



CONTRACT NUMBER 508830

DEISA
**DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR
SUPERCOMPUTING APPLICATIONS**

European Community Sixth Framework Programme
RESEARCH INFRASTRUCTURES
Integrated Infrastructure Initiative

Extension of User Tools on DESHL 3.0
Installation/User Manual for DESHL 3.0

Deliverable ID: DEISA-JRA7-4.2

Due date: April 30th 2006

Actual delivery date: May 17, 2006

Lead contractor for this deliverable: EPCC, UK

Project start date: May 1st, 2004

Duration: 4 years

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Content

Table of Content.....	1
1. Introduction.....	2
1.1 Executive Summary.....	2
1.2 References and Applicable Documents.....	2
1.3 Document Amendment Procedure.....	3
1.4 List of Acronyms and Abbreviations.....	3
2. DESHL Overview.....	4
3. Prerequisites / certificates.....	7
3.1 Required Infrastructure.....	7
3.2 Certificates.....	7
4. Installation.....	8
4.1 Unpack the bundle.....	8
4.1.1 UNIX unpacking.....	8
4.1.2 Windows unpacking.....	8
4.2 Edit the DESHL command script.....	8
4.3 Edit the DESHL Sites configuration file.....	9
4.4 Edit the logging configuration file.....	10
4.5 Verify the settings.....	11
5. Data Staging.....	12
5.1 File naming conventions.....	12
5.2 Listing available storages.....	12
5.3 Copy.....	12
5.4 Exists.....	13
5.5 isDir.....	13
5.6 isFile.....	13
5.7 list.....	14
5.8 makeDir.....	14
5.9 move.....	14
5.10 remove.....	14
6. Job submission and management.....	16
6.1 Submitting a job.....	16
6.2 status.....	18
6.3 terminate.....	19
6.4 fetch.....	19
6.5 jobs.....	19
7. Tutorial.....	21
7.1 Upload, submit and manage a simple job.....	21
7.2 Staging in data for a job, staging out results.....	23
8. Trouble shooting guide.....	25
8.1 File Permissions.....	25
8.2 Script paths.....	25
8.3 Reporting bugs / requesting features.....	26

1. Introduction

1.1 *Executive Summary*

This document is intended as an installation guide and user manual for the DESHL v3.0 client for accessing DEISA heterogeneous resources. The document describes how to install and configure the client, and describes how to perform common operations such as job submission and file movement.

Chapter 2 gives a brief overview of the DESHL and its position within the encompassing DEISA project.

Chapter 3 lists the system requirements, components and certificates that are required for successful operation of the DESHL client.

Chapter 4 describes the installation and configuration steps. These steps **MUST** be followed to successfully install and run the DESHL client.

Chapter 5 describes how the DESHL client may be used to import files to, export files from and move files within the DEISA environment.

Chapter 6 describes how the DESHL client may be used to submit and manage jobs.

Chapter 7 contains a tutorial showing how to perform typical routine operations using the DESHL client.

Chapter 8 contains a trouble-shooting guide which aims to illustrate likely problems with installation and usage.

1.2 *References and Applicable Documents*

Where indicated, diagrams use UML notation [6]

- [1] Open Group specification for Batch Environment Services, http://www.opengroup.org/onlinepubs/009695399/utilities/xcu_chap03.html#tag_03
- [2] JRA7 Quality Plan v2.0", DEISA JRA7 Report, Deliverable ID DEISA-JRA7-1.2, May 3rd 2005
- [3] GGF SAGA Research Group, <https://forge.gridforum.org/projects/saga-rg/>
- [4] GGF JSDL Working Group, <https://forge.gridforum.org/projects/jsdl-wg/>
- [5] Apache Log4j logging, <http://logging.apache.org/log4j/docs/>
- [6] Unified Modelling Language <http://www.uml.org/>
- [7] Access to the DEISA Supercomputing Grid Infrastructure http://www.deisa.org/userscorner/primer/access_to_grid.php
- [8] Job Submission through UNICORE http://www.deisa.org/userscorner/primer/jobs_submission.php
- [9] Certificates http://www.deisa.org/userscorner/primer/access_to_grid.php#2.3

1.3 Document Amendment Procedure

The document procedure is covered in the Quality Plan [2] (Section 6.6).

1.4 List of Acronyms and Abbreviations

API	Application Programming Interface
CLT	Command Line Tool
DEISA	Distributed European Infrastructure for Supercomputing Applications
DER	Distinguished Encoding Rules [X.690]
DESHL	DEISA Services for the Heterogeneous management Layer
DRM	Distributed Resource Manager
GGF	Global Grid Forum
HPC	High Performance Computing
JRA	DEISA Joint Research Activity
JRA7	Seventh Joint Research Activity
JSDL	Job Submission Description Language
OGSA	Open Grid Services Architecture
NJS	UNICORE Network Job Supervisor
PFX	Personal information eXchange
PKCS	Public Key Cryptography Standards
PKCS#12	PFX standard for file format storing private keys with accompanying public key certificates protected with a pass phrase
PKI	Public Key Infrastructure
SAGA	Simple API for Grid Applications
SSL	Secure Sockets Layer
UML	Unified Modelling Language
UNICORE	Uniform Interface to Computing Resources
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
X.509 PKI	Specification for PKI certificates for the Internet, currently v3

2. DESHL Overview

DEISA is a consortium of leading national supercomputing centres that operates a persistent, production quality, distributed supercomputing environment with continental scope. The purpose of DEISA is to enable scientific discovery across a broad spectrum of science and technology, by enhancing and reinforcing European capabilities in the area of high performance computing. This is made possible through a deep integration of existing national high-end platforms, tightly coupled by a dedicated network and supported by innovative system and grid software.

