Genesis II Quick User's/Reference Manual

1 Introduction

Genesis II is an integrated implementation of the standards and profiles coming out of the Open Grid Forum Open Grid Services Architecture (OGSA) Working Group. Further, Genesis II is a complete – if somewhat minimal – set of grid services for users and applications which not only follows our maxim – "by default the user should not have to think" – but is also a from-scratch implementation of the standards and profiles – not a wrapping of existing artifacts.

Genesis II is combined development and research project at the Virginia Center for Grid Research (VCGR) and the UVa Department of Computer Science. The goal of Genesis II is develop an open source, standards-based grid platform to support both high-throughput computing and secure data sharing. Such a platform will serve the grid user community by providing free, easy-to-use tools to exploit grid technology to fulfill their computation and data management needs. For the grid research community, Genesis II is designed to be an interoperable (via adherence to emerging standards), freely available (via open-source), and flexible platform to develop new grid technologies and models without researchers having to create entire new grid implementations from scratch. VCGR will lead the way in exploiting Genesis II as both a user platform and a research platform.

Genesis II features include:

- ***Web Services based implementation*** promotes interoperability as well as simpler integration with many 3rd party products.
- ***Data and Compute technology*** to support a wide range of potential application domains from simple data sharing to multi-platform, multi-organizational high-throughput computing. Grid computing technology is still in its infancy and many problems remain. Particular areas that we will focus on are data access, security (confidentiality, data integrity, access control, policy negotiation), dependability (availability, SLA's, policy languages), and grid standards.
- ***Familiar Hierarchical Namespace*** gives users a comfortable environment to work in which means that less time is spent learning new tools and syntax.
- ***Open Source code-base*** as part of our ongoing commitment to provide not only fully integrated platforms for production use, but also to encourage outside development and research efforts.
- ***OGSA Standards Based implementation*** will allow for Genesis II to interoperate/interact with other developing grid efforts as those standards move forward.
- ***Secure infrastructure based on emerging web standards*** (such as XACML, SAML, and WS-Security) further promotes interoperability and integration without sacrificing security.
- ***Multi-platform support*** allows for organizations to make better use of everything from their cluster of Linux boxes to the Windows desktops at their users' desks.

- ***Robust data services with familiar interfaces*** are essential to adoption Genesis II bases much of its data services on standards being developed by the OGSA and other OGF working groups:
    - ByteIO
    - RNS
    - ExportDir
    - Familiar, Unix-like command line tool
- ***High-throughput computing motivates much of the design***. Genesis II compute resources are provided via OGSA-BES and also leverage RNS interfaces to facilitate easy user interaction:
    - BES Container
    - BES Queue
- ***Other OGSA and OGF technologies influence various aspects of Genesis II***. Many core architectural components of Genesis II are based on various standards coming out of the OGF and the Web Services world:
    - WS-Naming
    - WS-Security

1.1 XML Namespaces

Various components of Genesis II rely on XML documents. These include configuration files, SOAP messages, etc. The following table lists namespaces and their associated namespace prefixes as used throughout this document.

Namespace Prefix	Namespace
mconf	http://www.mark-morgan.net/org/morgan/util/configuration
genii	http://vcgr.cs.virginia.edu/Genesis-II
gsh	http://vcgr.cs.virginia.edu/genii/xsh/script
geniix	http://vcgr.cs.virginia.edu/genii/xsh/grid

1.2 Conventions

Throughout this document, we use a number of notational conventions to aid the description of various key features and abilities of Genesis II. These are listed below:

- **${GENESIS_II}** is used to indicate the base directory or path location of the Genesis II installation on a given system;

1.3 Key Concepts

In order to use and configure GenesisII, it is useful to understand some key concepts about Web/Grid services in general and the design of GenesisII.

1.4 Genesis II State

All state in the Genesis II system is stored under a user specific directory. This state is often maintained by a simple JDBC database called Hypersonic, but some of it is also

stored simply as raw files. Either way, all state is located in **${java.home}**/.genesisII where **${java.home}** is whatever the value of the java.home System property is in your environment (usually this is **$HOME** on a Unix environment and C:\Documents and Settings\login-id on a Windows machine). This directory must reside on local disk for that machine to ensure that the database does not become corrupted. Because of this requirement, it is not possible for multiple machines to share this directory and as such all configuration is done on a machine by machine basis.

If at any time you wish to wipe out all state associated with a Genesis II container and re-initialize the system, you may do so by deleting the **${java.home}**/.genesisII directory and once again following the steps outlined in this document for bootstrapping a system. Note that if you do this, all state from the previous grid will be lost.

1.5 Configuration

Genesis II is highly configurable allowing administrators to tailor the system in many different areas. This includes the ability to dynamically add or remove tools available to clients, the ability to specify certificate and key stores to use for various aspects of the Genesis II security infrastructure, the ability to dynamically configure ports, directories, etc.

Two configuration files exist within Genesis II, and both can be located in the **${GENESIS_II}**/configuration folder. The first of these, client-config.xml describes various properties of the Genesis II system when used as a client[1], while the server-config.xml file includes configuration options of relevance to server side operations and functions. Both files have the same general format.

Each configuration file is an XML document containing a number of child elements which represent sections of the configuration. One section in particular, the mconf:config-sections element, is required and indicates to the configuration engine the list of sub-sections to expect within that given configuration document.

1.6 Security Concepts

Security is an extremely important part of any grid system and Genesis II offers a wide range of configurable security options. Throughout this document, we will describe mechanisms for affected and altering the security of various components and sessions, but before we introduce any commands to this effect, we describe here concepts that Genesis II makes use of and that you will need to be familiar with in order to successfully secure your Genesis II system.

1.6.1 Transport Layer vs. Message Layer Security

Genesis II allows for both transport layer and message layer security but the latter is discouraged given the tremendous performance cost of encrypting SOAP content in

[1] In this context, client refers to the role a piece of code takes on, not a specific tool or application. In other words, even code within a service can participate in a client role if it makes calls on other remote services using the Genesis II client libraries and stubs.

XML. By default, Genesis II ships with transport layer security turned on (i.e., HTTPS communication) with server-side authentication only. Many of these settings can be changed in the client and server configuration files.

1.6.2 Genesis II Lightweight SAML (GAML)

Genesis II supports a SAML-like mechanism for authorizing client access to services and resources. This GAML authorization module permits administrators and users to associate X.509 certificates with read, write, and execute categorized access control lists on individual resources arbitrarily. When configured correctly, these access control lists will limit the access that various users have to the system resources.

Flexibility is a key design goal of the Genesis II system and as such we place very little restrictions on the types of X.509 certificates that can be used for identification of users. Any valid X.509 certificate can be used so long as that certificate, or one of the certificates in its certificate chain[2], appears in the appropriate access control list for the target resource.

