the file `$HOME/.globus/usercert_request.pem` to `ca@nbi.dk`.

The certificate request procedure creates the new directory `.globus` in your `$HOME` area, and places there three files[†]:

`usercert_request.pem`	the official certificate request, to be mailed to `ca@nbi.dk`
`userkey.pem`	your **p**rivate key
`usercert.pem`	your public certificate placeholder (initiated with zero size)

When and if the CA fulfils your request, they will send you the actual public certificate, which you must store as `$HOME/.globus/usercert.pem`, overwriting the placeholder.

> If you are not a Nordic country resident, you may well try (upon consulting the NorduGrid personnel at `nordugrid-support@nbi.dk`) to use the same procedure to request a certificate from another Certificate Authority. However, most of them have own established procedures (see Appendix B for a reference), and you are **strongly** advised to follow them.

2.1.7 Launch a Grid Session

Log into the Grid:

```
grid-proxy-init
```

Type your password when requested, and enjoy the Grid world!

2.2 Local Installation

For a local installation by a user without system administrator privileges, it is advised to download and install a stand-alone distribution of the NorduGrid client. This package is distributed in `.tgz` format and contains all the required third-party components (Globus Toolkit®). Provided with the package are scripts to do all the necessary initial configuration and setup.

2.2.1 Download

Download a pre-compiled stand-alone binary distribution suitable for your operating system from the NorduGrid Downloads area at `http://ftp.nordugrid.org/download` , see Figure 2.2.

2.2.2 Unpack

Put the downloaded package in a directory of your choice and execute

```
tar xvzf nordugrid-standalone-<version>.tgz
```

This will create a new directory `nordugrid-standalone-`<*version*>, and the downloaded `.tgz` package can be safely removed

[†]Keys and certificates, as well as their locations, can be called different names. In the instructions above, default naming scheme was used. If for some reasons you would like to use different names, feel free to contact `nordugrid-support@nbi.dk` for further directions.

2.2.3 Configure

Configure and set up environment variables (most notably, `$NORDUGRID_LOCATION` and `$GLOBUS_LOCATION`) by doing the following:

```
cd nordugrid-standalone-<version>
source setup.sh
```

If you are working in a C shell, use `setup.csh` instead. Upon first execution, this will print a lot of informational output. Make sure there are no "error" or "failure" messages.

2.2.4 Certificates

The standalone installation will create a directory `$NORDUGRID_LOCATION/share/certificates` which has to be populated with all the certificates you will need in your work. Follow the steps in Section 2.1.5, describing what has to be downloaded and where it has to be installed. Refer to Appendix A for instructions how to work with RPM files without having system administrator privileges, or simply use the provided tarballs instead of RPMs.

> The standard location defined by Globus for the certificates is `/etc/grid-security/certificates`. However, if you have no system administrator privileges, you must either relocate the certificates to your `$NORDUGRID_LOCATION/share/certificates` directory, or create `$HOME/.globus/certificates` directory and store the certificates there. The third option is to define the `X509_CERT_DIR` variable, pointing to any directory where you prefer to store the certificates.

2.2.5 Request a Certificate

If you do not have a personal certificate, refer to Section 2.1.6 for instructions on what has to be downloaded and how to issue a certificate request. Just as recommended in the previous Section, take care of relocating the files from the default `/etc/grid-security/certificates` directory to one of the following locations:

```
$NORDUGRID_LOCATION/share/certificates
$HOME/.globus/certificates
$X509_CERT_DIR
```

2.2.6 Launch a Grid Session

Log into the Grid:

```
grid-proxy-init
```

Type your password when requested, and enjoy the Grid world!

Chapter 3

Getting Access to Grid Resources

Sections 2.1.5 and 2.1.6 touch the issue of authentication and authorization on the Grid. In order to be able to use the NorduGrid middleware in most basic way, you don't have to know much more than explained there. However, more complex tasks may arise, and this Section attempts to provide bits of knowledge which will help you in understanding the access control issues. For a description of the Grid login procedure, refer to Section 5.1.

3.1 Authentication: Grid Certificates

In a Grid environment such as the NorduGrid, users normally don't have local password-protected accounts on computing resources they intend to use. You should hold instead an electronic certificate, which ensures unique authentication. Possession of a certificate, however, does not automatically authorize you to use all the Grid resources. Access control for the computing resources is a matter of a local policy: site administrators retain the full control of choosing which Grid user is allowed to use local resources. Typically, such local authorization process is done by mapping an accepted set of Grid users onto a set of local user accounts.

Since Grid services act on behalf of users, such services and all the resources which provide them must be certified as well.

Certification is done by trusted authorities of the Grid, called **Certificate Authority** (CA). Typically, there is one CA per country. For a non-authoritative list of CAs, refer to Appendix B.

The Grid utilizes public key, or asymmetric, cryptography for authentication of users, resources and services [7]. According to the basics of the public key cryptography, each resource on the Grid has a key pair: a public and a private key. The public key is, quite naturally, made public while the private key must be kept secret. Encryption and authorization is performed using the public key, while decryption and digital signature is performed with the private key.

It is important to note that generating a key pair does not automatically provide you access to the Grid resources. A CA needs to sign your key pair, thus confirming your identity. This signing procedure of the CA is often referred to as *"issuing a certificate"*.

Within the Globus era, the key file (e.g., `userkey.pem`) and the certificate file (e.g., `usercert.pem`) correspond to the key pair of the public-key cryptography. The `userkey.pem` file (or `resourcekey.pem` when a Grid resource is concerned) contains the private key encrypted with your password (called *"pass phrase"* by Globus). The certificate file (`usercert.pem`) contains your public key together with additional important information such as the *Subject Name* (SN) of the certificate holder, the name of the signing CA, and a digital signature of this CA. The important role of the CA is to establish a trustful connection between the identity of the user and the public key in the certificate file. The digital signature of the CA in the user's certificate file officially declares that the public key in the file belongs to the specific user (Subject Name). The certificate files are encoded with the x.509 [8] format.

In order to obtain a valid "passport" to the Grid, you need to create a key pair and submit your public key to your CA (this process is called as a *"certificate request"*) for a signature. The CA will follow its certificate policy and upon successful evaluation of your request your public key will be signed and posted back to you.

As it was mentioned before, all the resources (i.e. gatekeepers, users, services) require a CA-signed key pair to be able to operate on the Grid.

An unsigned key par is generated by the following command:

```
grid-cert-request
```

The created files are placed by default in your `.globus` directory. The `userkey.pem` holds your private key encoded with your pass phrase (you are prompted to supply this pass phrase during the key pair generation). This file must only be readable by its owner. The `usercert_request.pem` file contains your unsigned public key together with your Subject Name and the name of your CA. This file should be mailed to the CA. The `grid-cert-request` creates an empty `usercert.pem` file as well, just as a placeholder, which later can be overwritten with your CA-signed certificate.

> Please always remember that a Grid passport consists of two files, the private key file and the public certificate file. You need to have both of them, the certificate file (`usercert.pem`) alone is not enough for the Grid. If you loose one of your key files, you will have to regenerate a new CA-signed key pair.

You can use the `openssl` [9] cryptography toolkit and the Globus provided commands to create, check and convert between different formats, and to manipulate your certificate files (actually, the Globus commands are just a friendly interface to the `openssl` toolkit). For further information, please read the `man` pages for `openssl`, `verify` and `x509` commands, or use the Globus commands with the `-help` option.

3.1.1 Working with certificates: examples

Certificate request: the following command creates a key pair for a user or a gatekeeper and prepares a formal certificate request

```
grid-cert-request
```

Change pass phrase of the certificate: use this command to change the pass phrase of the private key file `userkey.pem`:

```
grid-change-pass-phrase -file userkey.pem
```

> Please note that once you forget the pass phrase, the certificate has to be requested anew, as there is no way to re-generate a password.

Inspect a certificate: to print all the information from the public certificate file `usercert.pem`, do the following

```
grid-cert-info -file usercert.pem -all
```

Duplicate a certificate with a new pass phrase: using this tool, your private key is encoded with a new pass phrase and stored in the `new_userkey.pem` file (first it asks for your old pass phrase, then twice for the new).

```
openssl rsa -in userkey.pem -des3 -out new_userkey.pem
```

Verify the certificate: to verify the `usercert.pem` certificate using the public key of the issuing CA, which is supposed to be in located in the specified `CApath`, do this:

```
openssl verify -CApath /etc/grid-security/certificates/ usercert.pem
```

Dump contents of the certificate: with this command you can display the contents of the `usercert.pem` certificate

```
openssl x509 -noout -text -in usercert.pem
```

Convert your Grid certificate into a Web one: to convert your certificate from the original `pem` format to `pkcs12` one, which can be used to authenticate yourself on the Web via Internet browsers, issue:

```
openssl pkcs12 -export -in usercert.pem -inkey userkey.pem -out cert.p12
```

3.2 Authorization: The Virtual Organizations

> **NB:** possession of a Grid certificate, even signed by a legal CA, does not automatically allow you to use any Grid resources!

As it was mentioned above, in order to be able to do such things as copy files over the Grid, or submit Grid jobs, your identity should be mapped to a local account on each resource – be it a computing cluster or a storage facility. As long as this mapping is done, you are *authorized* to use a resource as any local user.

To facilitate and automate such mapping, NorduGrid have set up a collective authorization method, the NorduGrid Virtual Organization (VO),, following practices of other Grid testbeds. This VO maintains a list of people which are authorized to use the NorduGrid resources. The VO tools provide an automatic method for the sites to easily maintain the *"VO member"↔"local Unix user"* mappings. These tools periodically query the VO user database and automatically generate the local *grid-mapfiles* following the local policy formulated in the VO-tools configuration file. The automatic mapping does not violate the site autonomy, because the site administrators retain a full control over their systems thanks to the possibility of denying access to "unwished" Grid users in the NorduGrid VO-tools configuration file.

The database of the VO is maintained by the *VO managers*. Their responsibility is to add, delete or modify user entries.

> You must contact a local NorduGrid VO manager in order to be added to the NorduGrid VO. The contact information can be found at the NorduGrid Web site. If you are not a Nordic countries resident, you may be added to the *Guests* VO.

The NorduGrid VO supports the creation of groups. A group is a subset of the a VO and is maintained by an appointed group manager. The group manager selects members of the group out of the full VO database. With the existence of the user groups, the site administrators can implement group based mappings, such that all the members of a certain group are mapped to the same local Unix user, in addition to the default user-based mappings.

Technically, the VO database is stored in an LDAP [10] database. For this purpose, a GSI [11] enabled OpenLDAP server is used, providing an entry and attribute level access control, based on the Grid certificates. The database managers, being authenticated and authorized through their certificates, make use of the OpenLDAP command line tools in order to add, delete or modify entries in the VO database. The NorduGrid sites periodically (4 times a day) run the `nordugridmap` utility in order to query the VO LDAP server and automatically create/update local user mappings according to a site policy (as defined in a `nordugridmap.conf` configuration file).

Chapter 4

Describing Grid Tasks

To describe a task to be submitted to the NorduGrid resources, a special scripting language is used. It is strongly based on the *Resource Specification Language (RSL)*, developed by the Globus project [12]. Using RSL allows passing job options and definitions to resource management systems. The NorduGrid architecture certain extensions to the RSL. This concerns not only introduction of new attributes, but also differentiation between two levels of the job options specifications:

User-side RSL, i.e., the set of attributes specified by a user in a job-specific file. This file is interpreted by the *User Interface (UI)* [4], and after the necessary modifications is passed to the *Grid Manager (GM)* [2]

GM-side RSL, i.e., the set of attributes pre-processed by the UI, and ready to be interpreted by the GM

As a user, you have to know only the user-side part, and utilize it to describe your Grid tasks.

In what follows, the description of the NorduGrid-extended RSL, further denoted as **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"

4.1 Task Description Language: xRSL

For a complete description of Globus RSL, see reference [12]. xRSL uses the same syntax conventions, but changes the meaning and interpretation of some attributes.

A Grid task is described by the mean of xRSL attributes, which can be either passed via a command-line, or, more conveniently, be collected a so-called xRSL-file (suggested extension *.xrsl*). Such a file contains a plain list of attribute assignment strings (*relations*) and boolean operands "&" (for AND) and "|" (for OR). Attribute names are case-insensitive.

A **relation** associates an attribute name with a value:

```
executable=''a.out''
```

Here `executable` is an attribute name, and `a.out` is its value. An attribute can have several values (blank-separated list):

```
arguments=‘‘abc’’ ‘‘123’’ ‘‘dir/file’’
```

or even correspond to pairs of values:

```
inputFiles=(‘‘thescript’’ ‘‘http://my.site.org/taskX/script1.sh’’)
           (‘‘my.idx’’ $HOME/index/the_index.bin)
           (‘‘thescript’’ ‘‘’’))
```

In the examples above, some strings are quoted, while some are not. When explicitly quoted, a string may contain any character. However, most literals used in job description (e.g. attribute names themselves, file names etc) don't have to be quoted, if they don't contain **special characters**.

The special characters are:

```
+   &   |   (   )   =   <   >   !   "   ’   ^   #   $
```

To quote a string containing special characters, you can use pairs of either single or double quotes. If your string, however, contains both such characters, you can define any character as an own delimiter, by preceding it with the "carat" (^) character: `jobName=^*My ‘‘good’’ job^*` makes use of a carat-escaped asterisk as a delimiter.

Typically, an xRSL job description starts with an ampersand ("&") , to indicate implicit **conjunction** (AND style) of all the attribute relations, enclosed in parentheses:

```
&(attribute=’’value1’’)(attribute=’’value2’’)...
```

Whenever a **disjunct**-request (OR style) of two or more attributes is needed, the following construction can be used:

```
(|(attribute=’’value1’’)(attribute=’’value2’’)...)
```

In expressions, the following operands are allowed:

```
=   !=   >   <   >=   <=
```

Commented lines should start with "(*" and be closed with "*)":

```
(*attribute=’’value1’’*)
```

Comments can not be nested.

Multiple job descriptions in one file are realized via a standard Globus RSL multi-request operand "+", which should precede the multiple job descriptions:

```
+(&(...))(&(...))(&(...))
```

The xRSL attributes can be written in a single string, or split in lines arbitrary; blank spaces between `(attribute=’’value’’)` pairs and inside relations are ignored.

4.1.1 URLs

File locations in NorduGrid can be specified both as local file names, and as Internet standard *Uniform Resource Locators (URL)*. However, there are some additional *options*, used by the Grid Manager.