Within DEISA, DESHL v3.0 provides OGSA standards-based access for users and their applications to manage jobs and transfer files in the DEISA heterogeneous supercomputing infrastructure. This is provided by a command line client application which supports SAGA standards for remote job and file management.

DESHL v3.0 comprises:

- command line tool (CLT) based on the Open Group Batch Environment Services specification [1];
- client library exposing an API (Application Programming Interface) based on the SAGA [3] standard currently being developed by a GGF (Global Grid Forum) Research Group;
- Grid Access Library for interacting with a UNICORE Grid.

Figure 1 indicates how users or their applications employ the DESHL v3.0 command line tool to access the DEISA heterogeneous supercomputing infrastructure.

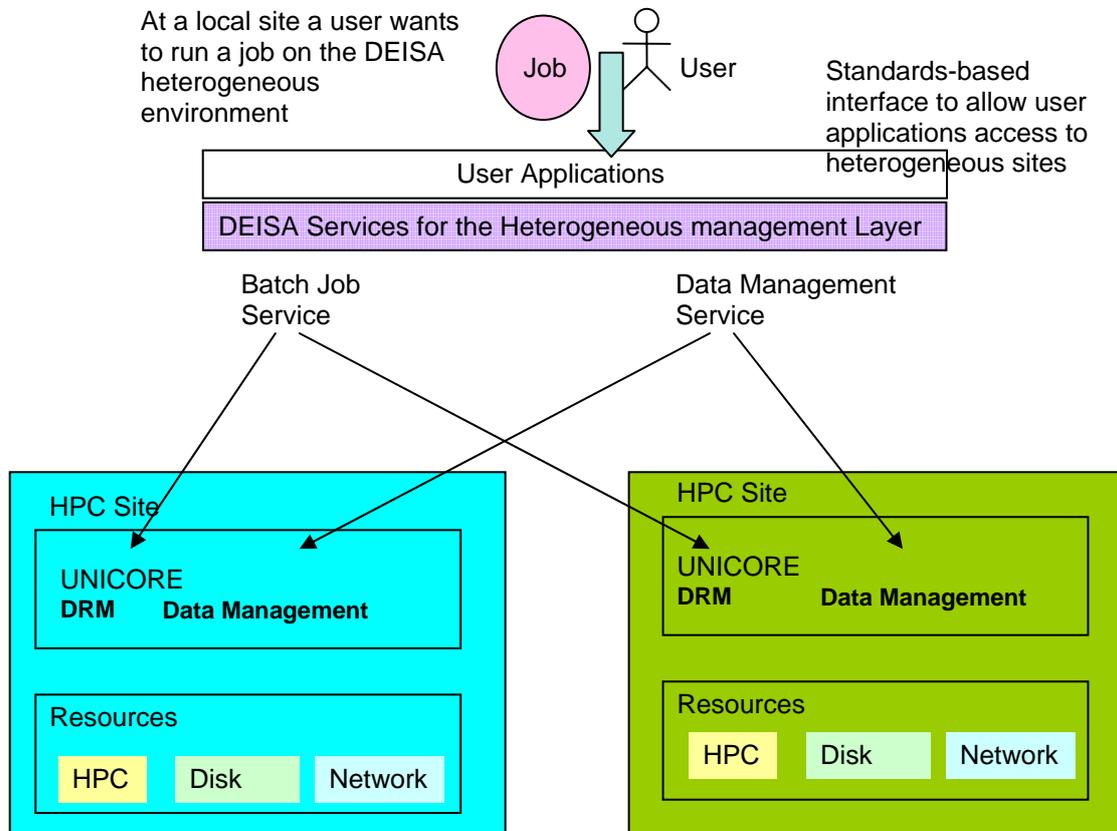


Figure 1: Using the DESHL to access the DEISA heterogeneous supercomputing infrastructure

The DEISA heterogeneous supercomputing infrastructure is organised as a UNICORE Grid ([7], [8]). UNICORE deals with the heterogeneity of the underlying supercomputing distributed resource managers and operating systems.

As shown in Figure 2, the DESHL Command Line Tool interacts directly with the DESHL client library (the SAGA-based API), which in turn utilises the Grid Access library to access a UNICORE Grid. The GGF JSDL (Job Submission Description Language) [4] standard is used to pass batch job specifications between the client library and the Grid Access Library and then on to the underlying UNICORE Grid.