Finally, Genesis II also employs a form of delegation that allows clients to pre-allow a number of delegations to occur on its behalf. In this way, if client A performs an operation on service B, which in turn needs to perform an operation on service C, then service B will be able to perform that operation cryptographically on behalf of Client A without Client A necessarily being contacted about the delegation. This mechanism improves the decoupling of the architecture and can be limited or relaxed as necessary in the configuration files.

1.6.3 X.509 Concepts and Tools

Because X.509 is used so heavily in Genesis II (both for GAML authentication and for transport layer security), a number of X.509 concepts are particularly relevant to this document. We define the following terms to aid in this discussion:

- **Key Store** – A location where X.509 certificates along with their private keys are stored along with an alias or friendly name that identifies them. This key store can be a file in a number of different file formats including PKCS12, JCS, and BC. The key store can also be the Windows Certificate store maintained automatically by the Windows operating system. Because key stores contain private keys, they are often password protected and typically used when clients or services need to take on the role of a given certificate.
- **Public Certificate** – Unlike a Key Store which contains both a certificate and a private key, a Public Certificate is a completely public entity which is used as the public identity of a user or resource. These are often stored in DER format certificate files and are usually used to indicate which certificate to add to an access control list. Because these files are public, they are not typically password protected.

[2] Genesis II can use certificates from a certificate chain to provide a rudimentary grouping mechanism for access control. For example, by adding the University of Virginia Standard Assurance PKI certificate to the access control lists for a target resource, I could grant access to any valid certificate produced by that CA.

- **Certificate Authority or CA** – A certificate authority or CA is a certificate or entity that, using its private key, has the authority to generate new children certificates. Often times this certificate is maintained by an organization that has been granted the authority to generate certificates for its members (such as the University of Virginia generating certificates for its students and faculty). However, in Genesis II, containers and services also can generate certificates (making them certificate authorities) for the purposes of identifying their own children resource. Further, many organizations will find it useful to generate a certificate authority for easy generation of user certificates for grid users such as administrators and managers.

To aid in the construction of these various components, Genesis II includes a certificate tool (cert-tool) which allows users to easily generate certificates and certificate authorities. Further, Java ships with its own tool (keytool) which can be used to extract public certificates from key stores. Both of these tools are used during a typical bootstrapping of the Genesis II software. The following sections detail how these tools can be used to generate certificate authorities, certificates and keys, and public certificate files.

1.6.3.1 Creating a CA in a Key Store

The cert-tool program included with Genesis II can easily be used to create a new CA. In effect, this is just another entry in a key store with an alias. However, while most certificates will be signed (authorized) by a CA, in this case the CA certificate will sign itself. This is not necessary for the use of Genesis II, but it provides a convenient way of grouping certificates together. If you choose to follow this model, the following example demonstrates a typical use of cert-tool for creating a CA.

```
cert-tool gen –dn=”C=US, ST=Virginia,
   L=Charlottesville, O=UVA, OU=VCGR, CN=VCGR CA” –
   output-keystore=ca.pfx –output-keystore-
   pass=password –output-alias=”VCGR Certficiate
   Authority”
```

In this example we create a certificate with a distinguished name (dn) of “C=US, ST=Virginia, L=Charlottesville, O=UVA, OU=VCGR, CN=VCGR CA”. We have specified ca.pfx as the key store file to use (and the default store type of PKCS12 will be used). Finally, we indicate a password for the key store, and an alias for our CA certificate and key. All of these parameters should be changed to fit each organizations preferences.

1.6.3.2 Importing a CA certificate into a Trust Store

Once you have created a certificate authority certificate, you probably will want to import this certificate into your trust store. Clients of the Genesis II system validate the certificate on the SSL connection by using your trust store. They also validate the certificates for resources on that system using this trust store. By default, this trust store is located in **${GENESIS_II}**/security and is called trusted.pfx. While you cannot change the location of this trust store, you are welcome to change the name of the file

itself, or the store type and/or password. To make these changes, you will need to edit the appropriate sections inside the **${GENESIS_II}**/configuration/client-config.xml configuration file. The two sections in question are the genii:ssl-properties and the genii:resource-identity sections.

The following example shows how to use cert-tool to import the ca certificate created above (parameters should be changed as appropriate).

```
cert-tool import –input-keystore=ca.pfx –input-keystore-
pass=password –input-alias="VCGR Certificate Authority" –
output-keystore=trusted.pfx –output-keystore-pass=trusted –
output-alias="VCGR Certificate Authority"
```

1.6.3.3 Creating a Certificate and Private Key from a CA

Given a certificate authority, you will likely want to create some certificates from that certificate authority. In particular, you will likely want to generate a host certificate for each container (unless your host already has a valid host certificate). This certificate is used by each container to create the SSL connection and many SSL clients require that this host certificate contain in the DN (in particular, the common name or CN field) the DNS name of the host in question. Additionally, one possible configuration for your Genesis II system will involve creating an "Administrator" certificate from this CA. While the Genesis II software does not require the notion of an administrator, the idea of a management level or root level certificate for your own resources if a useful abstraction. The following command line illustrates how the CA created above can be used to create a new Administrator certificate. The command line should be changed as appropriate for host certificates.

```
cert-tool gen –dn="C=US, ST=Virginia,
   L=Charlottesville, O=UVA, OU=VCGR,
   CN="Administrator" –input-keystore=ca.pfx –input-
   keystore-pass=password –input-alias="VCGR
   Certificate Authority" –output-keystore=admin.pfx –
   output-keystore-pass=admin-password –output-
   alias="Administrator"
```

1.6.3.4 Extracting the Public Certificate from a Key Store Entry

So far, the certificates we have generated contain private keys and can be used by various cryptographic entities to take on the role or identity that a certificate and key indicate. However, it is also useful to have a public identity that can be shared with others that can later be added to access control lists. In essence, these public identities are the certificates in question minus the private keys. Genesis II allows for public certificates to be added to access control lists either directly from the local file system, or from the grid file system. Either way however, you will need to extract these public certificates from their various key stores. If these keys are contained in non-Java key stores (such as the Windows Certificate store), then you will need to use some key store specific means for extracting the information. If however they were generated by Java keytool, or by the Genesis II cert-tool applications, you can use the Java keytool program to extract them.

In the following example, we extract the public certificate for the Administrator identity created above. Once again, parameters should be changed to reflect desired usage properties.

```
keytool –export –keystore admin.pfx –storetype PKCS12 –
   alias Administrator –file admin.cer
```

2 Download

Genesis II is distributed freely to the public under an Apache-style open source usage agreement. To download a copy of Genesis II, go to the web page http://vcgr.cs.virginia.edu/genesisII/downloads.html and follow the instructions there.