The following transfer protocols and metadata servers are supported:

- `ftp` – ordinary *File Transfer Protocol (FTP)*
- `gsiftp` – GridFTP, the Globus-enhanced FTP

- `http` – ordinary *Hyper-Text Transfer Protocol (HTTP)*
- `https` – HTTP with Globus GSI authentication
- `ldap` – ordinary *Lightweight Data Access Protocol (LDAP)* [10]
- `rc` – Globus *Replica Catalog (RC)* [13]
- `rls` – Globus/EDG *Replica Location Service (RLS)* [14]
- `file` – local file access

An URL can be used in a standard form, i.e.

```
<protocol>://host[:port]/<file>
```

Or, to enhance the performance, it can have additional options:

```
<protocol>://host[:port][;option[;option[...]]]/<file>
```

For a metadata service URL, construction is the following:

```
rc://[location[|location[...]]@]<host>[:port]/<DN>/<lfn>
rls://[url[|url[...]]@]<host>[:port]/<lfn>
```

Here the URL components are:

`location`	`<location_name_in_RC>[;option[;option[...]]]`
`host[:port]`	IP address of a server
`DN`	Distinguished Name (as in LDAP) of an RC collection
`lfn`	Logical File Name
`url`	URL of the file as registered in RLS
`file`	local to the host file name with a full path

The following options are supported:

`threads=<number>`	specifies number of parallel streams to be used by GridFTP; default value is 1, maximal value is 10
`cache=yes|no`	indicates whether the GM should cache the file; default is `yes`
`secure=yes|no`	indicates whether the GridFTP data channel should be encrypted; default is `no`

Local files are referred by specifying either location relative to the job submission working directory, or by an absolute path (the one that starts with "/"), preceded with a `file://` prefix.

Examples of URLs are:

```
http://grid.domain.org/dir/script.sh
gsiftp://grid.domain.org:2811;threads=10/dir/input_12378.dat
ldap://grid.domain.org:389/lc=collection1,rc=Nordugrid,dc=nordugrid,dc=org
rc://grid.domain.org/lc=collection1,rc=Nordugrid,dc=nordugrid,dc=org/zebra/f1.zebra
file:///home/auser/griddir/steer.cra
```

4.1.2 Task Description: Attributes

A task is typically described by a binary executable to be processed by a computer, parameters passed to that executable, and requirements which must be met by a system to be able to execute the task. For example, a simple "Hello World" task description would specify `/bin/echo` as the executable and "Hello World" as the argument. In xRSL language it will look as follows:

```
&(executable=‘‘/bin/echo’’)(arguments=‘‘Hello World’’)
```

In a more complex case, a binary executable may have to be downloaded from a remote location, process an input data and create an output file. This task specification will have to include locations of all three files: executable, input and output, and perhaps few other requirements, e.g., the necessary disk space and an e-mail address to which the notification of the task completion should be sent. Such an xRSL file may be written like the following:

```
&
(executable=‘‘myprog’’)
(inputFiles=
   (‘‘myprog’’ ‘‘http://www.myserver.org/myfiles/myprog’’)
   (‘‘myinput’’ ‘‘gsiftp://www.mystorage.org/data/file007.dat’’)
)
(outputFiles=
   (‘‘myoutput’’ ‘‘gsiftp://www.mystorage.org/results/file007.res’’)
)
(disk=1000)
(notify=‘‘e myname@mydomain.org’’)
```

As you can see, the xRSL attribute names are largely self-explanatory. xRSL handles most possible task description attributes, and the next Section contains the comprehensive list of them, complete with usage examples.

Attributes

The following attributes can be specified in a user's xRSL file or string for job description.

executable

Usage: `(executable=<string>)`

Example: `(executable="myprog.exe")`

The executable to be submitted to LRMS.

`string` file name (including path), local to the computing element (CE)

If an executable has to be transferred from the submission node, it has to be specified in the `inputFiles` list, otherwise it will be added to that list by the UI.

If the file name starts with a leading slash ("`/`"), it is considered to be **the full path to the executable at a CE**; otherwise the location of the file is **relative** to the session directory (where job input and files are stored). If the attribute's value starts with an environment variable ("`$...`"), the value of this variable is resolved locally, but if it is enclosed in double quotes, it will be resolved at the remote computing element:
`(executable=$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

For more discussion on practical use of this attribute, refer to Section 4.2.2.

arguments

Usage: `(arguments=<string> [string] ... )`

Example: `(arguments="10000" $(ATLAS)/input.dat)`

List of the arguments for the executable.

`string` an argument

inputFiles

Usage: `(inputFiles=(<filename> <location>) ... )`

Example:
```
(inputFiles=(file1.dat gsiftp://grid.quark.lu.se/scratch/test.dat)
            (file2.dat /scratch/bigfile.dat)
            (file3.dat ""))
```

List of files to be copied to the computing element before the execution.

`filename` file name, local to the computing element and always relative to the session directory

`location` location of the file (`gsiftp`, `https`, `ftp`, `http` URLs, or a path, local to the submission node). If void (""), the input file is taken from the submission directory.

If the list does not contain the standard input file (as specified by `stdin`) and/or the executable file (as specified by `executable` if the given name), the UI appends these files to the list.

executables

Usage: `(executables=<string> [string] ...)`

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 is relative to the session directory) is not listed, it will be added to the list by the UI.

cache

Usage: `(cache=yes|no)`

Example: `(cache=yes)`

Specifies whether input files specified in the `inputFiles` should be placed by default in the cache or not. This does not affect files described by the `executable`, which will be placed in the session directory always.

If not specified, default value is "yes".

outputFiles

Usage: `(outputFiles=(<string> <URL>) ... )`

Example: `(outputFiles=(file1.dat gsiftp://grid.uio.no/storage/file_num_11)`
`(file2 rc://grid.fi.uib.no/group1/result2) )`

List of files to be retrieved by the user or uploaded by the GM and registered in a Replica Catalog.

`string` file name, local to the *Computing Element (CE)*

`URL` URL of the remote file (`gsiftp`, `https`, `ftp`, `http` or a Replica Catalog pseudo-URL); if void (""), the file is kept for manual retrieval.

Using a RC pseudo-URL (see Section 4.1.1), you should make sure that the location is already defined in the RC. If few locations are specified, only those found in the RC will be used. GM will store output files in **one** location only. If the first one in the list fails, it would try the next. If no locations are specified, all found in the RC will be used.

If in RC pseudo-URL the component `host[:port]/DN` is not specified, the one given in the `replicaCollection` attribute is used.

If the list does not contain standard output and/or standard error files (as specified by `stdout/stderr`), the UI appends these file names to the list. If the `<URL>` is not specified (void, ""), files will be downloaded by the user via the UI.

cpuTime

Usage: `(cpuTime=<time>)`

Example: `(cpuTime=240)`

Maximal CPU time request for the job.

`time` time (minutes)

If only number is specified, the time is assumed to be minutes. Otherwise, a free format is accepted, i.e., any of the following will be interpreted properly:

```
1 week
3 days
2 days, 12 hours
1 hour, 30 minutes
36 hours
9 days
240 minutes
```

memory

Usage: `(memory=<integer>)`

Example: `(memory>=500)`

Memory required for the job.

`integer` size (Mbytes)

disk

Usage: `(disk=<integer>)`

Example: `(disk=500)`

Disk space required for the job.

`integer` disk space ,Mbytes

runTimeEnvironment

Usage: `(runTimeEnvironment=<string>)`

Example: `(runTimeEnvironment=Atlas-1.1.0)`

Required runtime environment

`string` environment name

The site to submit the job to will be chosen by the UI among those advertising specified runtime environments. Before starting the job, the GM will set up environment variables and paths according to those requested.

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))`.

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.

Use the "`>=`" operator to request a version "equal or higher".

middleware

Usage: `(middleware=<string>)`

Example: `(middleware=NorduGrid-0.3.99)`

Required middleware.

`string` Grid middleware name

The site to submit the job to will be chosen by the UI among those advertising specified middleware. Usage is identical to that of the `runTimeEnvironment`. Use the "`>=`" operator to request a version "equal or higher". Request `(middleware=nordugrid)` defaults to `(middleware>=nordugrid-0.0.0.0)`.

stdin

Usage: `(stdin=<string>)`

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 UI.

stdout

Usage: `(stdout=<string>)`

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 UI. If the standard output is not defined, UI assigns a name.

stderr

Usage: `(stderr=<string>)`

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 UI. If the standard error is not defined, UI assigns a name.

join

Usage: `(join=yes|no)`

Example: `(join=yes)`

If "yes", joins `stderr` and `stdout` files into the `stdout` one. Default is `no`.

gmlog

Usage: `(gmlog=<string>)`

Example: `(gmlog=myjob.log)`

The job log file, containing all the job-related messages from the GM.

`string` file name, local to the computing element and relative to the session directory

The job log file should be listed as an `outputFiles` attribute; otherwise it will be forced to that list by the UI.

jobName

Usage: `(jobName=<string>)`

Example: `(jobName=MyJob)`

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 UI. It is also available through the Information System.

ftpThreads

Usage: `(ftpThreads=<integer>)`

Example: `(ftpThreads=4)`

Defines how many parallel streams will be used by the GM during `gsiftp` transfers of files.

`integer` a number from 1 to 10

If not specified, parallelism is not used.

cluster

Usage: `(cluster=<string>)`

Example: `(cluster=nbi)`

The name of the execution cluster.

`string` known cluster name, or a substring of it

Use this attribute to explicitly force job submission to a cluster, or to avoid such. The job will not be submitted if the cluster does not satisfy other requirements of your job. Disjunct-requests of the kind `(|(cluster=clus1)(cluster=clus2))` are supported. To exclude a cluster, use `(cluster!=clus3)`.

queue

Usage: `(queue=<string>)`

Example: `(queue=pclong)`

The name of the remote batch queue.

`string` known queue name

Use this attribute to explicitly force job submission to a queue.

startTime

Usage: `(startTime=<time>)`

Example: `(startTime="2002-05-25 21:30")`

Time to start job execution at a worker node. Unless you have computing nodes pre-allocated by some other means, this does not guarantee that the job will actually start running at the specified time; however it guarantees that it will not be launched before.

`time` time string, YYYY-MM-DD hh:mm:ss

lifeTime

Usage: `(lifeTime=<time>)`

Example: `(lifeTime=60)`

Maximal time to keep job files (the session directory) on the gatekeeper upon job completion.

`time` time (days)

Typical life time is 1 day (24 hours). Specified life time can not exceed local settings.

notify

Usage: `(notify=<string> [string] ... )`

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] 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 (CANCELED)

No more than 3 e-mail addresses per status change accepted.

replicaCollection

Usage: `(replicaCollection=<URL>)`

Example:
```
(replicaCollection="ldap://grid.uio.no:389/lc=TestCollection,
                    rc=NorduGrid,nordugrid,dc=org")
```

Location of a logical collection in the Replica Catalog.

`URL` LDAP directory specified as an URL (`ldap://host[:port]/dn`)

rerun

Usage: `(rerun=<integer>)`

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 2.

architecture

Usage: `(architecture=<string>)`

Example: `(architecture=i686)`

Request a specific architecture.

`string` architecture (e.g., as produced by uname -a)

nodeAccess

Usage: `(nodeAccess=inbound|outbound)`

Example: `(nodeAccess=inbound)`

Request cluster nodes with inbound or outbound IP connectivity. If both are needed, a conjunct request should be specified.

dryRun

Usage: `(dryRun=yes|no)`

Example: `(dryRun=yes)`

If "yes", do dry-run: RSL is parsed, but no job submission to LRMS is made. Use for xRSL validation.

rsl_substitution

Usage: `(rsl_substitution=(<string1> <string2>))`

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 simplify xRSL editing. 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`.

environment

Usage: `(environment=(<VAR> <string>) [(<VAR> <string>)] ... )`

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.

count

Usage: `(count=<integer>)`

Example: `(count=4)`

Specifies amount of sub-jobs to be submitted for parallel tasks.

`integer` a number (default is 1)

4.2 Examples

In this section you can find some examples of xRSL job description.

While the entire xRSL can be passed to the `ngsub` as a single string at a command line, it is much more convenient to create a file with an arbitrary name, containing the xRSL string, and pass the file using `ngsub -f filename`. Formatting of the file is not important, as long as it does not contain Microsoft/DOS linefeeds.

Attribute names are case-insensitive, that is, you can use `CPUTime` or `cputime` interchangeably. The only attribute wich **must** be specified is `executable`, others can be added by the user depending on the needs, in no particular order.

4.2.1 Hello World

The simplest way to say "Hello World" is to assume that any cluster on the Grid has the Unix `echo` executable in the `/bin/` directory. In most cases it will work, so the following xRSL file can be composed:

```
&
(* main executable of the task *)
 (executable=/bin/echo)
(* arguments for the main executable *)
 (arguments="Hello World" )
(* standard output will be redirected to this file: *)
 (stdout="hello.txt")
(* standard error will be redirected to this file: *)
 (stderr="hello.err")
(* Grid Manager auxilliary logs will be stored in this directory: *)
 (gmlog="gridlog")
(* give job a distinct name for easy monitoring *)
 (jobname="My Hello Grid")
(* instruct cluster that your job should be placed in a queue with *)
(* the sufficient time limit for the job to get completed *)
 (cputime=5)
(* choose only those clusters which have a proper NorduGrid installation *)
 (middleware>="nordugrid-0.3.24")
```

In this example, strictly speaking, only the `executable` and `arguments` are needed to describe the task. However, as the job will be executed in batch (like everything on the Grid), you will never see the message "Hello World" on your screen. Instead, you should instruct the Grid Manager to capture the standard output to a file, which you can later retrieve. This is how `stdout` attribute appears in the xRSL. It may happen that something goes wrong (e.g., `echo` is located in `/usr/bin/`), and an error will be produced. To be able to analyze the errors, users are advised to use `stderr` and `gmlog` attributes, which instruct the Grid Manager where to store possible error messages.

Job name is a convenient way to identify the task. Although every job is assigned a unique Grid ID, it is far from being intuitive, and as soon as you plan to submit more than one job, you should think of a set of good names for them, which would save you much time. The attribute `jobname` should be used to associate every job with a user-defined name.

Since the Grid will parse the task to a batch system, it is always a good idea to specify the needed CPU time. Some clusters are known to have the default execution time set to 0, which effectively kills any job. The `cputime` attribute takes the time value (in minutes) and passes it to the Grid Manager. Do not overestimate the time, as your job may end up in a "long" queue, and wait unnecessarily long time before being executed.

The last attribute in this example is `middleware`. This may be quite helpful in the environment where some clusters run untested versions of the middleware. When not sure what to specify, do `ngsub -v`, and use the printed version number.

4.2.2 An Own Executable

In the previous Section, the `executable` attribute had the value `/bin/echo`.

> A very important thing to understand is that the Grid Manager always expects the file specified by the `executable` attribute to be at the execution machine, not user's machine.