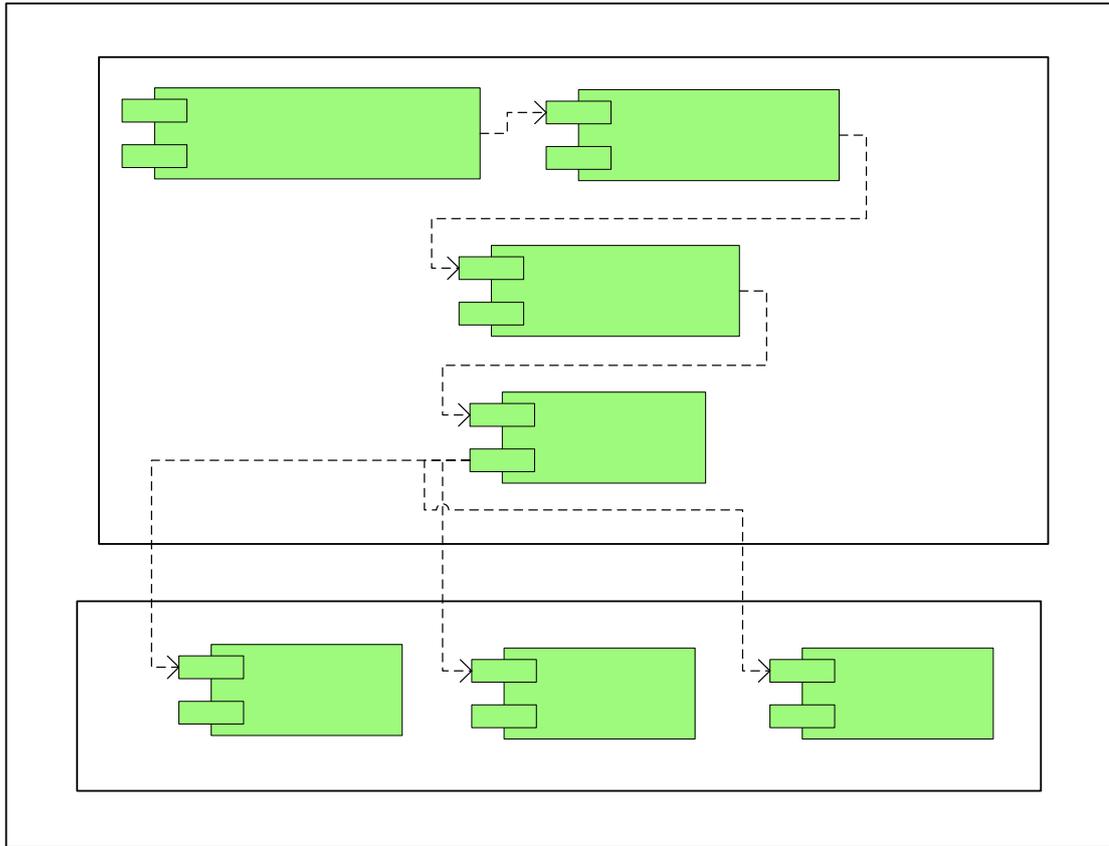


Figure 2: A UML component diagram showing the DESHL components.

The DESHL v3.0 command line tool provides the following job management and file transfer capabilities.

- determine the DEISA sites to which a user can submit a batch job to
- submit a batch job to a DEISA site
- terminate a batch job at a DEISA site
- view the status of a batch job on a DEISA site
- upload a file to a DEISA site
- download a file from a DEISA site
- delete a file from a DEISA site
- determine if a file exists on a DEISA site
- list the contents of a directory on a DEISA site
- rename a file on a DEISA site
- copy/move a file between DEISA sites

The DESHL Client Library contains those SAGA operations necessary to support the file transfer and job management capabilities provided in the DESHL command line tool.

The DESHL client is installed on a user's workstation or PC, and consists of the DESHL Command Line Tool, the DESHL Client library, the Grid Access library and the ARCON client library.

3. Prerequisites / certificates

3.1 Required Infrastructure

The DESHL v3.0 client has been tested on UNIX systems and on Windows XP. Its principal requirement is access to a correctly configured installation of Java 5.0.

This guide does not cover any details of server configuration. If you have concerns about the configuration or operation of a particular site, please contact the appropriate DEISA user support team at this site.

3.2 Certificates

A certificate compliant to the EU Grid PMA minimal requirements [9] is required to use the DESHL. Details of how to obtain this certificate vary from country to country, therefore please contact the appropriate certificate issuing authority in your region for details on how to apply for a certificate. For each site that the user wishes to access, the DESHL client requires both the user's certificate and the issuing authority's root certificate to be made available in a directory which is accessible to the DESHL client at runtime. Both certificates must be X.509 (v3) compliant (DER encoded as base64 with additional header and footer lines). For the remainder of this document it is assumed that a user's certificate is available in this format and that where a password is mentioned, the password is for the user's certificate.

4. Installation

The following steps are required to install and configure the DESHL v3.0 client:

1. **Unpack the installation bundle**
2. **Edit the DESHL command script**
3. **Edit the DESHL sites configuration file**
4. **Edit the logging configuration file**
5. **Verify the settings**

These steps are explained in more detail below.

4.1 *Unpack the bundle*

The DESHL client is supplied as a zipped tar archive file, DESHL-v3.0.tar.gz. The file can be downloaded from <http://forge.nesc.ac.uk/projects/deisa-jra7>

The archive file contains documentation including this manual, and a zipped tar file containing the DESHL client itself, deshl30.tar.gz.

4.1.1 *UNIX unpacking*

```
$ gunzip DESHL-v3.0.tar.gz
$ tar xvf DESHL-v3.0.tar
$ gunzip deshl30.tar.gz
$ tar xvf deshl30.tar
```

4.1.2 *Windows unpacking*

- 1) Right-click on DESHL-v3.0.tar.gz in a Windows Explorer window, and save all of the contents into a directory.
- 2) Right-click on deshl30.tar.gz to save the DESHL client into the directory from which it will be run.

4.2 *Edit the DESHL command script*

DESHL commands are executed by running either a command script, `deshl`, on UNIX systems or a Windows batch file, `deshl.bat`, on Windows XP systems. In both cases, this script requires editing by the user before the DESHL client can be run.