3 System Bootstrap

We break the bootstrapping process up into two phases; security set up, and system initialization. You may choose to ignore the security set up phase if you wish to run a simple test net of Genesis II. If you do this, Genesis II will by default bootstrap with a container certificate called *skynet* for each container and will use this certificate to assign initial access control rights for bootstrap resources. Further, this will effectively become your administrator certificate and as such you will need to log in to this certificate when bootstrapping the net. If you choose to do this, when it comes time to log in to the system for bootstrapping, you will be logging into the **${GENESIS_II}**/security/keys.pfx key store with the password of keys.

For the remainder of the bootstrap section of this document we assume that you are creating a Genesis II network on 2 machines (*host1.domain.name* and *host2.domain.name*) and that host1 will be the primary, or bootstrap host (i.e., this will be the host on which you will run *MOST* of the commands).

3.1 Security Set Up

In this section, we describe a typical security set up for the two-host system described above. Extensions to multiple hosts beyond two should be performed similarly and should be evident to the reader.

First, you should create a certificate authority from which you will form container certificates and the administrator certificate. Follow the directions given in the previous section for creating a certificate authority of your choice. For this example, we use the following command line:

```
cert-tool gen –dn=”C=US, ST=Virginia,
   L=Charlottesville, O=UVA, OU=VCGR, CN=VCGR CA” –
   output-keystore=ca.pfx –output-keystore-
   pass=password –output-alias=”VCGR Certificate
   Authority”
```

For this one certificate, it will not be necessary for you to change any Genesis II configuration files. This certificate is not used directly by the system but rather is useful for you as an administrator to help configure your new system.

Next, using this certificate authority, you should create an administrator certificate and two host certificates (one for each host). The host certificates should be copied to appropriately named files on the respective host machine installation directories. While you can use any file name you like for these certificates, they must reside inside of the **${GENESIS_II}**/security directory. For the purposes of this example, we will call these files host1.pfx and host2.pfx respectively. The command lines which create these three certificates are given below. As before, all certificates indicated can have any parameter values that are appropriate for your organization with the exception of the output alias. Currently, the Genesis container assumes that the name of the container certificate is "VCGR Container". If you change any other values for the host/container certificates, these changes must be reflected in the genii:ssl-properties and genii:resource-identity sections within the **${GENESIS_II}**/configuration/server-conf.xml file.

```
cert-tool gen –dn="C=US, ST=Virginia,
   L=Charlottesville, O=UVA, OU=VCGR,
   CN="Administrator" –input-keystore=ca.pfx –input-
   keystore-pass=password –input-alias="VCGR
   Certificate Authority" –output-keystore=admin.pfx –
   output-keystore-pass=admin-password –output-
   alias="Administrator"

cert-tool gen –dn="C=US, ST=Virginia,
   L=Charlottesville, O=UVA, OU=VCGR,
   CN="host1.domain.name" –input-keystore=ca.pfx –
   input-keystore-pass=password –input-alias="VCGR
   Certificate Authority" –output-keystore=host1.pfx –
   output-keystore-pass=host1-password –output-
   alias="VCGR Container"

cert-tool gen –dn="C=US, ST=Virginia,
   L=Charlottesville, O=UVA, OU=VCGR,
   CN="Administrator" –input-keystore=ca.pfx –input-
   keystore-pass=password –input-alias="VCGR
   Certificate Authority" –output-keystore=admin.pfx –
   output-keystore-pass=admin-password –output-
   alias="VCGR Container"
```

After these certificates have been created, you will need to export the administrator public certificate so that it can be used for initial access control lists during the system initialization process. This identity will automatically be given read, write, and execute permission for all service resources (those that are created by default on the web server). The following command line usage exports the public admin certificate for the identity created above:

```
keytool –export –keystore admin.pfx –storetype PKCS12 –
   alias Administrator –file admin.cer
```

The `file` parameter indicates what the name of the generated public exported certificate file should be (DER encoded) and can be any file you choose. Once this is done, you should edit the **${GENESIS_II}**/configuration/server-conf.xml file to reflect this change. Inside this configuration file is a section called genii:authorization that contains a property indicating the authorization model to use, and a commented out property indicate the certificate to use for default access control. Uncomment this latter property and point it at the admin public certificate file you just created. The following shows an example of what this new section should look like.

```
<genii:authorization>
     <mconf:property
          name="genii.security.authz.authz-enabled"
          value="true"/>
     <mconf:property
          name="genii.security.authz.bootstrapOwnerCertPath"
          value="security/admin.cer"/>
</genii:authorization>
```

3.2 System Initialization

After the security configuration has been finished, you are ready to bootstrap your new Genesis II system. This involves starting the container on every machine in your planned grid (host1 and host2 in this example) and then running a bootstrap script to create the initial RNS space and setup initial users. Technically speaking a valid Genesis II grid exists without running the bootstrap script, but such a net would be difficult to address and program against. RNS provides a useful abstraction for human interaction with grid services.

You may start the container on a given host by changing to the **${GENESIS_II}** directory and running the `runContainer.sh` script (`runContainer.bat` in Windows environments). This will start the container and for first time system initialization will set some default access control lists for the bootstrap services. While the bootstrap script is run only once for any given grid configuration, this runContainer script should be used every time the container needs to be started (whether it is because the container previously was shut down manually (by terminating the runContainer process) or because of a machine restart.

After the container start-up has finished (indicated by the line “Container Started” appearing on the output console for the runContainer script), you may run the bootstrap script which will set up the first time state for your new grid. This script is an instance of a simple XML scripting language supported by the Genesis II software and will need to be edited to match your site’s local preferences. In particular, you will need to edit the geniix:login command of the script to use the correct admin.pfx file for your configuration (if you are using the default configuration that ships with Genesis II, you may ignore this step).

To run this bootstrap script, you need to invoke the grid script tool with the **${GENESIS_II}**/bootstrap.xml script supplied as its argument:

```
grid script bootstrap.xml
```

The default version of this script assumes a couple of directory and file structures exist, which can be used for initializing users and extra machines in a given net. If these are missing, the script simply ignores those steps. The two file system structures in question are a directory structure containing user and group public certificate files which will automatically be added your new system (both as public certificate grid files, and as access control lists). This directory structure includes a root directory called certs which contains under it a directory called users and another one called groups. Each of these sub-directories in turn may contain as many public certificate files as you wish for each category of identity. Each certificate file must have the .cer extension and the name given to each identity in the grid will be the file name minus that extension. For example, if I have a certificate in my certs/users directory with a filename of morgan.cer, a user whose grid name is morgan will be created. This certs directory must exist under the current working directory from which the bootstrap script is run.