This means that if a user wants to submit a job executing an own tool `say_hello` and specifies in the xRSL text `(executable=say_hello)`, the Grid Manager will attempt to execute a file `say_hello` in the temporary session directory, local to the job. Naturally, this file will not appear there by magic. Instead, the User Interface, prior to job submission, will interpret such a request as if a user has the executable file `say_hello` in the current directory at the submission machine, and will upload the file to the destination. Absence of the leading slash (`/`) makes the User Interface assume that the file in question resides locally, in the path relative to the current directory.

So far, so good, but what if the executable resides not in the current directory, but elsewhere on your computer, or, even better, at a remote location? If this is the case, you have to instruct the User Interface how to get the file by using the `inputfiles` attribute:

```
&
(* main executable of the task *)
 (executable=say_hello)
(* arguments for the main executable *)
 (arguments="Hello again" )
(* where does the executable reside *)
 (inputfiles=(say_hello file:///home/john/bin/say_hello))
(* standard output will be redirected to this file: *)
 (stdout="hello.txt")
```

The `inputfiles` attribute instructs the User Interface to copy the file from `/home/john/bin/say_hello` on your local computer to `say_hello` in the session directory at the execution machine. If the file resides at an GridFTP or HTTP server, use `gsiftp://` or `http://` respectively instead of `file://`to specify the protocol.

4.2.3 Many Executables

The Grid Manager is capable of managing more than one executable file, and xRSL provides means for users to specify such files. A typical example is when a user script prepares and starts a pre-compiled binary. In this case, both files must be given executable permissions, but only the “main” script should be submitted for execution by the local system. To make this working, users must list all the files which need executable permissions in the `executables` attribute, as shown below:

```
&
(executable=run.sh)
(arguments=1664900 100000)
(executables=ffungen)
(inputFiles=(ffungen ""))
(outputFiles=(ffun.hbook gsiftp://hathi.hep.lu.se/test/flong/flong1.hbook))
(jobName=flong1)
(stdout=flong1.out)
(join=yes)
(notify="e oxana.smirnova@hep.lu.se")
(ftpThreads=6)
(middleware="NorduGrid-0.3.26")
```

Chapter 5

Grid Session

Before starting your first Grid session, check whether the following have been accomplished:

1. You have the NorduGrid client installed (see Chapter 2)
2. You have a valid Grid certificate (see Section 3.1)
3. You are a member of one of the NorduGrid-accepted VOs (see Section 3.2)

5.1 Logging Into The Grid

Access to the Grid resources in the Globus world is made via a so-called *proxy* – a kind of a temporary token, which you pass to the Grid services, allowing them to act on your behalf.

Initialization of the proxy is done via the standard Globus command:

```
grid-proxy-init
```

This command searches for your public certificate and private key in the default directory (`$HOME/.globus`), and upon a pass phrase confirmation, creates a file containing your Grid proxy in the `/tmp` directory of your computer. The proxy file name typically starts with ‘‘`x509up_u`’’, which is followed by your UNIX/Linux UID.

This Globus proxy is public, hence anybody having access to it can impersonate you. To minimize this risk, proxies have limited lifetime. By default, a proxy is valid for 24 hours. This, however, means that if your Grid task will last longer than the proxy validity, it will not be able to get finished because of the proxy expiration. Either for this reason, or, on contrary, if you'd like to have your proxy short-living, it is possible to create a proxy with a user-defined life span:

```
grid-proxy-init -valid 27:45
```

In this example, a proxy will leave 27 hours and 45 minutes.

To avoid troubles, please always try to make sure that your proxy has a validity period sufficient for your task to come to completion. However, it is not advised to generate proxies living several days.

The `grid-proxy-init` command has several options: e.g., you can select a location of the newly generated proxy, or specify another location for your Grid certificate and/or key. For detailed information on these options, use `grid-proxy-init -help`.

If you are using different computers to log in to the Grid, it is **strongly** advised **NOT TO COPY** your private key across the systems. Instead, use a dedicated machine to generate a proxy, and then copy the proxy file to another computer, preferably into your `$HOME` directory, and describe the new proxy location in the `X509_USER_PROXY` environment variable:

```
scp /tmp/x509up_u230 another.machine.org:myproxy
ssh another.machine.org
export X509_USER_PROXY=$HOME/myproxy
```

5.2 Logging Out

Logging out is performed by destruction of the proxy. You can either physically erase the corresponding file, or issue

```
grid-proxy-destroy
```

5.3 Working with jobs

Job submission on the NorduGrid is made via the *User Interface (UI)* part of the NorduGrid toolkit [4]. The UI provides a set of command line tools, allowing to submit jobs, trace their status, kill jobs, retrieve job output, and perform some other related functions. For a complete description, please refer to the UI manual [4].

5.3.1 Submit A Job

The `ngsub` command is the most essential one, as it is used for submitting jobs to the NorduGrid. The jobs are described using the extended resource description language (xRSL), see Section 4.

```
ngsub [options] [xrsl]
```

Options:

`-c, -cluster`	`[-]`textemname	explicitly select or reject a specific cluster
`-C, -clustlist`	`[-]`textemfilename	list of clusters to select or reject
`-g, -giisurl`	*url*	URL of a central Information System server
`-G, -giislist`	*filename*	list of GIIS URLs
`-f, -file`	*filename*	xrsl file describing the job to be submitted
`-o, -joblist`	*filename*	file where the job IDs will be stored
`-dryrun`		add dryrun option to the xRSL
`-dumpxrsl`		do not submit – dump transformed xRSL to stdout
`-t, -timeout`	*time*	timeout for queries (default 40 sec)
`-d, -debug`	*debuglevel*	0 = none, 1 = some, 2 = more, 3 = a lot
`-x, -anonymous`		use anonymous bind for queries
`-X, -gsi`		use GSI-GSSAPI bind for queries
`-v, -version`		print version information
`-h, -help`		print this help

Arguments:

`xrsl ...`	xrsl strings describing the jobs to be submitted

A simple *"Hello World"* job would look like:

```
ngsub '&(executable=''/bin/echo'')(arguments=''Hello World'')(stdout=''hello.txt'')'
```

> Hint: use single quotes for the xRSL string and double quotes for attribute values.

Such a request would submit the task to any available cluster, as there are no specific requirements specified. The job will be executed in batch mode, and the standard output will be written to a file `hello.txt` at the execution cluster. You will have to retrieve this file manually, using `ngget` command (Section 5.3.4).

If a submission was successful, the job ID will be printed by `ngsub`.

If a job is successfully submitted, a **job identifier** (*job ID*) is printed to standard output. This job ID uniquely identifies the job while it is being executed. A typical job ID looks like follows:

```
gsiftp://site.it.uni.org:2812/jobs/10308913211503407485
```

You should use this as a handle to refer to the job when doing other job manipulations, such as querying job status (`ngstat`, Section 5.3.2), killing it (`ngkill`, Section 5.3.5), re-submitting (`ngresub`, Section 5.3.6), or retrieving the result (`ngget`, Section 5.3.4).

> Every job ID is a valid URL for the job session directory. You can always use it to access the files related to the job, by using data management tools (see Section 5.4 and Chapter 6).

The job description in the xRSL format can be given either as an argument on the command line, as in the example above, or can be **read from a file** by using the `-f` option. Several jobs can be requested at the same time by giving more than one xRSL argument, or by repeating the `-f` option. It is also possible to mix xRSL arguments and `-f` options in the same `ngsub` command.

To **validate** your xRSL script without actually submitting a job, use the `-dryrun` option: it will capture possible syntax or other errors, but will instruct the GM not to submit the job for execution.

If the `-o` option is given, the job identifier is also written to a file with the specified filename. This file can later be used with the corresponding `-i` option of the other job manipulating User Interface commands.

```
ngsub -o my_jobid_list -f myjob.xrsl ngkill -i my_jobid_list
```

The `-c` option can be used to **force** a job to be submitted to a particular cluster, or to reject submission to a cluster. The matching is done by case insensitive substring match to the cluster name (i.e. hostname) or to the cluster alias name, as defined in the Information System. The `-c` option can be repeated several times, for example:

```
ngsub -c grid.nbi.dk -c grid.tsl.uu.se -f myjob.xrsl
```

This will submit a job to either `grid.nbi.dk` or `grid.tsl.uu.se`. For convenience, you may list the sites in a file, and use the `-C` option to refer to the whole list:

```
ngsub -C preferred_sites -f myjob.xrsl
```

If a cluster name or file name is preceded with a minus sign ("-"), this cluster (or the list) will be avoided during the submission. This gives a possibility to blacklist unwanted sites:

```
ngsub -C -blacklist -f myjob.xrsl
```

The `ngsub` command locates the available clusters by querying the Information System. By default, a list of the NorduGrid Information System **servers** is distributed with the middleware and is stored in `$NORDUGRID_LOCATION/etc/giislist`. However, a user is free to choose another set of servers, either by storing them in `$HOME/.nggiislist` file, or by specifying them via the `-g` option:

```
ngsub -g ldap://hostname[:port]/DN -f myjob.xrsl
```

You may prefer to store the list of GIIS servers in a file other than the default one, and use the `-G` option to instruct `ngsub` to contact those servers instead:

```
ngsub -G my_GIIS_list -f myjob.xrsl
```

If you would like to **trace** the process of resource discovery and requirements matching, a very useful option is `-d`. The following command:

```
ngsub -d 2 -f myjob.xrsl
```

will print out the steps taken by the User Interface to find the best cluster satisfying your job requirements.

It often happens that some sites that `ngsub` has to contact are slow to answer, or are down altogether. This will not prevent you from submitting a job, but will slow down the submission. To speed it up, you may want to specify a shorter timeout (default is 40 seconds) with the `-t` option:

```
ngsub -t 5 -f myjob.xrsl
```

5.3.2 Querying Job Status

The `ngstat` command is used for obtaining the status of jobs that have been submitted to NorduGrid.

```
ngstat [options] [job ...]
```

Options:

`-a, -all`		all jobs
`-i, -joblist`	*filename*	file containing a list of jobids
`-c, -clusters`		show information about clusters
`-C, -clustlist`	`[-]`textemfilename	list of clusters to select or reject
`-s, -status`	*statusstr*	only select jobs whose status is *statusstr*
`-g, -giisurl`	*url*	URL of a central Information System server
`-G, -giislist`	*filename*	list of GIIS URLs
`-q, -queues`		show information about clusters and queues
`-l, -long`		long format (extended information)
`-t, -timeout`	*time*	timeout for queries (default 40 sec)
`-d, -debug`	*debuglevel*	0 = none, 1 = some, 2 = more, 3 = a lot
`-x, -anonymous`		use anonymous bind for queries
`-X, -gsi`		use GSI-GSSAPI bind for queries
`-v, -version`		print version information
`-h, -help`		print this help

Arguments:

`job ...`	list of job IDs and/or jobnames

Typicaly, one should use `ngstat` with the job ID as printed by `ngstat`. If the job IDs were saved to a file, `ngstat -i filename` will give the list of all the jobs status. Many users prefer to use `ngstat -a` to display the status of all the jobs they ever submitted*. To check the status of jobs being on a particular stage of execution (e.g., only running jobs), use the `-s` option:

*Or, more precisely, since the last they ran `ngclean`

```
ngstat -s ‘‘INLRMS: R’’
```

The list of basic job status codes:

`ACCEPTED`	job submitted but not yet processed
`PREPARING`	input files are being retreived
`SUBMITTING`	interaction with LRMS ongoing
`INLRMS: Q`	job is queued by LRMS
`INLRMS: R`	job is running
`FINISHING`	output files are being transferred
`FINISHED`	job is finished
`CANCELING`	job is being cancelled
`DELETED`	job is removed due to expiration time

Most of the states can be appended with the "`: PENDING`" message, if the processing of the state have not been started yet. The `FINISHED` state may be followed by an error message if the job failed, starting with "`: FAILURE`" string.

The jobs are normally removed from the clusters if they were not retrieved within 24 hours after job end.

A standard output produced by `ngstat` is rather short, informing the user only about the job status. Use `ngstat -l` to receive the complete job information as stored in the system.

5.3.3 Capturing Job Output

It is often useful to monitor the job progress by checking what it prints on the standard output or error. The command `ngcat` assists here, capturing the corresponding information from the execution cluster and pasting it on the user's screen. It works both for running tasks and for the finished ones. This allows a user to check the output of the finished task without actually retreiving it.

```
ngcat [options] [job ...]
```

Options:

`-a, -all`		all jobs
`-i, -joblist`	*filename*	file containing a list of job IDs
`-c, -clusters`		show information about clusters
`-C, -clustlist`	`[-]`textemfilename	list of clusters to select or reject
`-s, -status`	*statusstr*	only select jobs whose status is *statusstr*
`-o, -stdout`		show the stdout of the job (default)
`-e, -stderr`		show the stderr of the job
`-l, -gridlog`		show the grid error log of the job
`-t, -timeout`	*time*	timeout for queries (default 40 sec)
`-d, -debug`	*debuglevel*	0 = none, 1 = some, 2 = more, 3 = a lot
`-x, -anonymous`		use anonymous bind for queries
`-X, -gsi`		use GSI-GSSAPI bind for queries
`-v, -version`		print version information
`-h, -help`		print this help

Arguments:

`job ...`	list of job IDs and/or jobnames

As one can see, `ngcat` can capture not only the standard output (`-o` option), but also the standard error (`-e` option) and the errors reported by the Grid Manager (`-l` option).

5.3.4 Retrieving Job Output

To retrieve the results of a finished job, the `ngget` command should be used. It will download the files specified by the `outputfiles` xRSL attribute (see Section 4.1.2) to the user's computer.

```
ngget [options] [job ...]
```

Options:

`-a, -all`		all jobs
`-i, -joblist`	*filename*	file containing a list of jobids
`-c, -cluster`	`[-]`textemname	explicitly select or reject a specific cluster
`-C, -clustlist`	`[-]`textemfilename	list of clusters to select or reject
`-s, -status`	*statusstr*	only select jobs whose status is *statusstr*
`-dir`	*dirname*	download directory (the job directory will be created in this directory)
`-j, -usejobname`		use the jobname instead of the digital ID as the job directory name
`-keep`		keep files on gatekeeper (do not clean)
`-t, -timeout`	*time*	timeout for queries (default 40 sec)
`-d, -debug`	*debuglevel*	0 = none, 1 = some, 2 = more, 3 = a lot
`-x, -anonymous`		use anonymous bind for queries
`-X, -gsi`		use GSI-GSSAPI bind for queries
`-v, -version`		print version information
`-h, -help`		print this help

Arguments:

`job ...`	list of job IDs and/or jobnames

The files to be retrieved by `ngget` should be described by the xRSL as follows:

```
(outputfiles=(file1 “”)(file2 “”))
```

That is, while the filenames must be listed, no output destination should be requested. This will tell the Grid Manager not to erase the files after job's end and allow the User Interface to download them.