The following values need to be edited in the DESHL command file (UNIX paths shown for illustration)

`JAVA_EXE` - this must be set to the location of your Java installation, e.g.

```
JAVA_EXE=/home/javag/jdk/jdk1.5.0_03/bin/java
```

`LOG4J_CONFIG_FILE` - this must be set to the location of your logging properties file (see 4.4). (Note: this must be specified as a valid URI). Normally this should point to the `clt-deshl-log4j.properties` file in the directory in which you installed the DESHL client, e.g.

```
LOG4J_CONFIG_FILE=file:///home/deshl/clt-deshl-log4j.properties
```

`SITE_CONFIG_FILE` – this must be set to the location of your DESHL sites configuration file (see 4.3), e.g.

```
SITE_CONFIG_FILE=/home/deshl/config.csv
```

`DESHL_CLIENT_INSTALL_DIR` – this must be set to the location of directory into which you installed the DESHL client, e.g.

```
DESHL_CLIENT_INSTALL_DIR=/home/deshl
```

Finally, UNIX users should change the permissions of the deshl command script to ensure that it is executable

```
$ chmod 700 deshl
```

4.3 Edit the DESHL Sites configuration file

The DESHL client requires a sites configuration file, which contains the list of DEISA sites which the user wishes to access for file movement and job management. The file must be in comma-separated-variable format (such as that produced by Microsoft Excel), with a separate row for each DEISA site. An example configuration file is included in the installation, `config.csv`. Each row in the configuration file MUST contain a full URL for the site's NJS and the path to the user's certificate for accessing that site. (For convenience a `certs` directory is created when the archive file is unpacked. However, it is not compulsory to use this directory). The example file contains settings for accessing DEISA using HPCx as the user's home site, and must be edited to match the user's default DESIA home site if different from HPCx. An example of the contents of this file is shown in Table 1.

<i>Sitename</i>	<i>Certificate</i>	<i>Password</i>	<i>Shortcut</i>
ssl://admin.hpcx.ac.uk:4433/HPCx	/home/deshl/mycert.p12	xxx	hpcx
ssl://admin.hpcx.ac.uk:4433/CSC	/home/deshl/mycert.p12		csc
ssl://admin.hpcx.ac.uk:4433/IDRIS%20ZAHIR	/home/deshl/mycert.p12		idris
ssl://admin.hpcx.ac.uk:4433/FZJ%20JUMP	/home/deshl/mycert.p12		fzj
ssl://admin.hpcx.ac.uk:4433/RZG%20SP4	/home/deshl/mycert.p12		rzg

Table 1: Example site configuration file

Each site must be specified as `ssl://<gatewayName>:<port_no>/<NJSname>`

Please note that in this configuration file the site name cannot contain spaces. Some of the DEISA sites are configured with spaces in the names, such as *IDRIS ZAHIR*. To specify this as a site name within the site configuration file, replace the space with `%20` as shown in the example above.

Optionally, a row may also contain the user's password for the certificate in the third column. Supplying the password for a site means that the user is not required to enter the password for each certificate when performing DESHL operations. Note

that as passwords are currently stored in an unencrypted state in the configuration file, if the user chooses to supply passwords in the configuration file then it is the responsibility of the user to ensure that the configuration file is held in a secure location. On UNIX systems, users should change the permissions of the configuration file to 600 to ensure that no other users can view the contents of the file. (The UNIX command script contains a check that will print a warning if the specified configuration file cannot be found, or has different file permissions from expected.)

```
$ chmod 600 config.csv
```

Optionally, a row may also contain a shortcut name in the fourth column. This means that the user is not required to type the full path of the site each time. For example, if a user wishes to copy a file between HPCx and IDRIS, using full paths the required command would be

```
$ deshl copy  
ssl://admin.hpcx.ac.uk:4433/HPCx/home/myfile.txt  
ssl://admin.hpcx.ac.uk:4433/IDRIS%20ZAHIR/home/myfile.txt
```

Using shortcut names, this becomes the following:

```
$ deshl copy hpcx/home/myfile.txt idris/home/myfile.txt
```

Within DEISA each user has a default home site, and the DEISA NJS names are specified relative to this home site (as in the example above).

In the configuration file, each site requires a certificate to be specified. Note that if all sites are specified in the file as using the same certificate, the user will only be prompted for one password per DESHL operation (or never prompted for the password if the password is supplied in the configuration file for the first site). This is the normal case for DEISA, where a single EU Grid PMA compliant certificate for a user is accepted by all sites.

A list of the sites configured in the site configuration file can be displayed by running the sites command:

```
$ deshl sites
```

4.4 Edit the logging configuration file

DESHL 3.0 uses the Apache log4j library [5] to allow logging of diagnostic output during DESHL operations. This is particularly useful for troubleshooting or if the user needs to send output to the DESHL developers for analysis. Whilst the user is encouraged to explore the online documentation for the log4j project, the following describes how to enable logging for DESHL 3.0.

In the DESHL installation directory, there is a file called `clt-deshl-log4.properties`. This contains the properties used to control the log4j behaviour.

The actual output file is set in the property `log4j.appender.LOGFILE.File`. By default this is set to create output in the file `clt-deshl.log` in the release directory.

```
log4j.appender.LOGFILE.File=clt-deshl.log
```

The last three properties in the file control how much information is output to the logging file.

```
log4j.logger.org.saga=INFO
log4j.logger.org.deisa.jra7=INFO
log4j.logger.de.fzj=INFO
```

The log4j library defines the following logging levels:

DEBUG	The DEBUG Level designates fine-grained informational events that are most useful to debug an application.
INFO	The INFO level designates informational messages that highlight the progress of the application at coarse-grained level.
WARN	The WARN level designates potentially harmful situations.
ERROR	The ERROR level designates error events that might still allow the application to continue running.
FATAL	The FATAL level designates very severe error events that will presumably lead the application to abort.
ALL	The ALL Level has the lowest possible rank and is intended to turn on all logging.

Normally the DESHL client should be run with logging set to INFO for the three libraries listed above. The user may however change the logging levels to facilitate troubleshooting or debugging.

The user is invited to search online for one of the many log4j tutorial examples.