Finally, the bootstrap script also assumes that a file called machines exists under the current working directory which contains a list of all non-bootstrap machines to be included in the new grid (each machine must already have a container running on it ready to accept incoming messages). This file is formatted with exactly one machine per line indicated as a URL. By default, the URL should use the https protocol and should be for a machine on port 18080. These values can change depending on configuration changes made by the administrators of your grid. The example file for our two host system would contain exactly one line with the following URL:

```
https://host2.domain.name:18080
```

During the bootstrap process you will be asked to enter the password for the login key store, and then to select the appropriate identity to use for bootstrapping the net. If you used the default security configuration, this password will be keys, and the identity will include the skynet credential. If you created your own administrator account, you will need to indicate the appropriate password and identity when asked to log in.

4 Command Reference

GenesisII includes a command line interface to perform virtually all functions. The vast majority of the functionality is thru the `grid` script - grid.bat for Windows platforms; grid.sh for Linux platforms – which launches the Java-based grid command-line tool. The `grid` tool contains a large number of sub commands to carry out most grid functions. With the exception of initial system startup, the `grid` command is likely to be the sole tool required to use and configure your grid system. Section 4.2 discusses all of the sub-commands available and their syntax.

There are several other command-line tools included with GenesisII. The runContainer script (.bat and .sh) starts/restarts the GenesisII Java container on the local machine.

4.1 RunContainer Script

Genesis II ships with a script called runContainer (.sh bash shell script on Unix, .bat batch file in Windows). This script will start up a web service container and initialize the deployed web services. If the script is run and the database for Genesis II has never before been initialized, runContainer will perform a first time initialization.

The container can be configured to use different ports and certificates based on settings in the configuration files (describe in part above in the security and bootstrap set up section). Further, setting the environment variable GENII_USER_DIR, when set, overrides the default state directory used by the container.

4.2 Grid Command

The `grid` tool contains a large number of sub commands to carry out most grid functions. With the exception of initial system startup, the `grid` command is likely to be the sole tool required to use and configure your grid system.

4.2.1 Requirements

Since the script launches the grid Java command line tool, so it is required that you have a proper version of Java installed on your machine and which is in your search path. For details on finding, installing and configuring a proper version of Java see XXX.

4.2.2 Command Modes

The `grid` command can run sub-commands in 2 different ways. The first method is to run the `grid` command to execute a single sub-command. This is done by executing the grid.bat or grid.sh script with the appropriate arguments. Each command run in this manner will start a Java Virtual Machine (JVM), execute the command, and then terminate the JVM. This is straightforward, but involved a significant amount of overhead – starting and stopping the JVM for each command takes on the order of 1 second and consumes a fair amount of machine resources. To make things a bit more efficient, the current GenesisII release includes a very primitive shell command.

Executing the `grid shell` command starts a JVM and a command execution console for entering commands to run. Commands executed within the console reuse the same running JVM, amortizing the overhead for JVM start/stop over all commands executed. It is important to note that the syntax for running commands within the shell is abbreviated for user convenience – the "grid" command no longer needs to be specified. For example, the "`grid ls \`" command is run within the shell simply as "`ls \`".

Note that the current implementation of the grid shell blocks access to the local system versions of command with the same name. For example, when running the grid shell, "ls" will run the `grid ls` command regardless of whether the local system version of `ls` is in the user's path or not. We plan to make the shell smarter in the future to make it easier to use both versions simultaneously.

4.2.3 Command List

- attach-container
- authz
- cat
- cd
- connect
- cp
- create-resource
- export
- ftpd
- gaml-chmod
- get-attributes
- help
- ln
- login
- logout
- ls
- mkdir
- pwd
- rm
- run
- schedule-termination
- script
- shell
- unlink
- whoami

4.2.4 Attach-Container

Syntax:

```
attach-container <genII-container-url> <rns-path>
```

Command Description:

Links the specified Genesis II container into the given hosting environment.

Arguments, Options and Flags:

`<genII-container-url>:`	The URL of the Genesis II container service to link into the grid directory namespace. For example, the default URL for a container running on a *host.domain.name* would be "https://host.domain.name:18080/axis/services/VCGRContainerPortType".
`<rns-path>:`	The directory path to which the target Genesis II container should be attached.

Discussion:

Genesis II containers run as independent services. When created/started they are not assigned or attached to any Genesis II grid system. It is useful to organize related grid services by connecting them into a common namespace to create a larger connected grid system. The `attach-container` command links an existing Genesis II container into a grid directory namespace – therefore it is very similar to the `ln` command.

Version Information:

From 0.6.0 (alpha) to present.

Likely to be merged into `ln` command in the near future and made obsolete.

Comment [jfk1]: Under development

4.2.5 Authz

Syntax:

```
authz <target>
```

Command Description:

Configures authorization for the specified target.

Arguments, Options and Flags:

`<target>:` ???

Discussion:

????

Version Information:

From 0.6.0 (alpha) to present.

4.2.6 Cat

Syntax:

```
cat <target-file0> ...
```

Command Description:

Displays contents of the target files to the stdout stream (ala the Unix cat command).

Arguments, Options and Flags:

`<target-file0>:` Grid directory path name to resource that supports the ByteIO interface.

Discussion:

Cat outputs the "contents" of any grid resource that supports the ByteIO interface to stdout. For resources that are actually files, cat works as you'd expect – the bytes that make up the file are streamed to the stdout device character by character. However, GenesisII has been designed such that many other types of resources also implement the ByteIO interface, in much the same way that more modern Unix systems put a file and directory interface on operating system information in the /proc file system. This makes it easier for users to get information via common, familiar tools. For example, the BES Activity port type returns activity status information via the ByteIO interface.

Version Information:

From 0.6.0 (alpha) to present.

4.2.7 Cd

Syntax:

```
cd <target-dir>
```

Command Description:

Changes the current directory to the one indicated.

Arguments, Options and Flags:

`<target-dir>:`	Grid directory path to new grid current working directory.

Discussion:

Changes the user's current working directory to the specified grid path.

Version Information:

From 0.6.0 (alpha) to present.

4.2.8 Connect

Syntax:

```
connect <connect-url>
```

Command Description:

Connects to an existing net.

Arguments, Options and Flags:

`<connect-url>:` URL of the root RNS information file for the target grid system to connect to.

Discussion:

This command sets up a user's environment to point to an existing grid's root RNS namespace. Once the user's environment is setup, subsequent commands will work within the specified namespace. The URL specified is of the root RNS information file for the target grid system to connect to. This file is created during the bootstrapping process for the grid system and must manually be placed on a web server by a grid system administrator. Therefore, the URL of this file depends on the preferences of the grid administrator. Ask the grid system administrator for the proper URL for the system you want to access.

Version Information:

From 0.6.0 (alpha) to present.

4.2.9 Cp

Syntax:

```
cp [--local-src] [--local-dest] <source-path> <target-path>
```

Command Description:

Copies files around RNS space.