> Files specified as `stdout`, `sterr` and the `gmlog` directory are always treated as `outputfiles` by the UI, which means you don't have to add them to the output files list.

By default, `ngget` will create in your current directory a new folder, with the same name as the remote session directory (typically, a numerical string). This new directory will contain all the files listed for download in your xRSL. If you would like to store the files in another location, use the `-dir` option. The option `-j` will assign your job name to the local directory with the output files (be careful not to call all the jobs same name when using this option).

When the job result is retrieved, the session directory is erased from the execution machine, and the job ID is removed from your list of submitted jobs. If you however want to keep the job for a while, use the `-k` option.

> Beware that the job results will be removed by the Grid Manager at the execution machine in 24 hours after the job completion independently of whether you retrieved the results or not.

5.3.5 Killing Jobs

It happens that a user wishes to cancel a job. This is done by using the `ngkill` command. A job can be killed at practically any stage of processing through the Grid.

```
ngkill [options] [job ...]
```

Options:

`-a, -all`		all jobs
`-i, -joblist`	*filename*	file containing a list of jobids
`-c, -clusters`		show information about clusters
`-C, -clustlist`	`[-]`textemfilename	list of clusters to select or reject
`-s, -status`	*statusstr*	only select jobs whose status is *statusstr*
`-keep`		keep files on gatekeeper (do not clean)
`-t, -timeout`	*time*	timeout for queries (default 40 sec)
`-d, -debug`	*debuglevel*	0 = none, 1 = some, 2 = more, 3 = a lot
`-x, -anonymous`		use anonymous bind for queries
`-X, -gsi`		use GSI-GSSAPI bind for queries
`-v, -version`		print version information
`-h, -help`		print this help

Arguments:

`job ...`	list of job IDs and/or jobnames

> Job cancellations are processed in an asynchronous manner, so it may take a few minutes before the job is actually cancelled.

5.3.6 Re-submitting Jobs

Quite often it happens that a user would like to re-submit a job, but has difficulties recovering the original job description xRSL file. This happens when xRSL files are created by scripts on-fly, and matching of xRSL to the job ID is not straightforward. The utility called `ngresub` helps in such situations, allowing users to resubmit jobs known only by their IDs.

> Only jobs where the `gmlog` attribute was given in the xRSL description (see Section 4.1.2) can be resubmitted.

```
ngresub [options] [job ...]
```

Options:

`-a, -all`		all jobs
`-i, -joblist`	*filename*	file containing a list of jobids
`-c, -cluster`	`[-]`textemname	explicitly select or reject a specific cluster
`-C, -clustlist`	`[-]`textemfilename	list of clusters to select or reject
`-s, -status`	*statusstr*	only select jobs whose status is *statusstr*
`-k, -kluster`	`[-]`textemname	explicitly select or reject a specific cluster as re-submission target
`-K, -Klustlist`	`[-]`textemfilename	list of clusters to select or reject as re-submission target
`-g, -giisurl`	*url*	URL of a central Information System server
`-G, -giislist`	*filename*	list of GIIS URLs
`-o, -joblist`	*filename*	file where the job IDs will be stored
`-dryrun`		add dryrun option to the xRSL
`-dumpxrsl`		do not submit – dump transformed xRSL to stdout
`-keep`		keep files on gatekeeper (do not clean)
`-t, -timeout`	*time*	timeout for queries (default 40 sec)
`-d, -debug`	*debuglevel*	0 = none, 1 = some, 2 = more, 3 = a lot
`-x, -anonymous`		use anonymous bind for queries
`-X, -gsi`		use GSI-GSSAPI bind for queries
`-v, -version`		print version information
`-h, -help`		print this help

Arguments:

`job ...`	list of job IDs and/or jobnames

> If the original job description contained input file locations specified as relative paths (non-URLs), the command must be issued in the same directory as the original `ngsub` instruction.

It is important to distinguish `-c` and `-k` options (as well as `-C` and `-K`). The former stands for `-cluster` and is used to instruct `ngresub` to look for jobIDs and descriptions only at a given cluster (or exclude a cluster). This is convenient to use together with the `-a` option, in case you would like to re-submit all the jobs which were originally sent to a specific cluster. The latter option, `-k`, stands for `-kluster`, and should be used to specify preferred re-submission target. For example, if you would like to re-submit all your jobs from one cluster to another, do:

```
ngresub -c grid1.ua.org -k grid2.ub.org
```

> If the re-submission was successful, the original job will be removed from the remote cluster unless you specify the `-keep` option.

5.3.7 Cleaning Up After Jobs

If a job fails, or you are not willing to retrieve the results for some reasons, a good practice for users is not to wait for the Grid Manager to clean up the job leftovers, but to use `ngclean` to release the disk space and to remove the job ID from the list of submitted jobs and from the Information System.

```
ngclean [options] [job ...]
```

Options:

`-a, -all`		all jobs
`-i, -joblist`	*filename*	file containing a list of jobids
`-c, -cluster`	`[-]`textemname	explicitly select or reject a specific cluster
`-C, -clustlist`	`[-]`textemfilename	list of clusters to select or reject
`-s, -status`	*statusstr*	only select jobs whose status is *statusstr*
`-f, -force`		removes the job ID from the local list even if the job is not found on the Grid
`-t, -timeout`	*time*	timeout for queries (default 40 sec)
`-d, -debug`	*debuglevel*	0 = none, 1 = some, 2 = more, 3 = a lot
`-x, -anonymous`		use anonymous bind for queries
`-X, -gsi`		use GSI-GSSAPI bind for queries
`-v, -version`		print version information
`-h, -help`		print this help

Arguments:

`job ...`	list of job IDs and/or jobnames

The `ngclean` should also be used whenever you wish to clean up the local list of submitted jobs, removing entries of old jobs removed by the Grid Managers. In this case, use the `-f` option.

5.3.8 Renewing User Proxy

Quite often, the user proxy expires while the job is still running (or waiting in a queue). In case such job has to upload output files to a Grid location (Storage Element), it will fail. By using the `ngrenew` command, users can upload a new proxy to the job. This can be done while a job is still running, thus preventing it from failing, or whithin 24 hours after the job end. In the latter case, the Grid Manager will attempt to finalize the job by uploading the output files to the desired location.

```
ngrenew [options] [job ...]
```

Options:

`-a, -all`		all jobs
`-i, -joblist`	*filename*	file containing a list of jobids
`-c, -cluster`	`[-]`textemname	explicitly select or reject a specific cluster
`-C, -clustlist`	`[-]`textemfilename	list of clusters to select or reject
`-s, -status`	*statusstr*	only select jobs whose status is *statusstr*
`-t, -timeout`	*time*	timeout for queries (default 40 sec)
`-d, -debug`	*debuglevel*	0 = none, 1 = some, 2 = more, 3 = a lot
`-x, -anonymous`		use anonymous bind for queries
`-X, -gsi`		use GSI-GSSAPI bind for queries
`-v, -version`		print version information
`-h, -help`		print this help

Arguments:

`job ...`	list of job IDs and/or jobnames

Prior to using `ngrenew`, take care of actually creating the new proxy:

```
grid-proxy-init -valid 24:00
```

Here `-valid` specifies for how long the proxy will be valid (see Section 5.1 for more details).

5.3.9 Synchronizing The Job ID List

If you are using User Interface installations on different machines, your local lists of submitted jobs will be different. To synchronise these lists with the information in the Information System, use the `ngrenew` command.

```
ngsync [options]
```

Options:

`-c, -cluster`	`[-]`textemname	explicitly select or reject a specific cluster
`-C, -clustlist`	`[-]`textemfilename	list of clusters to select or reject
`-g, -giisurl`	*url*	URL of a central Information System server
`-G, -giislist`	*filename*	list of GIIS URLs
`-f, -force`		dont ask for confirmation
`-t, -timeout`	*time*	timeout for queries (default 40 sec)
`-d, -debug`	*debuglevel*	0 = none, 1 = some, 2 = more, 3 = a lot
`-x, -anonymous`		use anonymous bind for queries
`-X, -gsi`		use GSI-GSSAPI bind for queries
`-v, -version`		print version information
`-h, -help`		print this help

> As the Information System has a certain latency, do not use `ngsync` immediately after submitting or killing a job, better wait for a few minutes.

5.4 Working with files

5.4.1 Copying Grid Files

The `ngcopy` is a powerful tool to copy files over the Grid. It is a part of the Grid Manager, but can be used by the User Interface as well.

```
ngcopy [options] [source] [destination]
```

Options:

`-h`		short help
`-v`		print version information
`-d`	*debuglevel*	0 = some, 1 = more, 2 = a lot
`-c`	*cache_path*	use cache
`-C`	*cache_data_path*	store cached data

Arguments:

`source`	source URL
`destination`	destination URL

Users can use `ngcopy` to copy files between a variety of different sources and destinations. Refer to Section 4.1.1 for the list of URLs accepted by this tool. Source URL can end with `''/''`. In that case, the whole fileset (directory) will be copied. Also, if the destination ends with `''/''`, it is extended with part of source URL after last `''/''`, thus allowing users to skip the destination file or directory name if it is meant to be identical to the source.

Since the job ID is nothing but a `gsiftp://` URL of the job top directory, you can use `ngcopy` to copy files from the job directory at any time.

5.4.2 Erasing Grid Files

Counterpart to the `ngcopy`, the `ngremove` command allows users to erase files at any location specified by a valid URL (see Section 4.1.1).

```
ngremove [options] [source]
```

Options:		
`-h`		short help
`-v`		print version information
`-d`	*debuglevel*	0 = some, 1 = more, 2 = a lot
`-c`		continue with meta-data even if it failed to delete real file
`-C`	*cache_data_path*	store cached data
Arguments:		
`source`		source URL

The most common use for `ngremove` is to erase the files in the Replica Catalog (see Section 6.2, as it will not only remove the physical instance, but also will clean up the logical record.

Chapter 6

Data Management

At this stage, data management possibilities in the NorduGrid system are rather limited. The basic operations are simple, one-by-one, file copy and removal (as explained in Section 5.4), with the eventual use of the Replica Catalog (Section 6.2) to assist in arranging files in logical collections, keep track of their replicas, and simplify storage resource discovery. The GSI-enabled `ncftp` client by Globus can also be used to interact with files on the Grid, available via GridFTP servers.

6.1 gsincftp

`gsincftp` is an attempt by Globus to create a basic GSI-enabled FTP client. It was never meant to be the Grid data management tool of choice, and does not include all possible protocol extensions needed for a Grid file transfer application. However, as no other similar client for the GridFTP server exists, people find it convenient to use `gsincftp` for basic data movement and remote file access operations.

To use the `gsincftp` client, you first need to ensure you have a valid proxy. Then you can log in to any NorduGrid cluster or Storage element by issuing

```
gsincftp cluster.dept.uni.org
```

This will open an interactive session, where a typical basic set of FTP client commands can be executed. For example, you can list the directory, browse through the directories tree, download, upload or remove files.

> Do not use `gsincftp` unless you know **exactly** what are you doing. Be particularly careful when uploading and deleting files: chances are that you destroy your own or others' work.

Every job you submit can be accessed via the `gsincftp` tool: simply use the host part of your job ID to log in and browse down the rest of the pass. For example, if your job ID is
`gsiftp://site.it.uni.org:2812/jobs/2115427105241848111`,
you can perfectly have a following session:

```
gsincftp site.it.uni.org
cd jobs/2115427105241848111
dir
get myfile
exit
```

Conveniently enough, tools `gsincftpput` and `gsincftpget` are available to upload and download files from a command line, thus allowing for scripting and bulk file movement. For example, the session above can be shortened to a single command (given you know there is a file named `myfile` in the job directory):

```
gsincftpget gsiftp://site.it.uni.org:2812/jobs/2115427105241848111/myfile
```

6.2 Replica Catalog in Examples

The NorduGrid Replica Catalog is nothing but slightly patched Globus Replica Catalog [13]. The support for it has been discontinued by Globus a while ago, and NorduGrid uses it just as a temporary solution for the distributed data catalogue. This Section only presents some usage examples and does not go into details of the Replica Catalog implementation, installation and support.

> Only authorised RC managers can perform operations which require write access. For details, contact `nordugrid-support@nordugrid.org`.

The Globus RC is an hierarchical database: the **catalogue** consists of **collections**, each collection consists of **logical file names** which correspond to one or more **physical files** at pre-defined **locations**. To register an arbitrary file in a RC, the following steps should be performed:

1. Ensure the collection exists
2. Ensure the physical location of the file is registered as one of the locations for the collection
3. Enter the file record into the RC, specifying the location and the file name **RELATIVE TO THE LOCATION**

> The limitation of the Globus RC is that it is impossible to assign arbitrary logical file names: a logical file name **must** be the actual file name, eventually prepend with the path relative to the registered location.

6.2.1 Define who you are

> This is the strongly recommended first step, simplifying authentication at the NorduGrid Replica Catalog

```
export GLOBUS_REPLICA_CATALOG_MANAGER="/O=Grid/O=NorduGrid/OU=nordugrid.org/CN=Jane Doe"
```

Or, for C shells:

```
setenv GLOBUS_REPLICA_CATALOG_MANAGER "/O=Grid/O=NorduGrid/OU=nordugrid.org/CN=Jane Doe"
```

The quoted string `/O=Grid/O=NorduGrid/OU=nordugrid.org/CN=Jane Doe` is your identity, as specified in the Subject of your Grid certificate. Use `grid-cert-info` command (Section 3.1.1) to find it.

6.2.2 Create a collection

```
globus-replica-management \
    -collection ldap://grid.uio.no/lc=A_Collection,rc=NorduGrid,dc=nordugrid,dc=org \
    -create
```

Here, `grid.uio.no` is the server address for the `NorduGrid` catalogue, and `A_Collection` is your new collection name. Since the RC is an LDAP database, collection URL must contain the LDAP-complying Distinguished Name, which includes the full NorduGrid RC name space (`dc=nordugrid,dc=org`). Default port number is 389, but if it is different, it should be specified in a standard IP manner: after the server address, separated by a column.