4.5 Verify the settings

As a simple test to verify that the DESHL client is correctly installed and configured, for each site configured try to list the contents of the home storage.

```
$ deshl list <gatewayName:port_no>/<njsName>/home/
```

or if using shortcuts:

```
$ deshl list shortcut/home/
```

For each site, the DESHL client should display a list of the files and directories in the top level of the HOME storage. If this fails or there are any errors, please consult the troubleshooting guide for further information. (The HOME storage is the site-specific home directory for the user defined in the UNICORE incarnation database [7].)

5. Data Staging

The DESHL client allows access to resources exposed by each UNICORE NJS. Each site provides access to a set of storages which the user may use to stage data in for jobs, retrieve data files produced by jobs etc.

5.1 File naming conventions

The DESHL client uses a convention for accessing a particular location on a DEISA site. This requires that all DEISA locations are specified as full URLs using SSL transport.

```
ssl://<gatewayName>:<port_no>/<njsName>/<storageName>/<filePath>
```

For example, for a file mydata.txt on storage “home” on HPCx the full URL for the file might appear as

```
ssl://host.epcc.ed.ac.uk:4010/myNJS/home/mydata.txt
```

Alternatively, if a shortcut (e.g., “myNjs”) has been defined for the NJS, the location could be specified to the DESHL client as

```
myNjs/home/mydata.txt
```

Assuming that the shortcut matches a shortcut configured in the DESHL client configuration file, the client automatically expands this to a full URL.

5.2 Listing available storages

The list of storages which a user may access at a DEISA site can be listed by running the list command, specifying the `-s` option.

```
$ deshl list <sitename> -s
```

where `<sitename>` is a site or site shortcut name defined in the site configuration file.

5.3 Copy

The DESHL copy command is used to import files from local storage onto the DESHL file system, to export data from the DESHL file system to local storage and to copy files between DEISA sites.

```
$ deshl copy <source> <target> [arguments]
```

Where the optional arguments are:

- 0: do not overwrite an existing target
- 1: overwrite existing target
- 2: non-recursive, do not copy directories
- 3: recursive, directory copy allowed

If no optional arguments are specified, the default behaviour is non-recursive, non-overwrite. Note that optional arguments must be separated by whitespace characters.

In similar fashion to UNIX commands, the command gives no output on success, but prints an exception message if any errors are encountered.

DESHL will initiate an import to the DESHL file system if the source file is on a local file system and the target is a location on the DESHL file system.

DESHL will initiate an inter-site transfer if the source and target are both valid locations on the DESHL file system but on separate sites.

DESHL will initiate an export if the source is a location on the DESHL file system and the target is a location on a local file system.

Currently the DESHL client cannot export directories; therefore remote files must be exported individually.

Note that the DESHL client cannot be used to copy between two locations which are both on a local file system not managed within the DEISA UNICORE Grid, such as between two directories on a user's local workstation.

5.4 *Exists*

This command is used to determine if a remote file or directory exists. It takes a single argument, the URL of the remote file or directory whose existence is to be verified.

```
$ desh1 exists <remote_filename>
```

The command prints out an appropriate message on successfully identifying if the path does exist or not, or an exception message if any errors were encountered.

5.5 *isDir*

The isDir command is used to determine if a remote location is a directory. It takes a single argument, the URL of the remote path.

```
$ desh1 isDir <path>
```

The command prints out an appropriate message on successfully identifying if the path is a directory or not, or an exception message if any errors were encountered.

5.6 *isFile*

The isFile command is used to determine if a remote location is a file. It takes a single argument, the URL of the remote path.

```
$ desh1 isFile <path>
```

The command prints out an appropriate message on successfully identifying if the path is a directory or an exception message if any errors were encountered.

5.7 *list*

The list command is used to list the contents of a remote directory. It takes a single argument, the full URL of the remote directory name.

```
$ deshl list <directory_name>
```

The command prints a directory listing on success, and prints an exception message if any errors were encountered.

The list command can also be used to list the storages available at a particular site using the `-s` option. (See Section 5.2)

5.8 *makeDir*

The makeDir function is used to create a remote directory. It takes the full URL of the directory to be created, plus optional arguments.

```
$ deshl makeDir <directory_name> [arguments]
```

Where the optional arguments are:

- 0: fail if parent directory does not exist (default)
- 1: create parent directories if not existing

The command prints an exception message if any errors are encountered.

5.9 *move*

The move function is used to rename a remote file or directory. It takes the full URLs of the file or directory to be renamed, the new name plus optional arguments. The move function is identical to the copy function, except that the source file or directory is deleted on successfully performing the copy operation.

```
$ deshl move <source> <target> [arguments]
```

Where the optional arguments are:

- 0: do not overwrite existing target (default)
- 1: overwrite allowed

The command prints a directory listing on success, and prints an exception message if any errors are encountered.

Note that the DESHL client cannot be used to move between two locations which are both on local file systems not managed within DEISA UNICORE Grid, such as between two directories on a user's local workstation.

5.10 *remove*

The remove function is used to delete a remote file or directory. It takes the full remote path plus optional arguments.

```
$ deshl remove <path> [arguments]
```

Where the optional arguments are:

- 0: do not delete directories (default)
- 1: allow deletion of directories

The command prints an exception message if any errors are encountered.

Note that the remove command cannot be used to delete files on local file systems.

6. Job submission and management

The DESHL 3.0 client allows the user to submit jobs to DEISA sites, monitor job status, terminate jobs and retrieve job output and error output.