Arguments, Options and Flags:

`<source-path>:`	Path to the source file for the copy. If the `–local-src` flag is used, it is the local path to a file within the user's local file system. If the `–local-src` flag is omitted, it is the grid directory path to a resource that implements the ByteIO interface.
`<target-path>:`	Path to the destination for the file copy. If the `–local-dest` flag is used, it is a local path within the user's local file system. If the `–local-dest` flag is omitted, it is a grid directory path. If the target path already exists, the copy is aborted.
`--local-src:`	Optional flag to indicate that source path is to a file within the local file system namespace rather than the grid directory namespace. If not specified, the `<source-path>` argument indicates a grids namespace path.
`--local-dest:`	Optional flag to indicate that destination path is to a file within the local file system namespace rather than the grid directory namespace. If not specified, the `<target-path>` argument indicates a grids namespace path.

Discussion:

Cp, like its Unix analog, copies a file from one location to another. One difference from the Unix command is that the Genesis II `cp` command allows both the source and the destination to be either a grid directory path or a path in the user's local file system. Another difference is that the Genesis II command allows the source to be any grid resource that implements the ByteIO interface, not just resources that are traditional files.

Version Information:

From 0.6.0 (alpha) to present.

4.2.10 Create-Resource

Syntax:

```
create-resource [--rns] <service-path> [new-rns-path]
   OR
create-resource --url <service-url> [new-rns-path]
```

Command Description:

Creates a new resource using generic creation mechanisms.

Arguments, Options and Flags:

`<service-path>:`	An RNS or grid path indicating a service from which you wish to create a new resource.
`<new-rns-path>:`	An RNS path which you wish to link to the newly created resource (effectively, the path by which you wish for your new resource to be referred to).
`--rns:`	An optional flag indicating that the first command line argument specifies an RNS path for a service rather than a service URL. This is the default though and may be ignored if desired.
`--url`:	An optional flag indicating that the first command line argument specifies the URL of a service from which to create a resource. Without this flag, the default assumption is that the first argument specifies an RNS path.

Discussion:

Creates a new resource using generic creation mechanisms. While some resource port types have preferred creation mechanisms (such as cp for files and mkdir for grid directories), all resource port types in Genesis II support a more generic creation mechanism which can be accessed via the create-resource tool. Using this tool, you can create a new resource from any Genesis II service that you can address (either via RNS or a URL) and assign a grid path to the newly created resource. This mechanism is used internal to create BES resources as well as scheduler resources.

Version Information:

From 0.6.0 (alpha) to present.

4.2.11 Export

Syntax:

```
    --create { --url <export-service-url> | <export-
  service-rns-path> } <local-path> [new-rns-path]
       OR
    export --quit { --url <export-root-url> | <export-root-
  rns-path> }
```

Command Description:

Creates a new exported root directory or quits an existing one.

Arguments, Options and Flags:

`--url <export-service-url>:`	URL for the service on the target container that implements the ExportedRootPortType (which acts as the factory for exported directories). By default this URL will often look like https://*host.domain.name*:18080/axis/services/ExportedRootPortType.
`<export-service-rns-path>:`	Grid namespace path to the service on the target container that implements the ExportedRootPortType. The name of the Genesis II container depends on how you've configured your system. For example, the Genesis II container on the bootstrap machine is given the name /containers/BootstrapContainer. The container may also be named via the attach-container command. The path to the ExportedRootPortType service is given as <container-path>/Services/ExportedRootPortType.
`<local-path>:`	Local file-system path on target Genesis II container for the directory to be exported. If the path does not exist or is not a directory, the export command will fail.
`new-rns-path:`	Path within grid namespace where the exported directory should be mounted.
`--url <export-root-url>:`	URL to an existing exported directory. Unlikely to be used and

	option will likely be removed in future release.
`<export-root-rns-path>:`	Grid namespace path to exported directory (i.e. the mount point name given in the create command. Used to identify the exported directory in the quit command.
`--quit:`	Indicates the quit sub-command. The `export –quit` command stops exporting the local directory pointed to by the target argument without deleting the underlying directories and files being exported (i.e. the resources on the local file system). This differs from using the rm command to remove the directory. Using the rm command on the exported directory or its contents deletes the underlying file system resource.
`--create:`	Indicates the create subcommand. The create command creates a new exported directory and links it into the grid namespace.

Discussion:

Genesis II supports the idea of attaching an entire local directory structure into a grid namespace, creating what we call an exported directory. From the viewpoint of the grid, the exporting a directory is like mounting it into the grid by naming a mount point into the grid directory structure. From within the grid the contents of the exported directory – both its directories and files, recursively – are mapped into the grid as RNS and ByteIO resources and are accessible to grid users subject to access control policy. It is important to note that the directory structure is not copied into the grid namespace. Rather both the local file system and the grid share the same underlying files and directories. So, changes made via the grid are reflected in the local file system and changes made via the local file system are reflected to the grid. This can be a very useful tool when a user wants to share data without having to continuously monitor consistency between copies.

To export a directory, you must first have a Genesis II container running on a machine that can access the target local directory (either on the same machine or via a remote file system like NFS). If no container already exists, use the `runContainer` command to start a container. Then, use the `export` command to mount the local directory into the grid namespace. The export command needs two pieces of information: which service to use to perform the export (either a URL or a grid directory path) and where to mount the exported directory within the grid directory namespace.

Example: To export the local directory "C:\MyFavoriteDir" from the machine host1.domain.name and mount it to the grid as /users/fred/myExportedDir, dot he following.

First, start a Genesis II container on host1.domain.name (if one is not already started).

Second, give the Genesis II container a name in the grid directory namespace – say /containers/host1.domain.name.

Finally, export the directory:

```
export /containers/host1.domain.name
/users/fred/myExportedDir
```

This will create a new grid directory entry /users/fred/myExportedDir which can be used to access the contents of C:\MyFavoriteDir. The directory C:\MyFavoriteDir\Dir1 would show up as /users/fred/myExportedDir/Dir1 and the file C:\MyFavoriteDir\File1 would show up as /users/fred/myExportedDir/File1 and so on recursively.

Version Information:

From 0.6.0 (alpha) to present.

4.2.12 Ftpd

Syntax:

```
ftpd [options] <port> [network-constraints ...]

Where options are:
     --idle-timeout=<secs>
     --data-connection-timeout=<secs>
     --max-auth-attempts=<number>
     --sandbox=<path>
and where network-constraints are:
     one or more dotted decimal quadruples (of the form
     ###.###.###.###)
```

Command Description:

Runs an FTP daemon on the given port.