6.2.3 Add a location to a collection

```
globus-replica-management \
    -collection ldap://grid.uio.no/lc=A_Collection,rc=NorduGrid,dc=nordugrid,dc=org \
    -location CASTOR \
    -create gsiftp://wacdr002d.cern.ch/castor/cern.ch/atlas/transfer/dc1/
```

The location name `CASTOR` should later be used as a mount point, instead of the physical URL `gsiftp://...`

6.2.4 Upload and register a file to a collection

> You must have a valid Grid proxy (see Section 5.1) to perform this operation

```
ngcopy \
  gsiftp://grid.quark.lu.se/ATLAS/mydata.zebra \
  rc://grid.uio.no/lc=A_Collection,rc=NorduGrid,dc=nordugrid,dc=org/yourdata.zebra
```

In the example above, the first available registered location will be used to store the file. To specify an explicit destination, do:

```
ngcopy \
  gsiftp://grid.quark.lu.se/ATLAS/mydata.zebra \
  rc://CASTOR@grid.uio.no/lc=A_Collection,rc=NorduGrid,dc=nordugrid,dc=org/yourdata.zebra
```

6.2.5 Register existing at a location file to a collection

File registration includes three steps:

1. Create a logical file entry in the collection:

```
globus-replica-catalog \
    -host ldap://grid.uio.no/lc=A_collection,rc=NorduGrid,dc=nordugrid,dc=org \
    -logicalfile path/file1.zebra \
    -create size 0
```

2. Verify that file location (mount point) is registered in the collection (see Sections 6.2.8 and 6.2.9), otherwise it has to be added (see Section 6.2.3).

3. Register the physical location(s):

```
globus-replica-management \
    -collection ldap://grid.uio.no/lc=A_collection,rc=NorduGrid,dc=nordugrid,dc=org \
    -location CASTOR \
    -files filelist \
    -register
```

If the file size is unknown, specify "0" as shown above. Logical entries can only be added one by one, while physical location(s) must be added via a `filelist`, one line per file:

```
path/file1.zebra
path/file2.zebra
```

Note the prepended `path`, relative to the registered location: it should be possible to construct a correct URL as `${LOCATION}${FILENAME}`.

6.2.6 Remove and unregister a file

```
ngremove \
   rc://grid.uio.no/lc=A_Collection,rc=NorduGrid,dc=nordugrid,dc=org/yourdata.zebra
```

6.2.7 Remove a location from a collection

```
globus-replica-management \
    -collection ldap://grid.uio.no/lc=A_collection,rc=NorduGrid,dc=nordugrid,dc=org \
    -location CASTOR \
    -delete
```

6.2.8 Find locations (URL prefixes) for a collection

```
ldapsearch -h grid.uio.no -p 389 \
   -b "lc=A_Collection,rc=NorduGrid,dc=nordugrid,dc=org" \
   "(&(objectclass=GlobusReplicaLocation)(uc=*))" uc
```

or, using the Globus interface to LDAP:

```
globus-replica-catalog \
    -host ldap://grid.uio.no:389/lc=A_Collection,rc=NorduGrid,dc=nordugrid,dc=org \
    -collection \
    -list-locations uc
```

6.2.9 Find a URL prefix for a location known by name

```
globus-replica-catalog \
    -host ldap://grid.uio.no:389/lc=A_Collection,rc=NorduGrid,dc=nordugrid,dc=org \
    -location CASTOR \
    -list-attributes uc
```

Chapter 7

The Grid Monitor

The Grid Monitor is perhaps the most convenient utility that allows users and sysadmins alike to check and monitor the status of the NorduGrid facility, status of each job, display user-specific information and otherwise obtain in an intuitive way the information stored in the NorduGrid Information System [3]. It is implemented as a set of pop-up Web pages, connected by hyperlinks, making it easy to browse and navigate. Some limited searching capacity is also available, through the "Match-it-yourself" module.

The structure of the Grid Monitor to great extent follows that of the NorduGrid Information System [3]. The basic objects are defined by the following schema's objectclasses:

- nordugrid-cluster: a cluster
- nordugrid-queue: a queue at the cluster, accessible by the NorduGrid users
- nordugrid-job: a NorduGrid job, associated with a queue
- nordugrid-authuser: a user, authorized to submit jobs to a given queue

The Grid Monitor also uses the NorduGrid Virtual Organisation (VO) organisationalPerson and Storage Element nordugrid-se objectclasses, and their attributes.

For each objectclass, either an essential subset of attributes, or the whole list of them, is presented in an easily accessible inter-linked manner. This is realized as a set of windows, each being associated with a corresponding module. There are nine major modules :

1) An overall Grid Monitor
2) Cluster Description
3) Queue Details
4) Job Information
5) User Information
6) Attributes Overview
7) Customizable Display Tool ("Match-it-yourself")
8) List of Storage Facilities
9) List of Users

Each module displays both dynamic and static information: for example, a queue name is static, while the amount of running jobs in this queue is dynamic. Most of the displayed objects are linked to appropriate modules, such that with a simple mouse click, a user can launch another module, expanding the information about the corresponding object or attribute. Each such module opens in an own window, and gives access to other modules in turn, providing thus a rather intuitive browsing.

In what follows, these modules are described in details, giving an overview of their functionality and usage hints.

7.1 The Grid Monitor

The basic module, providing access to the most required information, is the Grid Monitor, showing the overall status of the system. It serves as a starting point for browsing the system information. The purpose of this module is to give a quick overview of the current status of the NorduGrid by showing the list of the available clusters and the most essential information about them: an alias, number of working processors, number of occupied processors and number of queueing jobs. In the current implementation, the main Grid Monitor window contains also the link to the user base of the NorduGrid. Figure 7.1 shows a screenshot of the running monitor. All the information shown is dynamic, including organizational names (countries in this case).

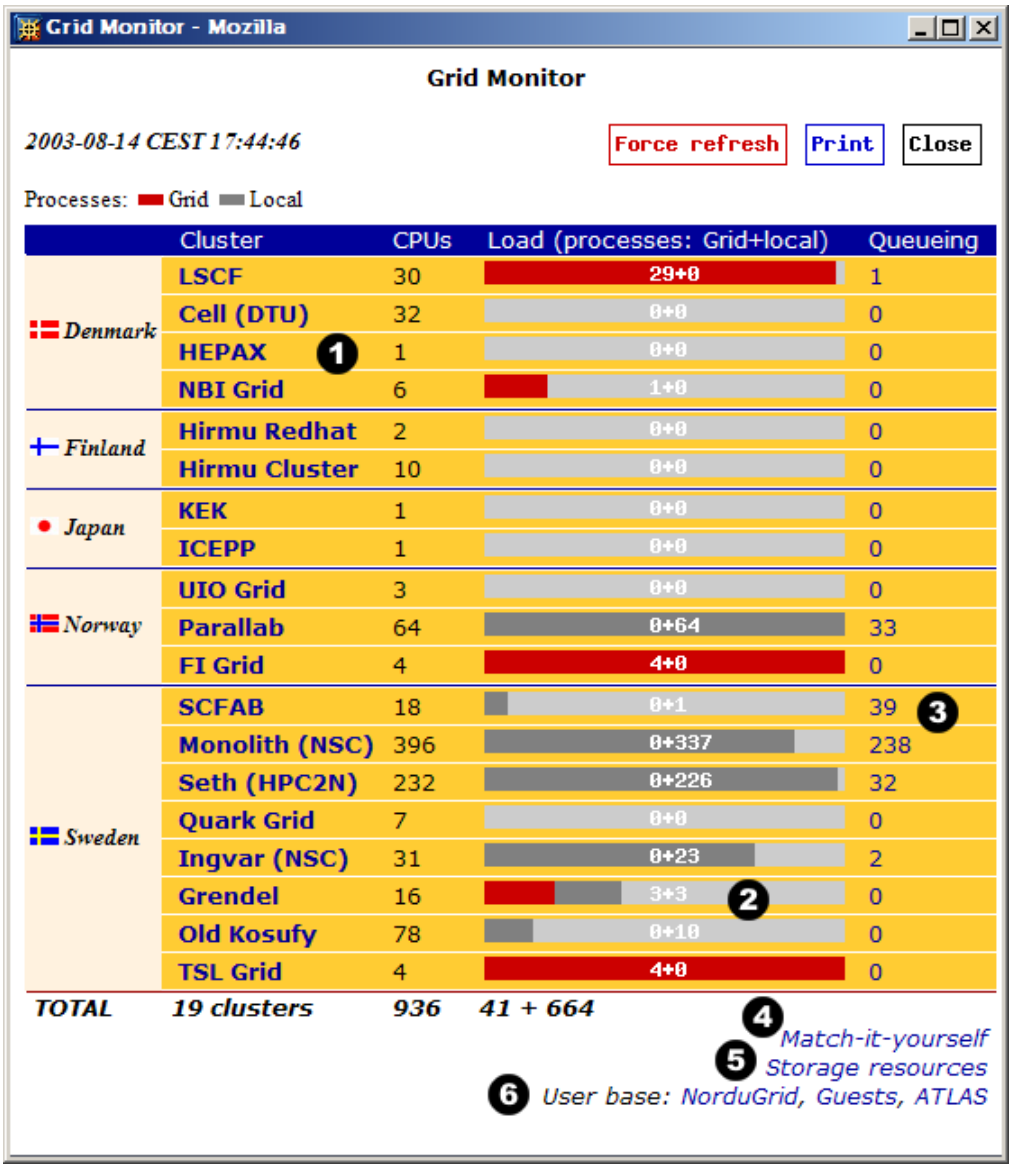


Figure 7.1: The Grid Monitor

In Figure 7.1, the numbered tags indicate clickable objects as explained below:

1) Cluster: a cluster alias, linked to the cluster description module (Section 7.2), which provides complete information about the current status of a cluster.
2) Load: a graphical and numeric representation of the cluster load, showing both Grid- and non-Grid (submitted locally) running processes. Red bar shows percentage of NorduGrid processes, while the grey bar shows total relative occupancy of a cluster. Numbers indicate the absolute amount of running processes, with first figure corresponding to the Grid, and second - to the non-Grid ones. It should

be noted that number of processes does not necessarily correspond to the number of running jobs: a parallel job can occupy several processors. By clicking on a bar, a user accesses the list of all Grid jobs, running on a cluster (Section 7.4).

3) Queueing: number of queueing jobs, which includes both jobs queued in an LRMS and those being pre-processed by the Grid Manager [2]. Only jobs which can be potentially executed in a Grid queue are counted. The number is linked to the same module as the Load item, with the only difference that it displays the list of the Grid-queued jobs. Note that non-Grid jobs are counted in the total number of queued jobs, while they can not be listed by the Grid Monitor, as they are not providing any information in the NorduGrid Information System.
4) Match-it-yourself: link to the "Match-it-yourself" interface (Section 7.7) which allows users to compose non-standard monitor requests.
5) Storage resources: link to the list of available storage resources (Section 7.8).
6) User base: several auxiliary links, providing an access to the VO-listing module (Section 7.9). The main purpose of this link is to provide an easy access to the user-specific information, such as the list of submitted jobs and available resources.