6.1 Submitting a job

To submit jobs, the DESHL client requires job scripts which define SAGA job directives. An example of a simple job script is shown below.

```
#!/bin/bash
# Test job script for DESHL using SAGA.
#
# SAGA JobDefinition based directives:
#$ SAGA_JobCmd = hello.sh
#$ SAGA_FileTransfer = file:///jobs/hello.sh#HOME > hello.sh
#$ SAGA_HostList = ssl://myhost.ac.uk:4433/myNJS
#$ SAGA_JobEnv = account_no=xxx
#
```

Listing 1: SAGA directives in job script.

In the above example, it is assumed that an executable job script called hello.sh exists on the target machine, with the URL

```
ssl://myhost.ac.uk:4433/myNJS/home/jobs/hello.sh
```

Note that the script containing the SAGA directives is not itself sent to the server, and is instead used to build a set of instructions for managing the job. The script to be executed on the server must already exist in the specified location (or can be uploaded to the DEISA site using the data staging commands).

The submit command takes a number of options followed by the SAGA batch job script file and any arguments to pass to that file:

```
$ deshl submit [options] <job_script> [arguments]
```

The job script can also contain directives. This command aims to follow the `qsub` Open Group specification for job submission [4].

From that specification the following options are supported, as shown in Table 2:

Short Option	Long Option	Name	Description
C	directive	Prefix	The prefix used for directives in the job script file
H	help	Help	Used to obtain a summary of the available options and command syntax
N	name	job name	The user defined name for the job
Q	destination	destination	The site the job should run on
V	variables	environment variables	A single name=value environment variable

Table 2: Job submission options

Job submission also supports directives in the job script file. A directive is a special comment line in that file. It must occur within the heading comment block, where the heading comment block must start from the first line of the file. The only supported comment designator is the hash or pound symbol, '#'. This must be the first character to be a valid comment line. For the comment line to be a directive the comment symbol must be immediately followed by the directive prefix symbol, by default the dollar symbol, '\$', but can be specified on the command line.

Directives are used to allow the user to set values for the SAGA `JobDefinition` [3] attributes in Table 3. The value for a directive is everything following the equals, '=', up to the end of the line, and then trimmed for whitespace characters. It is up to the user to supply suitable values for the targeted resource manager. The values shown below are examples.

SAGA JobDefinition attribute	Description
SAGA_JobCmd	The job executable path. The only required attribute. By default this is set to the <code>job_script</code> parameter specified on the command line, but the user can override this by setting this directive (recommended). <pre>#\$ SAGA_JobCmd = /my/job.exe</pre>
SAGA_JobArgs	An argument to the job. Usually set from arguments on the command line. <pre>#\$ SAGA_JobArgs = arg #\$ SAGA_JobArgs = another arg</pre>
SAGA_JobEnv	An environment variable for the job. <pre>#\$ SAGA_JobEnv = val=val1 #\$ SAGA_JobEnv = var2=val2</pre>
SAGA_JobName	A name for the job. <pre>#\$ SAGA_JobName = my job</pre>
SAGA_FileTransfer	A file transfer local to the destination site, by default all stage ins are to the USPACE, all stage outs HOME. A stage in is designated by a >, a stage out by a <. File values must be a valid URI. <pre>#\$ SAGA_FileTransfer = file:///dat#HOME>file:///dat#USPACE #\$ SAGA_FileTransfer = file:///out#HOME<file:///out#USPACE</pre>
SAGA_HostList	The destination host for the job. This must be the locator for a Unicore NJS the user has access to, i.e. specified in the DESHL client configuration file. (Note that shortcuts cannot be specified here.) <pre>#\$ SAGA_JobHostList = ssl://gateway.host:4433/SomeNJS</pre>
SAGA_NumTasks	The number of Tasks <pre>#\$ SAGA_NumTasks=3</pre>
SAGA_NumCpus	The number of Threads per Task

	<code>#\$ SAGA_NumCpus=5</code>
SAGA_Memory	<p>The total maximum memory limit for the job in MB. This is used to set the data limit as</p> <pre>data_limit =SAGA_Memory / SAGA_NumTasks</pre> <p>It is also used to set the stack limit (if not already set) to <code>SAGA_Memory / 6</code></p> <p>The stack limit may also be set explicitly by setting an environment variable called <code>'stack_limit'</code></p>
	<code>#\$ SAGA_Memory=256</code>
SAGA_WallClockSoftLimit	The time the job should run for in seconds
	<code>#\$ SAGA_WallClockSoftLimit=600</code>

Table 3: SAGA JobDefinition attribute directives.

A directive must occupy a single line, with only one directive allowed per line; the directive examples shown in Table 3 are wrapped for space reasons.

The file conventions used are the same as for the data staging commands. Note that only local file transfers to the site are allowed when submitting a job. This is to allow files that must be in the USPACE when the job runs to be staged in and to allow files to be staged out of the USPACE when the job completes. For other file transfers use the appropriate data staging command. Any scripts or files which are required by the job must be explicitly staged into the USPACE using the SAGA directives.

A successful submission results in a message containing the job identifier on standard output. The job identifier is an encoded combination of the job number and the site where the job was submitted, and uniquely identifies the job within DESHL. The following shows an example submission:

```
$ deshl submit sleeper.sh 180
Your job: x957383131ssl%3A%2F%2Fmyhost.ac.uk%3A4433%2FmyNJS, has
been successfully submitted.
```

Where the job identifier is

```
x957383131ssl%3A%2F%2Fmyhost.ac.uk%3A4433%2FmyNJS
```

If the job submission is unsuccessful then an error message will be returned on standard error (no output on standard output). Further information may be available from logs.