Arguments, Options and Flags:

`<port>:`	The port on which the ftp daemon will listen.
`--idle-timeout=<secs>:`	The number of seconds that a connected ftp session will last without activity before the server closes it. The default value is 150 seconds.
`--data-connection-timeout=<secs>:`	The number of seconds that an unconnected data connection will last before the server closes it automatically. The default value for this option is 60 seconds.
`--max-auth-attempts=<number>:`	The maximum number of failed authorization attempts that will be allowed on any given FTP session before the session is closed. The default value is 10.
`--sandbox=<path>:`	This optional value allows the user to restrict the directories under which connected sessions will be allowed to browse. For example, a value of /home/mmm2a would restrict all ftp access to directories and entries in the grid located

	in or under /home/mmm2a. Note that this option does not prevent the user from following links outside of this path already set up in the grid.
`network-`*constraints*:	One or more dotted decimal quadruples indicating network ranges to accept connections from. For any given decimal number, a 0 indicates that any value is allowed while a non-zero value restricts connections to that value. For example, the value 128.143.0.0 would allow connections from any IP address that started with 128.143.

Discussion:

This command starts up a simple FTP server which will translate standard FTP client requests into grid requests. This tool is largely a simple tool for simple grid manipulation and demonstration. More advanced tools are forthcoming and we anticipate this command slowly being phased out.

Version Information:

From 0.6.0 (alpha) to present.

Comment [jfk2]: Under development

4.2.13 Gaml-Chmod

Syntax:

```
    gaml-chmod <target> ( <[<+|->r][<+|->w][<+|->x]> |
  <octal mode> ) ( [--local-src] <cert-file> | --everyone )
```

Command Description:

Sets read/write/execute GAML authZ permissions for a target.

Arguments, Options and Flags:

`<target>:`	???
`<{+|-}r>:`	???
`<{+|-}w>:`	???
`<{+|-}x>:`	???
`<octal mode>:`	???
`<cert-file>:`	???
`--local-src:`	???
`--everyone:`	???

Discussion:

????

Version Information:

From 0.6.0 (alpha) to present.

4.2.14 Get-Attributes

Syntax:

```
get-attributes <target>
```

Command Description:

Retrieves and prints the attribute document for a target.

Arguments, Options and Flags:

`<target>:`	The target resource in the grid naming space for which the user wishes to retrieve an attributes document.

Discussion:

Attributes are XML elements stored in a large XML document that describe both static and dynamic information about a given resource. All resources in Genesis II have attributes documents associated with them. This tool retrieves that document and prints it to the stdout stream. Note however that the output is XML and not necessarily human readable. Further, no attempt at formatting or pretty-printing is made by this tool. If the user wishes to view this document he or she is advised to store the resultant document in an XML file and to then view that file with an appropriate XML viewer.

Version Information:

From 0.6.0 (alpha) to present.

4.2.15 Help

Syntax:

```
help [<tool-0>] .. [<tool-n>]
```

Command Description:

Prints information about each tool and its usage.

Arguments, Options and Flags:

`<tool-n>:`	Name of a `grid` tool (aka sub-command). Multiple may be specified in which case the help messages for each tool are listed in order. If no tool name is specified, help prints the list of tools.

Discussion:

Help displays information about the commands supported by the `grid` tool. Multiple may be specified in which case the help messages for each tool are listed in order. If no tool name is specified, help prints the list of tools.

Version Information:

From 0.6.0 (alpha) to present.

4.2.16 Login

Syntax:

```
    login [keystore file] [--no-gui] [--
storetype=PKCS12|JKS] [<keystore password>]
```

Command Description:

Inserts GAML authentication information into the user's context (effectively logging them into the grid using an X.509 certificate as an identity).

Arguments, Options and Flags:

`<keystore file>:`	An optional parameter indicating a java key store file to use for retrieving certificates and private keys. On windows this parameter can be left out if you only intend to log in from the windows certificate store.
`<keystore password>:`	This value is used as the key store parameter. This form of the login command should be used with caution as the key will be publicly displayed.
`--text:`	By default the login tool attempts to use graphical dialogs for the log in process and fails over to text based interaction only if a graphical display is not possible. However, this optional flag forces the log in tool to use a text based interaction model (useful for debugging sometimes and scripts).
`--storetype:`	An optional value indicating whether or not the target key store file (if specified) is a JKS or PKCS12 key store. By default login in assumes PKCS12. Note that this option has no affect on Windows certificate stores.

Discussion:

The login tool allows a user to select a certificate and private key from one or more key stores. After logging in, a list of available certificate aliases will be displayed and the user will have the opportunity to select one of them to take on as his or her identity for the remainder of that session (or until the session times out after an hour). At first glance users get a little confused by the fact that they sometimes do not need to enter any passwords for identities managed by the Windows certificate store. However, because Windows automatically manages this store and protects the certificates in it based on the users log in session at that console, a password is not needed in this case.

Version Information:

From 0.6.0 (alpha) to present.

4.2.17 Ln

Syntax:

```
ln <source-path> <target-path>
   OR
ln --epr-file=<epr-file> <target-path>
   OR
ln --service-url=<url> <target-path>
```

Command Description:

Links EPRs into grid directory namespace.

Arguments, Options and Flags:

`<source-path>:`	Grid directory path to a grid resource or service. The path is used to lookup the EPR for the specified resource or service.
`<target-path>:`	Grid directory path where EPR should be linked. If the specified `<target-path>` already exists, the `ln` command fails.
`--epr-file=<epr-file>:`	Local file system path name to a file that contains an EPR in XML format. This is a useful way to collect an EPR from a command output and later use to connect the resource into the grid directory namespace.
`--service-url=<url>:`	URL for a Genesis II service. The command uses the URL information to generate an EPR for the service so that it can be linked into the grid directory namespace.

Discussion:

The `ln` command works much like the Unix `ln` command – it creates a link from a grid resource or grid service to a name in the grid directory structure. Any well-formed EPR can be linked into the Genesis II grid directory namespace.

Version Information:

From 0.6.0 (alpha) to present.

4.2.18 Logout

Syntax:

```
logout
```

Command Description:

Logs user out.

Arguments, Options and Flags:

None.

Discussion:

The logout command removes any authentication information from the user's grid context, ending the user’s login session.

Version Information:

From 0.6.0 (alpha) to present.

4.2.19 Ls

Syntax:

```
ls [-ldae] [<target>*]
```

Command Description:

Lists information about elements in the grid directory namespace.

Arguments, Options and Flags:

`<target>:`	Grid directory path to resource to list. If the target resource implements the RNS directory interface, it lists the contents of the directory (unless the –d flag is used). If the target does not implement the RNS directory interface, then information about the resource's entry in RMS is listed.
`-l:`	Long listing. Similar to Unix ls command, it lists additional information about the entry.
`-d:`	List information about the target resource, not its directory entries if the resource is a directory.
`-a:`	List all entries – do not skip entries whose names begin with "." as is done by default.
`-e:`	Additionally, list EPR information for each matching entry.