7.2 Cluster Description

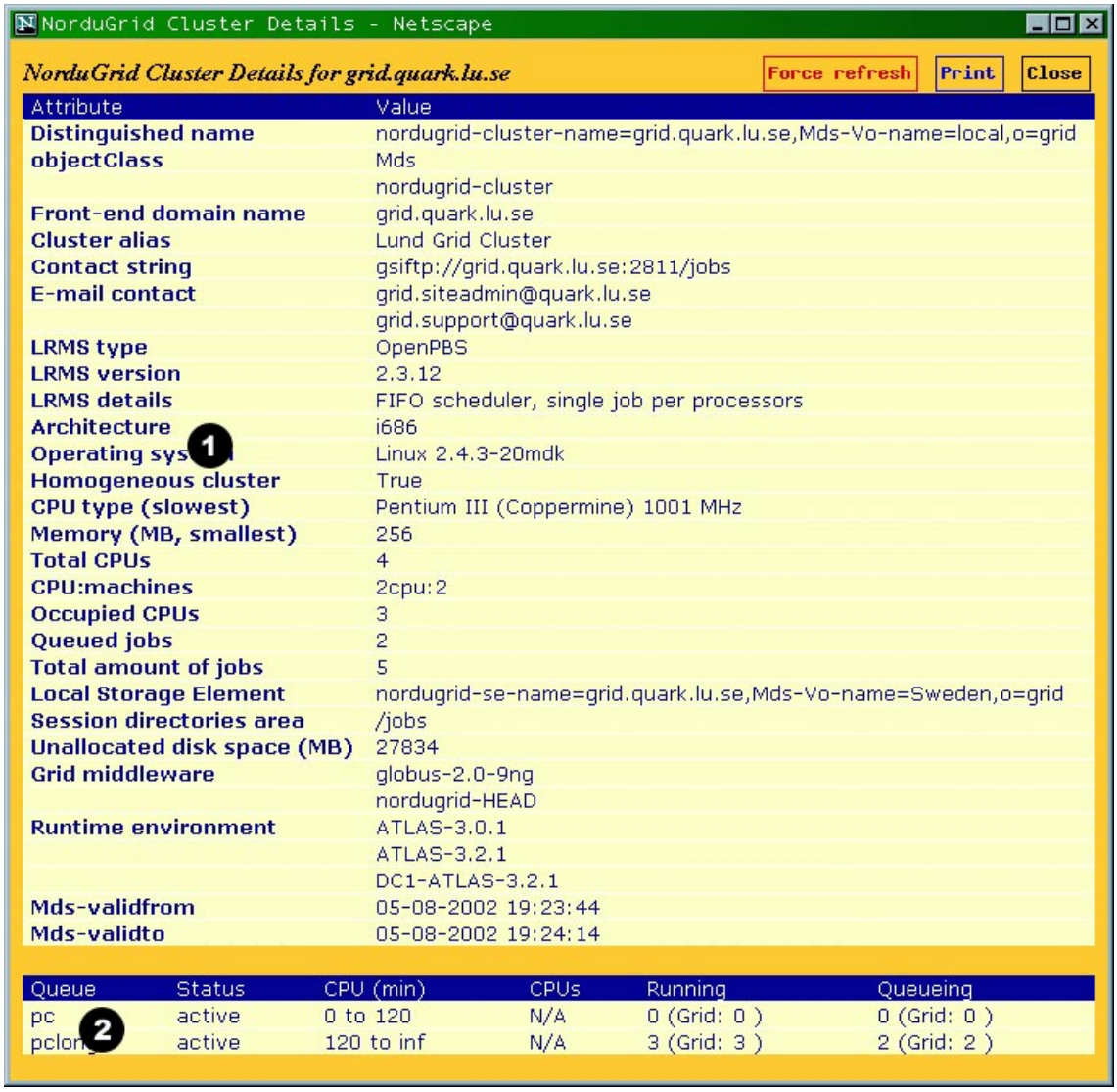


Figure 7.2: NorduGrid cluster details

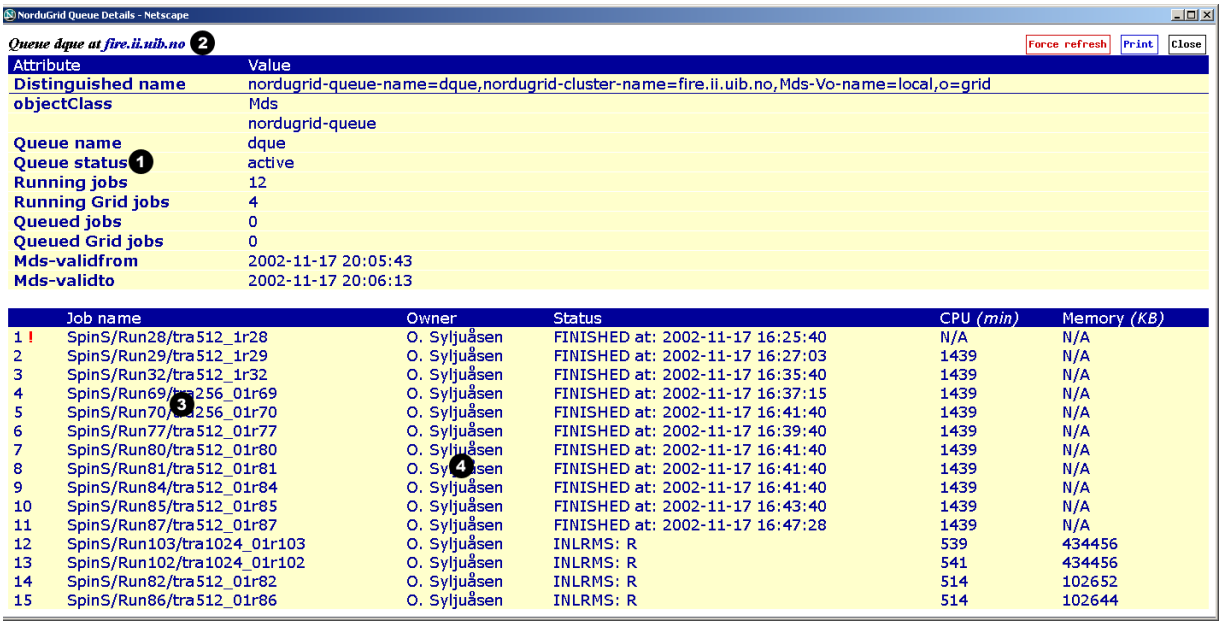
The cluster description module displays all the cluster attributes stored in the MDS, as well as most relevant information about the queues, accessible by the NorduGrid users. The window thus contains two lists, as shown in Figure 7.2:

1) Attributes: this is a dump of all the attributes of the nordugrid-cluster objectclass, dynamic and static ones. Such attributes as cluster alias, or domain name, are static; others are dynamic, with the values obtained by the MDS from the information providers: e.g., total CPU number, amount of jobs, or available disk space. More details about these attributes can be found in the NorduGrid Information System description [3]. Each attribute (apart from the MDS time stamps) is linked to the Attributes Overview module (Section 7.6), such that clicking on an attribute name brings the list of the values of this particular attribute on all the NorduGrid clusters. For instance, this is the most convenient way to browse available disk space or runtime environment values over the system.
2) Queues: the list of queues at a given cluster, accessible by the NorduGrid users. While the detailed list of queue attributes and corresponding jobs can be obtained by clicking on a queue name (see Queue Details module description, Section 7.3), the most essential parameters are listed already in the Cluster Description module. They are: queue name, queue status, queue length (minimal and maximal), number of CPUs assigned to a queue (if available), and number of running and queued jobs. Since queues can be shared between Grid and local users, the total number of jobs is shown, with the number of Grid jobs in parentheses.

The Cluster Description module is linked from most other modules (except the List of Users one): clicking on a domain name of a cluster brings the Cluster Description window.

7.3 Queue Details

In the NorduGrid Information System, the nordugrid-queue objectclass is described by a set of queue-specific attributes, and has two sub-trees: nordugrid-job and nordugrid-authuser. This structure reflects the fact that users are not implicitly authorized to submit jobs to any queue. However, the list of users allowed to a specific queue is a fairly static information, and thus is beyond the scope of the Grid Monitor*.


NorduGrid Queue Details - Netscape

Queue dque at fire.ii.uib.no 2 — Force refresh | Print | Close

Attribute	Value
Distinguished name	nordugrid-queue-name=dque,nordugrid-cluster-name=fire.ii.uib.no,Mds-Vo-name=local,o=grid
objectClass	Mds
	nordugrid-queue
Queue name	dque
Queue status 1	active
Running jobs	12
Running Grid jobs	4
Queued jobs	0
Queued Grid jobs	0
Mds-validfrom	2002-11-17 20:05:43
Mds-validto	2002-11-17 20:06:13

	Job name	Owner	Status	CPU *(min)*	Memory *(KB)*
1 !	SpinS/Run28/tra512_1r28	O. Syljuåsen	FINISHED at: 2002-11-17 16:25:40	N/A	N/A
2	SpinS/Run29/tra512_1r29	O. Syljuåsen	FINISHED at: 2002-11-17 16:27:03	1439	N/A
3	SpinS/Run32/tra512_1r32	O. Syljuåsen	FINISHED at: 2002-11-17 16:35:40	1439	N/A
4	SpinS/Run69/tra256_01r69	O. Syljuåsen	FINISHED at: 2002-11-17 16:37:15	1439	N/A
5	SpinS/Run70/tra256_01r70	O. Syljuåsen	FINISHED at: 2002-11-17 16:41:40	1439	N/A
6	SpinS/Run77/tra512_01r77	O. Syljuåsen	FINISHED at: 2002-11-17 16:39:40	1439	N/A
7	SpinS/Run80/tra512_01r80	O. Syljuåsen	FINISHED at: 2002-11-17 16:41:40	1439	N/A
8	SpinS/Run81/tra512_01r81	O. Syljuåsen	FINISHED at: 2002-11-17 16:41:40	1439	N/A
9	SpinS/Run84/tra512_01r84	O. Syljuåsen	FINISHED at: 2002-11-17 16:41:40	1439	N/A
10	SpinS/Run85/tra512_01r85	O. Syljuåsen	FINISHED at: 2002-11-17 16:43:40	1439	N/A
11	SpinS/Run87/tra512_01r87	O. Syljuåsen	FINISHED at: 2002-11-17 16:47:28	1439	N/A
12	SpinS/Run103/tra1024_01r103	O. Syljuåsen	INLRMS: R	539	434456
13	SpinS/Run102/tra1024_01r102	O. Syljuåsen	INLRMS: R	541	434456
14	SpinS/Run82/tra512_01r82	O. Syljuåsen	INLRMS: R	514	102652
15	SpinS/Run86/tra512_01r86	O. Syljuåsen	INLRMS: R	514	102644

Figure 7.3: NorduGrid queue details

The Queue Details module provides the list of the queue attributes and of all the jobs scheduled (running or waiting) to this queue. Figure 7.3 shows the NorduGrid queue description window, with clickable fields marked by numbered tags as follows:

*List of queues available for a given user can be obtained through the User Information module.

1) Attributes: the dump of the queue attributes. Just like the cluster attributes (Section 7.2), they can be both static and dynamic. Every attribute is linked to the Attributes Overview module (Section 7.6), which allows to browse the values of each attribute over all the NorduGrid system.
2) Cluster name: each queue is associated with the cluster, which name is shown at the top of the window. Clicking the cluster name brings up the Cluster Description window (Section 7.2).
3) Job name: from the Queue Details window, users can get access to detailed information about every job in the queue by clicking the job name. Each job name is linked to the Job Information module, described in Section 7.4.
4) Owner: The Grid authentication mechanism allows to associate every job with a corresponding user, even though an actual Unix account owner may be a generic "griduser". The Grid Monitor uses this feature to display explicitly each job owner. In the Queue Details window (as in all other modules), user's name is linked to the User Information module (Section 7.5), which displays all the resources available for a given user, as well as the list of user's jobs.

Queue Information module is accessible via links to queue names in the Cluster Information (Section 7.2), Job Information (Section 7.4), User Information (Section 7.5) and Attributes Overview (Sec 7.6) modules.

7.4 Job Information

The Job Information module is activated on three different occasions:

- To display a list of all running NorduGrid jobs on a cluster
- To display a list of all queued NorduGrid jobs on a cluster
- To show the full information on a given job

Lists of running and queued jobs are accessible from the top Grid Monitor window (Section 7.1) by clicking the corresponding fields (marked 2 and 3 in Figure 7.1). As shown in Figure 7.4, such a list contains not only job names, but also their respective owners, status (as returned by the Grid Manager), execution time (in case of running jobs), and the submission queue.

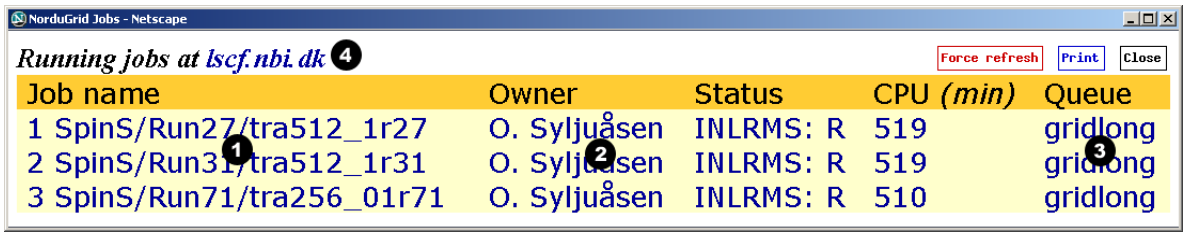


Figure 7.4: NorduGrid job list

Most of the fields in a job list window are linked to the corresponding monitor modules, giving access to more detailed information:

1) Job name: just like in the Queue Details window (Section 7.3), the job name is linked to the Job Information window, described below. However, while the Queue Details module lists the jobs in a given queue, the Job Information window gives an overview of all the NorduGrid jobs on a cluster.
2) Owner: this field is also identical to the one in the Queue Details window: user's name is linked to the User Information module (Section 7.5), which displays all the resources available for a given user and the list of user's jobs.
3) Queue: the name of the queue is liked to the Queue Details window (Section 7.3), which gives a snapshot of the queue status, including al the NorduGrid jobs submitted to a particular queue – running or waiting.
4) Cluster name: clicking on the cluster name brings up the Cluster Description window (Section 7.2), which gives a general overview of a given cluster and the status of its queues (those available for the NorduGrid users).

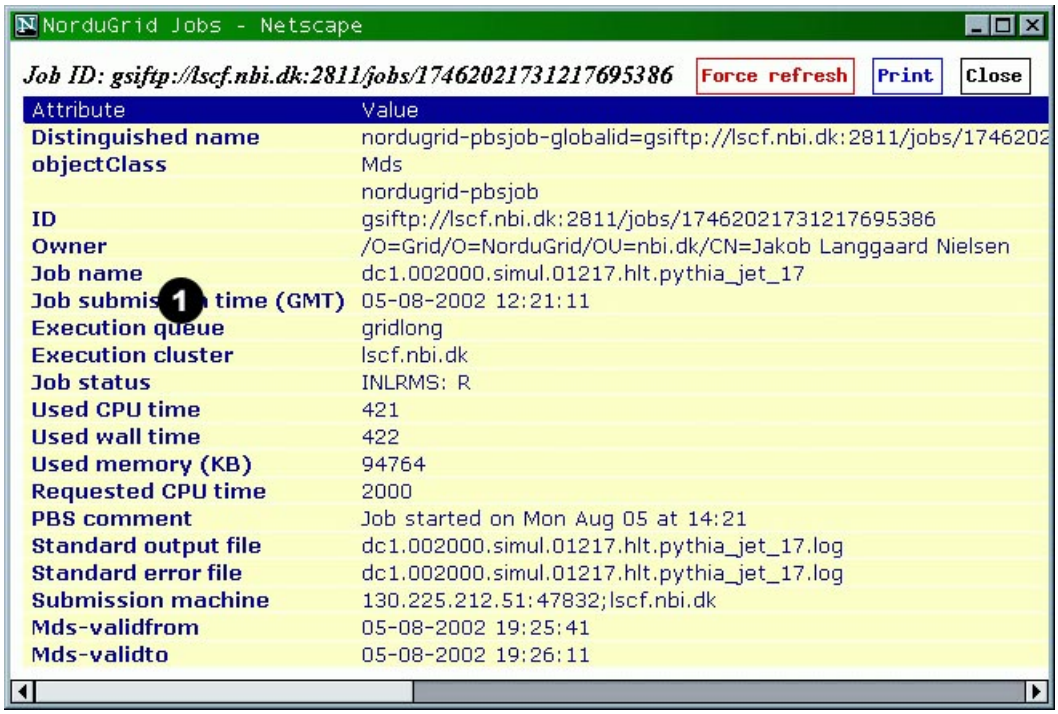


Figure 7.5: NorduGrid job statistics

The job information window is invoked by clicking on a job name in any Grid Monitor window which lists jobs. It is handled by the same module which produces running/queued job list, and contains a simple dump of all the available job attributes (see Figure 7.5). Just like in the Cluster Description and Queue Description windows, each attribute is clickable (as indicated by a tag numbered 1 in Figure 7.5), and is linked to the Attributes Overview module (Section 7.6). This is a convenient way to compare jobs that reside on the system.

7.5 User Information

The User Information module of the Grid Monitor gives access to all the available information, related to a given user. This includes the list of available resources (queues, processors and disk space), and the list of user jobs, residing on the system at the time of query. To collect this information, the whole system has to be queried, therefore invocation of this module typically takes quite a bit of time (at least comparing to most other modules).

Figure 7.6 shows a typical User Information window, where the numbered fields are linked to other Grid Monitor modules:

1) Job name: this field is linked to the Job Information window (Section 7.4), providing access to the detailed information on a given job. Unlike Job Information or Queue Information modules, which list jobs local to a cluster, the User Information module collects all the jobs submitted by a given user to the whole system.
2) Cluster: since the User Information window displays all the jobs associated with a given user, description of each respective cluster is available by clicking the cluster name. This brings up a cluster description window, described in Section 7.2.
3) Queue: this field is linked to the Queue Details module (Section 7.3), thus giving access to the information about the status of the relevant queue.
4) Cluster: the upper part of the User Information window lists the NorduGrid resources, available for a user. Each cluster, to which a user is authorized to submit jobs, is indicated by its name. Cluster names are linked to the Cluster Description window (Section 7.2), giving detailed information on available resources.