6.2 status

The status command displays the current status of a submitted job, where `job_identifier` is the job identifier which was displayed in response to a successful submit command, or displayed by the jobs command.

```
$ deshl status [options] <job_identifier>
```

The output from status is a keyword indicating the state of the job at the target site. The SAGA job states supported are:

- **RUNNING** – the job is queued or running

- DONE_OK – the job has finished successfully
- DONE_FAIL – the job has failed

This command aims to follow the `qstat` Open Group specification for job status [1].

6.3 *terminate*

The `terminate` command is used to request that a job be stopped if it is currently running, destroyed and any associated resources freed. This command takes the job identifier created when `submit` was successfully called, or listed by the `jobs` command.

```
$ desh1 terminate [options] <job_identifier>
```

This command aims to follow the `qdel` Open Group specification for job termination [1]. Note that it is up to the remote site to determine when the job should be deleted, therefore the job may continue to appear in the list of jobs for some time after the request has been sent. When the job has been terminated, the job status appears as `DoneFail` when viewed using the `status` command. Any output files generated by the job up to point of termination can be retrieved using the `fetch` command. (Note that once a job has been terminated, it cannot be restarted.)

6.4 *fetch*

Once a job is completed (determined by using the `status` command), then any resources used by the job can be released and any output retrieved by using the `fetch` command. (Note that any resources associated with a job are not released until the user fetches the job.)

```
$ desh1 fetch <job_identifier> [to_dir]
```

The `job_identifier` is one previously returned by a job submission, whose status shows the job is complete (an identifier for a job that has already had `fetch` invoked will no longer be recognised). The optional `to_dir` is the directory to retrieve any output from the job to, if not specified the current directory is used.

On successful completion no output is given but any `stdout` or `stderr` files that the job created will be available in the specified or current directory.

If there was a problem an error message will be displayed on standard error. More information may be available in the logs. (See Section 4.4)

6.5 *jobs*

All of the user's current job identifiers can be retrieved by invoking the `jobs` command.

```
$ desh1 jobs
```

This command queries all of the sites configured in the user's configuration file for the user's current jobs, where a current job is defined as one that has been successfully submitted but not yet fetched. The identifiers for current jobs are collated and presented on standard output with each identifier on a new line. Nothing is output if there are no current jobs.

If there was a problem an error message will be displayed on standard error. More information may be available in the logs.

7. Tutorial

7.1 Upload, submit and manage a simple job

hello-submit.sh

```
#!/bin/bash
# Test job script for DESHL using SAGA.
#
# SAGA JobDefinition based directives:
#$ SAGA FileTransfer = file:///jobs/hello.sh#HOME > hello.sh
#$ SAGA_HostList = ssl://admin.hpcx.ac.uk:4433/HPCx
#$ SAGA JobEnv = account no=xxx
#$ SAGA JobCmd = hello.sh
```

hello.sh

```
#!/bin/bash

echo "Here I am on `hostname` in $PWD. Current date is
`date`"
```

The installation bundle contains two simple script files; a SAGA script, hello-submit.sh, and an executable script, hello.sh. hello-submit.sh contains a set of SAGA directives for submitting and managing the execution of a job, the job in this case being the executable script, hello.sh.

This tutorial will show how to import the executable script onto a DEISA site, and then use the SAGA script with the DESHL command line tool to submit the job and retrieve the output from the job.

The following steps are required:

1. Verify if a remote directory exists
2. Create a remote directory for the script
3. Upload the executable script to the server
4. Edit the SAGA script
5. Submit the job
6. Get the job status
7. Cleanup the job and retrieve the job output

1. Verify if a remote directory exists

First we want to determine if the directory exists:

```
$ desh1 exists hpcx/home/jobs
```

If the remote directory exists, the following message should be displayed:

```
exists: target hpcx/home/jobs exists
```

Otherwise, the following message is displayed:

```
exists: target hpcx/home/jobs does not exist
```

2. Create a remote directory for the script

If the directory does not exist, create it using the `makeDir` command:

```
$ deshl makeDir hpcx/home/jobs
```

3. Upload the executable script to the server

Next we need to upload the job script, `hello.sh`, from the local machine to the DEISA site where the job will be run using the `copy` command:

```
$ deshl copy /home/deshl/hello.sh hpcx/home/jobs/hello.sh
```

4. Edit the SAGA script

Next, we need to edit the SAGA script, `hello-submit.sh`, to match the DEISA site that we want to run on. `SAGA_HostList` must be changed to be the NJS to which the job will be submitted. On the line which contains "`SAGA_JobEnv = account_no=xxx`", the `account_no` value must be set to a valid account on the target machine. (Please contact the administrator of the site to which you intend to submit jobs for an appropriate account number.)

5. Submit the job

We can now submit the job.

```
$ deshl submit hello-submit.sh
```

(A message will be displayed informing that the default job name, the name of the script, has been overridden by the `SAGA_JobCmd` directive in the script – please see the note at the end of this section.)

This returns a job identifier, which we can use to monitor the job's status

6. Get the job status

```
$ deshl status  
x957383131ssl%3A%2F%2Fmyhost.ac.uk%3A4433%2FmyNJS
```

Once the job has successfully completed, the status should be reported as `DONE_OK`

7. Cleanup the job and retrieve the job output

The output from the job can now be retrieved and any resources consumed by the job released, using the `fetch` command. The `fetch` command allows us to retrieve the job output into a specified directory; for this example we will retrieve into the local directory `/home/deshl/joboutput`

```
$ deshl fetch x957383131ssl%3A%2F%2Fmyhost.ac.uk%3A4433%2FmyNJS -d
/home/deshl/joboutput
```

This will place two files in the specified directory; one of these is the job's output to stdout, and the other is the job's output to stderr.