Comment [jfk3]: Add format information

Discussion:

The `ls` command is the grid directory namespace analog to the `ls` command. `Ls` lists information about entries within the grid directory namespace. For non-directories (those resources that do not implement the RNS interface), it lists information about the specific resource's entry in the grid namespace. For grid directories (any resource that implements the RNS interface), ls lists information about its entries (unless the –d flag is used).

Version Information:

From 0.6.0 (alpha) to present.

4.2.20 mkdir

Syntax:

```
mkdir [-p] [--rns-service=<service-path>] <target-dir>+
```

Command Description:

Creates the grid directory(s) indicated.

Arguments, Options and Flags:

`<target-dir>:`	One or more path names in grid namespace for the new directories. If the `target-dir` already exists in the grid namespace, the command fails.
`-p:`	Optional flag indicating that all missing parent directories in the `<target-path>` should be created if they do not already exist.
`--rns-service=<service-path>:`	Describes the grid path name of an RNS creation service to use when creating the new RNS directory. This can be used as a means to determine on which Genesis II Container the new directory resource will be placed. Optional.

Discussion:

If the –rns-service option is omitted, the new RNS directories are created on the same RNS service as that which owns the parent of the new grid directory.

Version Information:

From 0.6.0 (alpha) to present.

4.2.21 Pwd

Syntax:

pwd

Command Description:

Prints the current directory.

Arguments, Options and Flags:

None.

Discussion:

Prints the current working grid directory for the user. The current working directory defaults to "/" and can be set using the cd command.

Version Information:

From 0.6.0 (alpha) to present.

4.2.22 Rm

Syntax:

```
rm [-rf] <target-path> ...
```

Command Description:

Destroys the target resource and removes given link from the grid directory namespace.

Arguments, Options and Flags:

`<target-path>:`	Grid directory path to resource that is to be destroyed.
`-r:`	Remove all entries recursively. `<target-path>` must be a directory for this option.
`-f:`	Force removal of resource from grid directory namespace, even if the operation to destroy the underlying resource fails.

Discussion:

The `rm` command destroys the target resource and removes given link from grid directory namespace. If you not wish to destroy the underlying resource, you should use the unlink command instead.

Version Information:

From 0.6.0 (alpha) to present.

4.2.23 Run

Syntax:

```
   run --name=<job-name> [--async-name=<rns-path>] {<bes-
container> | <scheduler>} [run-options] <executable>
[program-args]
      OR
   run [--name=<job-name>] [--async-name=<rns-path>] --
jsdl=<jsdl-file> {<bes-container> | <scheduler>}
      OR
   run --check-status <job-path>
```

Command Description:

Runs, or checks the status of, a job (also known as an activity) on a BES container (or through a scheduler).

Arguments, Options and Flags:

`--name=<job-name>:`	Give the job the specified name. This name is used to give a meaningful name to a job that a user can browse to inside of a BES container. Because BES containers implement an RNS interface by default, users can later come back and browse the container as if it were a grid directory. Doing so will show the names of all jobs currently managed by that container.
`--async-name=<rns-path>:`	Using this option indicates that the run command is to terminate after the job is submitted to the grid, i.e. the job is to run asynchronously from the run command that started it. <rns-path> indicates the grid directory path that should be given to the activity resource created by the BES container for this job. This resource can later be used to obtain status information and cancel the job.
`<bes-container>:`	Specifies grid path to BES container that should run the job. Mutually exclusive with `<scheduler>` argument.
`<scheduler>:`	Specifies grid path to a scheduler resource that should be used determine job's placement. Mutually exclusive with `<bes-container>` argument.
`<executable>:`	Specifies the executable path to run on the remote host. This path MUST exist on the remote machine after all of the input data stages have completed.

```
    --check-status <jobpath>:  Sub command to check the status of an
                               asynchronously run job.  The <jobpath>
                               argument points to the grid directory path
                               that indicates the activity in question (as
                               specified by the <async-name> argument in
                               the job's initial run command).

run-options:
    --stdout=<out-filename>:   The name of a file to redirect stdout from
                               the running activity to.  The presence of this
                               option does not necessarily imply that that
                               output will be collected at the end of the run.
                               If you wish to specify that you must
                               additionally give an appropriate --stage-out
                               option.
    --stderr=<err-filename>:   The name of a file to redirect stderr from the
                               running activity to. The presence of this
                               option does not necessarily imply that that
                               output will be collected at the end of the run.
                               If you wish to specify that you must
                               additionally give an appropriate --stage-out
                               option.
    --stdin=<in-filename>:     The name of a file from which to redirect
                               stdin to the running activity.  The presence
                               of this option does not necessarily imply that
                               the indicated file will be available on the
                               remote machine.  The file must exist after
                               the stage in portion of the job has
                               completed.
    --stage-in=<filename/stage-in-uri>:
                               An optional value that indicates the uri from
                               which to stage in a piece of data before the
                               activity runs and the name that that staged in
                               data should be given in the local file system.
                               This option may occur more than once
                               indicating multiple stage in requests.
    --stage-out=<filename/stage-out-url>:
                               An optional value that indicates the uri to
                               which to stage out a piece of data after the
                               activity runs and the name of the file in the
                               local file system from which to retrieve that
                               data.  This option may occur more than once
                               indicating multiple stage out requests.
    --D<env-var>=<env-value>:
                               An optional value indicating an environment
                               variable that should be set when the activity
                               runs.  This option may occur more than once
```

	on a given command line indicating multiple environment variables.
`program-args:`	These are arguments passed directly to the job (via the program's command line arguments).

Discussion:

This tool both runs or executes legacy jobs on remote BES containers and also checks the status of those jobs. For clarity, we describe three distinct usages of this command separately.

When run with the --jsdl option (which specifies a local XML file assumed to contain a valid JSDL document), this tool creates a new activity on a remote BES container (either directly, or by way of a scheduler) using that JSDL file as a description of the activity that the user wishes to execute. In this form, the tool performs little to know sanity checking beyond simple XML validation of the JSDL document and existence checking for BES/Scheduler targets.

When this tool is run without the --check-status flag and without the --jsdl option, it assumes that the remainder of the command line will describe the contents of a JSDL file which it will build on the fly. This permits the user to easily script or create activities without necessarily understanding or creating manually a valid JSDL document. However, the user is still responsible for conveying enough information from which to create the valid JSDL document. For example, data that is staged in or out has the format of <filename/stage-uri> allowing for a valid JSDL document that describes both the name of the file and the URI from which/to which to stage it. For example, when used as a stage in argument, one could stage data from an HTTP server to the file foo.dat by using a value like foo.dat/http://tempuri.org/somepath/somefile.dat, etc.