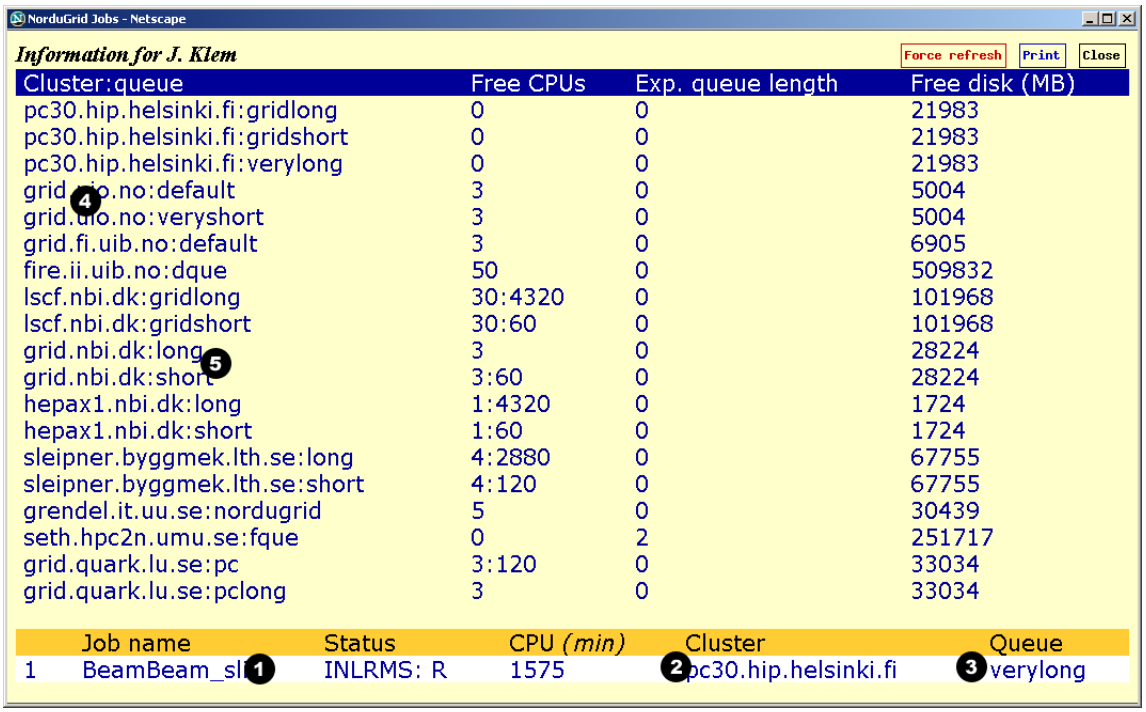


Figure 7.6: NorduGrid user information

5) Queue: since users authorization may be not only cluster-based, but also queue-based, the allowed queue information can be accessed by clicking a queue name. This brings up the Queue Details window, described in Section 7.3.

The simplest way to access the User Information window is via the List of Users (Section 7.9), although it can be invoked from any Grid Monitor window where a user name is displayed (e.g., a Job Information or a Queue Details window).

7.6 Attributes Overview

As was mentioned above, every NorduGrid objectclass attribute, appearing in a Grid Monitor window, is linked to the Attributes Overview module, which queries all the relevant objects on the system and delivers a comparative list of the attributes. Similarly to the User Information module, querying all the NorduGrid resources takes somewhat long time, as the Grid Monitor does not have an own cache.

This module can also be accessed via the "Match-it-yourself" interface (Section 7.7). In this case, it can list as many attributes as specified by a user request, eventualy applying the user selection criteria.

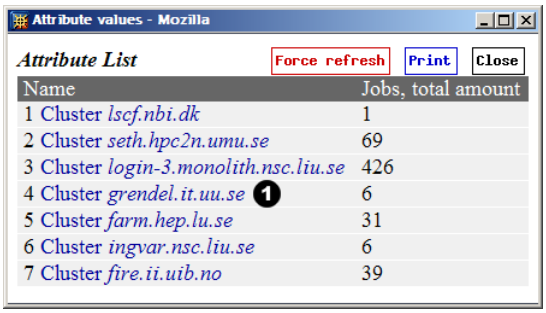


Figure 7.7: NorduGrid objects grouped by attribute

Figure 7.7 shows a typical result of the Attributes Overview query: in this example, the nordugrid-cluster attribute "Jobs, total amount" was queried, and a comparative list of results returned. The Resource field

(indicated by the tag 1) depends on the nature of the attribute, and can be either of:

- cluster name, linked to the Cluster Description module,
- cluster name and queue name, linked to the Cluster Description and Queue Details modules respectively,
- job ID string (see ref.[2] for details), linked to the Job Information module.

7.7 "Match-it-yourself"

The "Match-it-yourself" is a customizable interface to the Attributes Overview module (Section 7.6). It allows users to choose which attributes of an object to display, optionally applying filters. While the other Monitor windows display a pre-defined set of data, this module gives an advanced user a possibility to build a customized request to the Information System.

An example use case for this interface could be a user desiring to view a list of his running (but not queued or finished) jobs, complete with used CPU and wall time, memory and job name. The "Match-it-yourself" tool would be then invoked for the job object, and the display request would contain Name, Used CPU time, Used wall time, Used memory (KB), and Status – the latter with a filter Status = INLRMS: R.

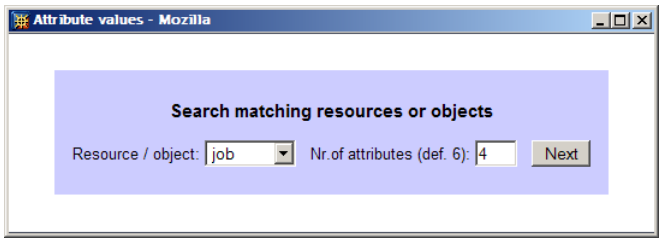


Figure 7.8: Object class selection window

Figure 7.8 shows the first screen of the "Match-it-yourself" interface, which welcomes users to select the object class to work with, and the amount of attributes to be displayed. When not sure about the latter, users should specify a top estimate – unused fields will be ignored in further searches.

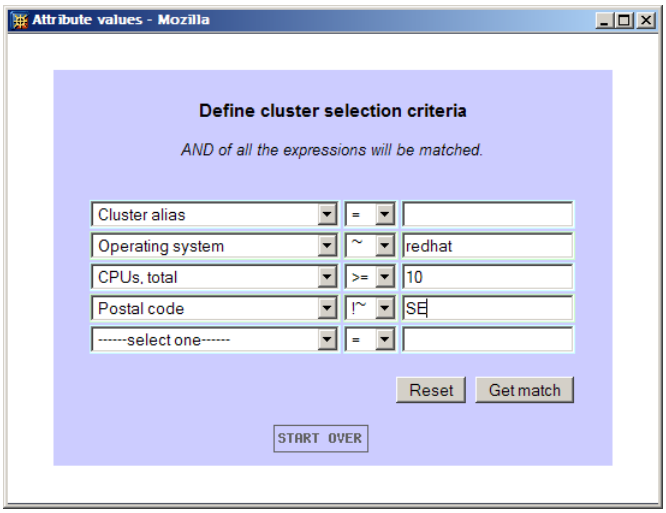


Figure 7.9: Attribute selection window

Figure 7.9 is a snapshot of the screen where the atttributes to display and their selection criteria are specified. If a user wishes to display an attribute for all the objects, independently of its value, the rightmost field may be either kept empty, or filled with an asterisk (∗), while the middle field should be set to "=". Whenever a filter has to be applied, an operator should be selected in the middle column, and a match string specified in

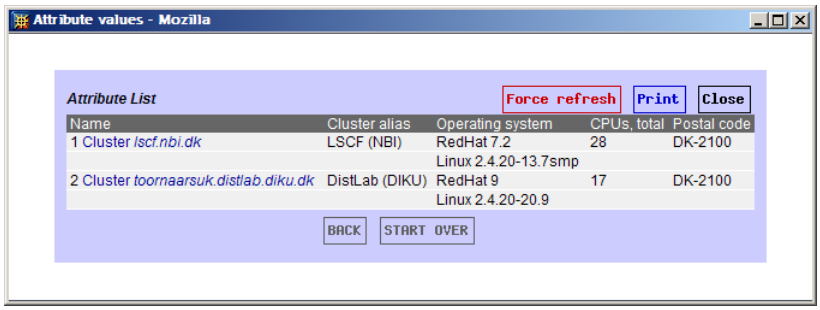


Figure 7.10: Customized cluster information display

the rightmost field. For example, if only clusters containing "NBI" in their domain names have to be shown, the attribute filter would be Front-end domain name ∼ nbi. Matches are case-insensitive.

Figure 7.10 is the result of the search according to the criteria defined in the exampel in Figure 7.9. Three filters were applied: on operating system attribute, total number of CPUs and postal code (in this case we were selecting any cluster which is not in Sweden). Since we wanted to display each cluster's alias as well, this attribute was added to the selection, but with a "match everything" scope. The attribute matching method is exactly the same as used by the Attributes Overview module (Section 7.6), and it re-uses the screen layout shown in Figure 7.7.

7.8 Storage Resources

Although there is no well-defined Storage Element concept in the NorduGrid, some information about the storage resources can be found in the Information System. The Storage Resources module, linked from the main Monitor window, displays all the available information for those Storage Elements which publish it. Particularly important is the base URL, which specifies the Grid mount point that could be used in job descriptions.

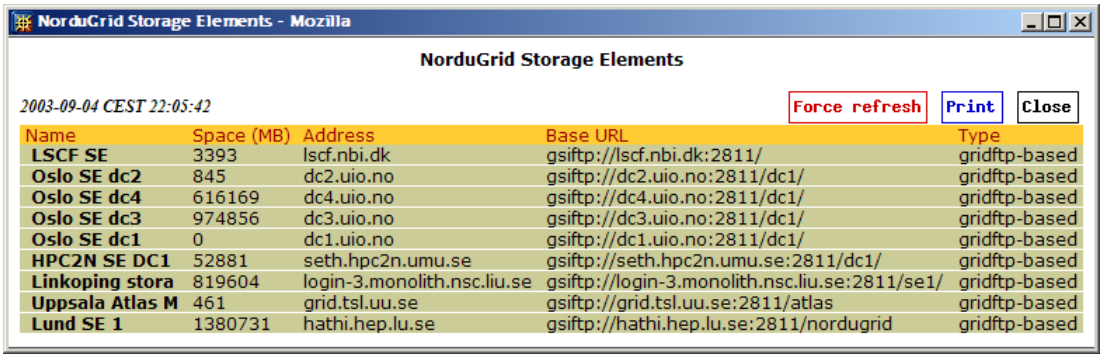


Figure 7.11: List of storage elements

7.9 List of Users

The List of Users module is different from the rest of the Grid Monitor modules because it does not deal with the NorduGrid MDS. Instead, it retrieves lists of users from the NorduGrid VO database [15]. It serves as a link between the two databases (MDS and VO), by interfacing each user record to the User Information module (Section 7.5). Figure 7.12 shows a screenshot of a typical VO user list, with numbered tags indicating clickable links as follows:

1) Name: user name as given in the corresponding Grid certificate field, linked to the User Information module (Section 7.5).

2) E-mail: E-mail address of a user, if available. It is linked to an e-mail URL, allowing to send a message to a user directly from the browser (if such an option is enabled in a browser).

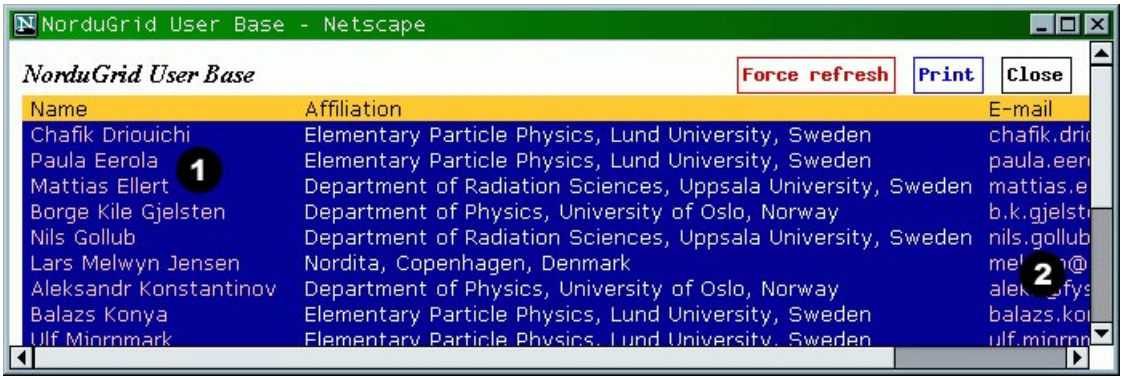


Figure 7.12: List of the NorduGrid users

The List of Users is available only from the top Grid Monitor window.

Chapter 8

HOWTO

UNDER CONSTRUCTION

Can I use different UIs simultaneously?

How do I know when the job will be removed?

What are the times in the notification e-mail?

How to request a certificate in a different location?

How to create a proxy in a different place?

How to request a certificate from another CA?

How to generate a proxy with a different lifetime?

How to check how long the proxy is valid?

How to renew a certificate?

How to use different certificates?

How to list all the jobs submitted by a user?
Click a user name in any window, e.g., in the List of Users (Figure 7.12, item 1). The List of Users is available from the top Grid Monitor window (Figure 7.1, item 4).

Why the jobs finished two days ago do not show up?
Job results are being kept in the NorduGrid session directory only for a limited period, typically 24 hours. After a session directory is erased, – either by a user request or after its lifetime expiration – all the job information disappears from the system.

How to list all jobs on the NorduGrid? Bring up any window containing list of jobs (e.g., click a "Load" bar in the top Grid Monitor window (Figure 7.1, item 2); then click on a job name to bring up a job information window (Figure 7.5); then click any job attribute, like job name or job owner. This will activate the attributes list window (Figure 7.7), containing the list of all the jobs.

How to list all jobs running on a cluster?
Click a "Load" bar in the top Grid Monitor window (Figure 7.1, item 2).

How to list all jobs in a queue?
Click a queue name in any window, e.g., the cluster information (Figure 7.2, item 2), the job information (Figure 7.4, item 3), or the user information (Figure 7.6, item 5).

How to list available runtime environments?
Click any "Cluster" field in the top Grid Monitor window (Figure 7.1, item 1) to bring up the Cluster Details window. Then click "Runtime environment" link in the "Attribute" column (Figure 7.2, item 1).

Appendix A

RPM For Everybody

You've been told that working with RPMs needs system administrator privileges? You've been misled.