Note 1: Please be aware that on the client, only the SAGA directives in a SAGA script file like hello-submit.sh are of interest, and the other lines are ignored, whereas on the host where the job is to be executed, any SAGA directives in a file are ignored and only the system commands are of interest.

Note 2: Normally the name of the script or executable to be run on the remote machine is specified via a SAGA_JobCmd property in the client script file, as shown in the above example (hello-submit.sh). However, if this is not present, this value is defaulted to the name of the script specified to the submit command. In this case, the name of the executable on the remote machine must EXACTLY match the name of the script specified to the submit command.

7.2 Staging in data for a job, staging out results

In this tutorial we demonstrate how to stage in data required by a job and how to retrieve data produced by a job. Our example script in this case will perform a simple concatenate on two specified files and the file containing the joined files will be our result.

catfiles-submit.sh

```
#!/bin/bash
# Test job script for DESHL using SAGA.
#
# SAGA JobDefinition based directives:
#$ SAGA FileTransfer = file:///jobs/concat.sh#HOME >
catfiles.sh
#$ SAGA FileTransfer = file:///data/myfile1.dat#HOME>filea
#$ SAGA FileTransfer = file:///data/myfile2.dat#HOME>fileb
#$ SAGA FileTransfer = file:///data/myfile3.dat#HOME<filec
#$ SAGA HostList = ssl://admin.hpcx.ac.uk:4433/HPCx
#$ SAGA JobEnv = account no=xxx
#$ SAGA JobCmd = concat.sh
```

concat.sh

```
#!/bin/bash

cat filea fileb > filec
```

As in the previous tutorial, use the copy command to import the executable script file, concat.sh, to the jobs directory on the appropriate DEISA host, and use the makeDir and copy commands to create a data directory and import the data files to the remote data directory. Remember to edit the SAGA script file, catfiles-submit.sh, to set the correct host and account information as in the previous example.

The job can now be submitted as before:

```
$ deshl submit catfiles-submit.sh
```

The job status can be monitored as before using the status command. When the job has completed, an output data file called myfile3.dat will have been produced in the remote data directory. This can be exported to the local file system for inspection using the copy command:

```
$ deshl copy hpcx/home/data/myfile3.dat /home/deshl/myfile3.dat
```

At this point the job still exists on the site where it was submitted. To free any resources associated with the job, the job should be fetched as in the previous example, or terminated. If the job is terminated, any resources are freed and no stderr or stdout produced by the job is retrieved by the client.

Important Note: In the directives, a ">" operator is used to donate copying of files into the USPACE where it can be accessed by the job. Conversely, a "<" operator is used to donate output from the USPACE to the DEISA file system on completion of a job. Note also that only locations within the DEISA file system may be specified for either of these operators, local files may not be specified. Instead, use the DESHL client data staging commands for copying files before running a job or retrieving output after a job has run.

8. Trouble shooting guide

8.1 File Permissions

Currently when files are copied to a target host the file permissions are not preserved. This will remove the execute permission when copying over an executable. To get round this a `chmod` job can be submitted to change the file's permissions once it has been copied.

For example consider the `hello.sh` job from above, Section 7.1:

```
$ deshl copy /home/deshl/hello.sh hpcx/home/jobs/hello.sh
```

When this is copied onto the DEISA file system, it may not have execute permissions and subsequently fail when an attempt is made to run the script. To overcome this, run the `chmod.sh` job (which is included in the `jobs` directory of the DESHL Command Line Tool release.) Edit the SAGA directives in the header comment to appropriate values.

```
#!/bin/bash
## SAGA FileTransfer = file:///bin/chmod#ROOT > chmod.sh
## SAGA HostList = ssl://admin.hpcx.ac.uk:4433/HPCx
## SAGA_JobEnv = account_no=xxx
```

Listing 2: Partial contents of `chmod.sh` showing SAGA directives.

Edit the location of `chmod` if it is different for the target host, set the target host to that which the file was copied to earlier. Add the account information if required by the site to which the job is submitted.

The absolute path of the copied file on the target host must be known. In the example it is `/home/deshl/jobs/hello.sh`, this is passed as an argument to `chmod.sh` along with the chosen file permissions, in this case `u+x` to set the file executable.

```
$ deshl submit chmod.sh u+x /home/deshl/jobs/hello.sh
```

Once this completes the copied file will have execute permissions and can then be submitted in the way shown in Section 7.1

8.2 Script paths

As mentioned previously in section 7.1, if job scripts are submitted which do not explicitly specify a `SAGA_JobCmd` directive, the name of the SAGA script specified to the `submit` command is taken as the name of the script which will be used by UNICORE, **including the path of the script**. For example, running `hello.sh` from section 7.1, but with the `SAGA_JobCmd` property removed

```
$ deshl submit hello.sh
```

results in UNICORE looking for a script called `hello.sh`, which must be explicitly staged in via the `SAGA_FileTransfer` directive.

However, copying the script into a different local directory, for example `myjobs`, and then running the script will then fail, as UNICORE will look for a file named

myjobs/hello.sh in the USPACE, and there will be a mismatch although the client script itself has not changed.

```
$ deshl submit myjobs/hello.sh
```

To avoid this, it is recommended to explicitly specify a `SAGA_JobCmd` directive in the SAGA script. If you find that a working script suddenly starts failing, please ensure that you have explicitly set a `SAGA_JobCmd` directive in the script to eliminate the possibility of path problems.

8.3 Reporting bugs / requesting features

If you wish to submit a bug report, request support or suggest features for future versions of DESHL, please access via the “Tracker” tab at the DEISA JRA7 web site:

<http://forge.nesc.ac.uk/projects/deisa-jra7>