Finally, the run tool can also check the status of an existing BES activity using the --check-status flag. In this scenario, the user indicates the RNS path of a BES activity from which to acquire status information. This RNS path can either be the one offered by the RNS interface of the BES container on which it is running, or a stored path somewhere else.

Version Information:

From 0.6.0 (alpha) to present.

4.2.24 Schedule-Termination

Syntax:

```
schedule-termination <target>+ <calendar-value>
```

Command Description:

Sets the termination time for a given grid resource.

Arguments, Options and Flags:

`<target>:`	Grid directory path to target resource. May specify multiple targets by separating arguments with spaces.
`<calendar-value>:`	Specification of when resource should terminate. There are two possible formats. The "date" format specifies the absolute date and time when then the resource should terminate and must be a Java parseable date. The exact format that this string can take depends on your Java configuration and locale. For more information, please refer to the Java API documentation for java.text.DateFormat (located at http://java.sun.com/j2se/1.5.0/docs/api/java/text/DateFormat.html). The "delta" format specifies a time relative to now when the resource should terminate and has the form: `+<positive integer> { ms|s|m|h|d }.` The meaning is to terminate the resource in `<positive integer>` millseconds, seconds, minutes, hours or days.

Discussion:

All grid resources in Genesis II can optionally have a pre-defined maximum lifetime - the concept, if not the exact interface come from OGSI (Open Grid Services Infrastructure) and WSRF (Web-Services Resource Framework) specifications. The schedule-termination command allows users/administrators to set the maximum lifetime for a resource (subject to permission). When a resource reaches the end of its specified lifetime, it is automatically destroyed by the Genesis II container. The scheduled-termination time does not in any way promise that the resource will remain around for that long – the usual methods for destroying a resource are still applicable (again, subject to permission).

Version Information:

From 0.6.0 (alpha) to present.

4.2.25 Script

Syntax:

```
script <script-file>
```

Command Description:

Runs a Genesis II script.

Arguments, Options and Flags:

`<script-file>:`	An XML script file describing the actions to perform. Refer to the Discussion below for a detailed description of this script file syntax.

Discussion:

The script tool executes a simple, XML-based, scripting language that permits users to write grid scripts that take advantage of amortized startup costs for the JVM and for various certificate keys, etc. The syntax of this file allows for a few rudimentary control flow operations to be performed as well as permitting the execution of any of the registered grid tools. These two categories of element in the script file manifest in two different XML namespaces used within the scripting document. The first, http://vcgr.cs.virginia.edu/genii/xsh/script (**gsh**), prefixes XML elements that indicate scripting constructs such as flow control statements, variables, etc. The latter, http://vcgr.cs.virginia.edu/genii/xsh/grid (**geniix**), indicates an XML element referring to a grid tool. Throughout the scripting, users may place variables within attribute values and text elements for almost all scripting constructs. For example, to echo a users name to the console (and assuming that a variable called USER exists), one might use an XML scripting elements like:

```
<gsh:echo message="The user is ${USER}"/>
```

Note that the variable is enclosed in `${}` markers.

For scripting constructs, a small number of elements exist including **echo**, **define**, **foreach**, and **param**. Each is described in turn in the following paragraphs.

The **echo** scripting construct allows a user to echo text to the standard output stream for his or her console. Echo must be an empty element and must include exactly one attribute called **message** that contains the text to be echoed to the display.

The **define** script element allows for new variables to be defined based off of other text values. Additionally, users may also specify that the value should be passed through a sed-like pattern replacement prior to its assignment to the variable. The **define** element is an empty element and must contain exactly one attribute called **name** which will indicate the name of the variable, and another attribute called **source** which indicates the value that the variable should take on. Further, users may optionally include a **pattern** and a **replacement** attribute indicating that a regular expression pattern should be replaced with another value within the variable's content. Finally, if the user is doing pattern replacement, he or she may indicate that the replacement should be global (i.e., applied to the entire value) or not (only applied once, the first time that the pattern is found) by using the **global** attribute. The following example should all of these elements in use.

```
   <gsh:define name="GREETING" source="Hello ${USER}!"
pattern="^Hello" replacement="Goodbye" global="false"/>
```

The **foreach** scripting element allows the user to create a for each loop within their scripts. Currently, only a limited set of iteration sets are supported. Explicitly, a user can loop over entries in a local or grid directory, and over lines in a text file. The **foreach** construct also supports a filtering mechanism. Every **foreach** element must contain exactly one attribute called **param-name** that indicates the variable name to use for the loop index during execution. Each loop must also include exactly one of the **source-dir**, **source-rns**, or **source-file** attributes indicating respectively a local directory, grid directory, or text file to loop over. Finally, each **foreach** loop may optionally contain an attribute called **filter** that indicates a regular expression filter that will be applied to the loop. An example **foreach** loop is given below:

```
<gsh:foreach param-name="FILE" source-rns="/bes-containers">
    <gsh:echo message="Grid directory contains entry ${FILE}"/>
</gsh:foreach>
```

The **param** scripting construct is used only within grid tool invocation elements to indicate a parameter to a grid tool. This element has no attributes and its content must be the text parameter to pass to the encompassing grid tool.

Finally, any XML element within the scripting language that resides in the geniix namespace is assumed to represent a Genesis II grid tool. There is no predefined set of tools supported but rather at run time the scripting language will look up each tool in turn based on its XML element name. If the tool is supported by the grid command, then the scripting language will support it. Parameters are given as instances of the **gsh:param** construct contained as child elements of the tool element.

For examples of all of these scripting elements, please refer to the **${GENESIS_II}**/bootstrap.xml file included with your Genesis II distribution.

Version Information:

From 0.6.0 (alpha) to present.

Comment [jfk4]: Under development

4.2.26 Shell

Syntax:

```
shell
```

Command Description:

Starts GenesisII interactive shell.

Arguments, Options and Flags:

None.

Discussion:

????

Version Information:

From 0.6.0 (alpha) to present.

4.2.27 Unlink

Syntax:

```
unlink <target-path> ...
```

Command Description:

Unlinks (without destroying) the target paths.

Arguments, Options and Flags:

`<target-path>:` Path to grid directory entry that should be removed from the namespace. Multiple targets can be specified in one command by separating the path names with spaces.

Discussion:

The `unlink` command removes a given path from the grid directory namespace, similar to the Unix `unlink` command. This command does not change or destroy the resource or service pointed to by the grid namespace entry. If you want to destroy the underlying resource or service, the `rm` command should be used instead.

Version Information:

From 0.6.0 (alpha) to present.

Comment [jfk5]: Under development

4.2.28 Whoami

Syntax:

```
whoami
```

Command Description:

Prints user's current authentication information.

Arguments, Options and Flags:

None.

Discussion:

????

Version Information:

From 0.6.0 (alpha) to present.

Last Updated: March 2, 2007.