RPM stands for *"RedHat Package Manager"* [16] and is a sophisticated archiving format, allowing not only to pack a set of files and directories in a single archive, but also to install them in an easy-to-manage manner. While a Tape Archive (`.tar`) or a ZIP archive, with which you may be familiar with, simply packs files with their relative paths, RPM stores entire directory information starting from the root. RPM also relies on a system-wide database to register software version numbers, dependencies and other relevant data. Naturally, directory structures are different on different systems, and not every user has proper permissions to modify them. Also, not everybody is allowed to write into the main RPM database. Evidently, RPM format was created having system administrators in mind. Nevertheless, even if you are a regular user with no special privileges, you can make a fair use of the RPM packages, especially so if creators of such packages made them *relocatable*, i.e., user-friendly. Luckily, most NorduGrid packages are relocatable, including the distributed Globus Toolkit® RPMs. Below are some quick instructions on how to work with RPM packages without having system administrator privileges.

A.1 Listing Contents of an RPM Package

To list files archived in an RPM package, issue an RPM query command:

```
rpm -qlp myfile.rpm
```

Here command line parameters have the following meaning:

- **q** – perform a query
- **l** – list files
- **p** – use the specified package file (`myfile.rpm` in this example)

A.2 Printing Out an PM Package Information

To retrieve general information about a given RPM package, issue the following RPM query:

```
rpm -qip myfile.rpm
```

Here command line parameters have the following meaning:

- **q** – perform a query
- **i** – print out package information
- **p** – use the specified package file (`myfile.rpm` in this example)

This query will print out all the essential information stored by the creator of the RPM package in question. This may contain such fields as package name and version, creator's name, build date and – what's most important for users – ***relocations***. Section A.7 explains how to make use of this information.

A.3 Simple Unpacking of an RPM Archive

In many cases you would only need to extract files from an RPM archive, without performing complex operations with an RPM database. There is a simple way to do it:

```
rpm2cpio < myfile.rpm | cpio -i --make-directories
```

This method makes use of the generic GNU CPIO utility, which deals with copying files t and from archives. The `rpm2cpio` command converts from RPM to CPIO format, and the output of this conversion is redirected to the `cpio` command. Command line parameters for the latter have the following meaning:

i	–	extract contents
make-directories	–	create directories when necessary

As a result, all the contents of the RPM package will be extracted into the current directory, creating relative paths.

A.4 Creating a Private RPM Database

To achieve higher flexibility and to get access to more RPM functionality, you may want to take a step forward and create your own RPM database. You only have to do it once, by issuing the following instructions:

```
mkdir $HOME/myrpmdb
rpm --initdb --dbpath $HOME/myrpmdb
```

Mentioned in this example directory `$HOME/myrpmdb` can of course be substituted with any other location which suits you best. The only requirement is that you must have write permissions in this location.

Having such a personal RPM database allows you to perform more complex operations with RPM packages, without resorting to simple unpacking and manual installation. To simplify your life even further, do the following:

```
echo '%_dbpath /home/yourname/myrpmdb' | cat >> $HOME/.rpmmacros
```

This will create a file `.rpmmacros` in your `$HOME` directory*, and add there the line specifying the database location. In this example, `/home/yourname/myrpmdb` should be substituted with your RPM database path. This will instruct RPM to use always your personal database. If you don't want to use this feature (for example, if you have several databases), take care to append

```
--dbpath $HOME/myrpmdb
```

to each RPM instruction which refers to such a database (e.g., package installation or removal) .

It may be useful to copy the system-wide database into your own one. RPM does not provide any tool for database synchronization (yet?), thus you have to do it hard way:

```
cp /var/lib/rpm/* $HOME/myrpmdb/.
```

Here `/var/lib/rpm/` is a typical default location of the system-wide database (it could be different though on different systems). Having thus your private database initialized with the existing system information may help you in dealing with *dependencies* (see Section A.8), but only for a while, since any system-wide upgrades will not be registered in your database, and there is no way to synchronize RPM databases in a clever manner.

*On some systems, you should use `>` instead of `>>` if the file `.rpmmmacros` did not exist before.

A.5 Installing an RPM Package

Having defined your private database (Section A.4), you can attempt to install any RPM package by doing the following:

```
rpm -ivh myfile.rpm
```

Here command line parameters have the following meaning:

- **i** – perform an installation procedure
- **v** – print extra information
- **h** – print progress bar with hash-marks

Keep in mind that if you didn't define a default database in `$HOME/.rpmmacros` , you'll have to append `--dbpath $HOME/myrpmdb` (or whatever is your private database location) to the directive.

Most likely, however, such a simple installation instruction will fail. There are two main reasons:

a) RPM packages attempt to install themselves in locations defined by their creators, not by you. How to get around, read in Section A.7.
b) RPM checks for other software needed by the package – *dependencies* – in your private database; however, most likely, such software was not installed by you and can not be found in your database. How to deal with it, read in Section A.8.

A.6 Upgrading RPM Packages

If you happened to have a package installed and simply want to upgrade it, issue the following instruction:

```
rpm -Uvh myfile.rpm
```

Here command line parameters have the following meaning:

- **U** – upgrade the installation
- **v** – print extra information
- **h** – print progress bar with hash-marks

Since this operation needs to access a database, you either have to have the default database location specified in the `$HOME/.rpmmacros` file, or to append `--dbpath $HOME/myrpmdb` (or whatever is your private database location) to the directive (see Section A.4 for details).

A.7 Relocating RPM Packages

As it was mentioned in Section A.2, an RPM package may contain information on relocatable directories. Such directories are listed as *relocations*, meaning that their contents can be relocated during the installation in another directory. To perform such an installation, do the following:

```
rpm -ivh myfile.rpm --relocate /oldlocation=/newlocation
```

Here `/oldlocation` is one of the relocatable directories of the original package as listed by the `rpm -qip myfile.rpm` command, and `/newlocation` is your preferred destination, specified as an absolute path (starting with "`/`"). The command line parameters have the following meaning:

- **i** – perform an installation procedure
- **v** – print extra information
- **h** – print progress bar with hash-marks
- **relocate** – instruct RPM to install files from `/oldlocation` into `/newlocation`

As of RPM version 4.2.1, multiple relocations are possible[†]. To use this option, specify several relocation pairs on the command line, i.e.,

```
--relocate /old=/new --relocate /old/subdir=/new/subdir.
```

Order of such pairs in the line is important, as RPM will attempt to create new directories and, quite naturally, can not skip a level.

If the package information (as listed by `rpm -qip`) contains several relocations, you must specify a `--relocate` pair for **a**ll of them.

It may turn out that the package creator forgot to describe some pathes as "relocateable", so that they do not appear in the relocations list of the RPM, and yet the package attempt to install in such directory. There's little you can do about it, except attempting to use the `--badreloc` option:

```
rpm -ivh myfile.rpm --relocate /oldlocation=/newlocation --badreloc
```

As in the case of installation, you either have to have the default database location specified in the `$HOME/.rpmmacros` or to append `--dbpath $HOME/myrpmdb` private database location) to the directive (see Section A.4 for details).

A.8 Dealing with Dependencies

Even when relocations proceeded smoothly, your package may still fail to get installed due to unsatisfied *dependencies* (this is the name for any other software needed by the package), which are absent from your private database (Section A.4). A bold way to deal with it is to force the installation, e.g.:

```
rpm -ivh myfile.rpm --relocate /oldlocation=/newlocation --nodeps
```

The command line parameters have the following meaning:

i	–	perform an installation procedure
v	–	print extra information
h	–	print progress bar with hash-marks
relocate	–	instruct RPM to install files from `/oldlocation` into `/newlocation`
nodeps	–	ignore dependencies

This is an admittedly non-elegant way. A better solution may be to enter the necessary information into the database by installing "virtual" RPMs which only register a database record without actually installing software; however this is clearly package-dependent, thus you should request such a "fake" RPM from the package provider.

A.9 Listing Installed Packages

To list all the packages installed via an RPM mecanisms, do:

```
rpm -qai
```

Here command line parameters have the following meaning:

q	–	perform a query
a	–	query all installed packages
i	–	print out package information

[†]Some older versions can also do multiple relocations, but most have one or another bug.

This command prints out the information stored in your RPM database, hence you either have to have the default database location specified in the `$HOME/.rpmmacros` file, or to append `--dbpath $HOME/myrpmdb` (or whatever is your private database location) to the directive (see Section A.4 for details).

A.10 Uninstalling RPM Packages

Whenever you'll have to uninstall a package, issue the instruction:

```
rpm -e mypackage
```

Here `mypackage` is the name of the package as listed by the `rpm -qai` command. Command line parameter has the following meaning:

e – uninstall packages

This command removes the files associated with the package from your system and erases the corresponding record from your RPM database. You either have to have the default database location specified in the `$HOME/.rpmmacros` file, or to append `--dbpath $HOME/myrpmdb` (or whatever is your private database location) to the directive (see Section A.4 for details).

A.11 Building RPM from Source

It may happen so that binary RPM packages for your system are unavailable, but a source RPM or a tarball is provided. You can build a binary RPM yourself from such sources, but some extra manipulations should be made first.

The default RPM top directory is `/usr/src/redhat` (for a RedHat system). Naturally, as a user, you have no write permission there. To solve the problem, start by creating a new directory structure in any place which has enough disk space and write permission:

```
mkdir -p rpmtop/RPMS/i386; mkdir rpmtop/SRPMS; mkdir rpmtop/SOURCES;
mkdir rpmtop/BUILD; mkdir rpmtop/SPEC
```

Now you should instruct RPM to use this directory instead of the system-wide one (assuming you created the `rpmtop` structure above in `/home/yourname/`):

```
echo '%_topdir /home/yourname/rpmtop' | cat >> $HOME/.rpmmacros
```

Next thing you should take care of, is the temporary directory for RPM builds, which by default is `/var/tmp/`, also unavailable for mere mortals. The solution is similar to the above, to create your own directory in any place which has enough disk space and write permission:

```
mkdir mytmp
```

This directory should also be specified in your `.rpmmacros` file:

```
echo '%_tmppath /home/yourname/mytmp' | cat >> $HOME/.rpmmacros
```

That's it; from now on you should be able to build binary RPMs from source by using the command:

```
rpmbuild -ba mypackage.src.rpm
```

or

```
rpmbuild -ta mypackage.tar.gz
```

depending whether the sources are provided as source RPM (`mypackage.src.rpm`) or as a tarball (`mypackage.tar.gz`). The command line options are, correspondingly:

ba – build binary and source packages from a source RPM

ta – build binary and source packages from (gzipped) tarball

Your RPMs will be placed in `/home/yourname/rpmtop/RPMS/i386/` directory of the example above. In case your architecture is different from `i386`, please take care of creating a corresponding subdirectory in `rpmtop/RPMS` (e.g., `rpmtop/RPMS/alpha`). If you do not know what is the architecture, use

```
uname -m
```

This command will typically print out a string like “i386” or “i586” or “i686” and such, which is the hardware architecture name of your machine.

Appendix B

Known Grid Certificate Authorities

This list of Certificate Authorities is non-authoritative, as the authorities and links use to change.

Country	**CA**	**URL**
Canada	GridCanada	http://www.gridcanada.ca/ca
Czech Republic	CESNET	http://www.cesnet.cz/pki
Cyprus	CyGrid	http://www.cs.ucy.ac.cy/cygrid-ca
Denmark	NorduGrid	http://www.nbi.dk/HEP/CA
Finland	NorduGrid	http://www.nbi.dk/HEP/CA
France	CNRS	http://igc.services.cnrs.fr/Datagrid-fr
Germany	GermanGrid	http://grid.fzk.de/ca
Greece	Hellas Grid	http://pki.physics.auth.gr/hellasgrid-ca
Ireland, Republic of	Grid-Ireland	http://www.cs.tcd.ie/grid-ireland/gi-ca
Italy	INFN	http://security.fi.infn.it/CA
Korea, Republic Of	KISTI	http://gridtest.gridcenter.or.kr
The Netherlands	NIKHEF	http://certificate.nikhef.nl
Norway	NorduGrid	http://www.nbi.dk/HEP/CA
Poland	Polish Grid	http://www.man.poznan.pl/plgrid-ca
Portugal	LIP	http://www.lip.pt/ca
Russia	MSU	http://lhc.sinp.msu.ru/CA
Slovakia	SlovakGrid	http://ups.savba.sk/ca
Spain	IFCA	http://www.ifca.unican.es/datagrid/ca
Sweden	NorduGrid	http://www.nbi.dk/HEP/CA
Switzerland	CERN	http://cern.ch/globus/ca
Taipei	ASGC	http://ca.grid.sinica.edu.tw
United Kingdom	e-Science Grid	http://www.grid-support.ac.uk/ca
	GridPP	http://www.gridpp.ac.uk/ca
United States	DOE Science Grid	http://www.doegrids.org
	ESnet	http://www.es.net/CA
	FNAL	http://computing.fnal.gov/security/pki

Bibliography

[1] The Globus Project. [Online]. Available: http://www.globus.org

[2] A. Konstantinov. The NorduGrid Grid Manager. [Online]. Available: http://www.nordugrid.org/documents/GM.pdf

[3] B. Kónya. The NorduGrid Information System. [Online]. Available: http://www.nordugrid.org/documents/ng-infosys.pdf

[4] M. Ellert. The NorduGrid toolkit user interface. [Online]. Available: http://www.nordugrid.org/documents/NorduGrid-UI.pdf

[5] O. Smirnova. Extended Resource Specification Language. [Online]. Available: http://www.nordugrid.org/documents/xrsl.pdf

[6] ——. The Grid Monitor. [Online]. Available: http://www.nordugrid.org/documents/monitor.pdf

[7] R. Labs. RSA Cryptography FAQ. [Online]. Available: http://www.nordugrid.org/documents/rsalabs˙faq41.pdf

[8] Public-Key Infrastructure (X.509) (pkix). [Online]. Available: http://www.ietf.org/html.charters/pkix-charter.html

[9] The Open Source toolkit for SSL/TLS. [Online]. Available: http://www.openssl.org/

[10] Open source implementation of the Lightweight Directory Access Protocol. [Online]. Available: http://www.openldap.org

[11] Grid Security Infrastructure (GSI). [Online]. Available: http://www.globus.org/security/

[12] The Globus Resource Specification Language RSL v1.0.", url =.

[13] Globus replica catalog service. [Online]. Available: http://www-fp.globus.org/datagrid/replica-catalog.html

[14] Replica location service. [Online]. Available: http://grid-data-management.web.cern.ch/grid-data-management/replica-location-service/index.html

[15] Description of the NorduGrid Virtual Organisation. [Online]. Available: http://www.nordugrid.org/NorduGridVO/vo-description.html

[16] RedHat Package Manager. [Online]. Available: http://www.rpm.org

Index