# Table of Contents

A. Welcome to Grid Computing ................................................................. 12

B. Document Overview ........................................................................... 13
   B.1. Intended Readers ........................................................................ 13
   B.2. Author List ................................................................................. 14
   B.3. Document Notation ...................................................................... 14
       B.4.1. HOME .................................................................................. 15
       B.4.2. TMP ..................................................................................... 15
       B.4.3. JAVA_HOME ........................................................................... 15
       B.4.4. GENII_INSTALL_DIR ................................................................. 15
       B.4.5. XSEDE_TEST_ROOT ................................................................. 15
       B.4.6. GENII_USER_DIR ................................................................. 15
       B.4.7. The “grid” Command ............................................................. 16
   B.5. Glossary ....................................................................................... 16

C. Introductory Topics ............................................................................ 20
   C.1. Learning the Grid Metaphor .......................................................... 20
   C.2. System Administrator Tutorial .................................................... 20
   C.3. Public Key Infrastructure (PKI) Tutorial ......................................... 20
   C.4. Documentation Updates ............................................................... 20

D. Genesis II Installation ........................................................................ 22
   D.1. Installer User Considerations ......................................................... 22
   D.2. Installing the Grid Client Using the Graphical Installer .................. 22
   D.3. Installing the Grid Client at the Command-Line ............................. 25
   D.4. Installing a Grid Container Using the Graphical Installer ............... 26
       D.4.1. OpenSSL Conversion ............................................................... 29
   D.5. Installing a Grid Container at the Command-Line .......................... 30
   D.6. Automatic Container Start-up ......................................................... 31
       D.6.1. System-wide Installation Start-up ............................................. 31
       D.6.2. Personal Installation Start-up .................................................... 31
   D.7. Installation on Linux Using an RPM or DEB Package ...................... 32
       D.7.1. RPM and DEB Upgrade Caution .............................................. 33
   D.8. Unified Configuration for Containers ........................................... 33
       D.8.1. Creating a New Container That Uses a Unified Configuration ...... 33
       D.8.2. Converting a Container in Split Configuration Mode to a Unified Configuration 35
D.8.3. Changing Container’s Installation Source for Unified Configuration .......................... 36
D.8.4. Updating Container’s Grid Deployment for Unified Configuration .......................... 36
D.8.5. Using a Grid Deployment Override ...................................................................... 37
D.8.6. Unified Configuration Structure .......................................................................... 37
D.8.7. Converting a Source-Based Container to a Unified Configuration ....................... 39
E. Grid Usage Topics ........................................................................................................ 40
E.1. Built-in Help ............................................................................................................... 40
E.2. Authentication and Authorization .............................................................................. 40
   E.2.1. Credentials Wallet .............................................................................................. 41
   E.2.2. How to Login & Logout ...................................................................................... 41
   E.2.3. Grid Access Control Lists (ACLs) ....................................................................... 44
E.3. Data Files ................................................................................................................... 46
   E.3.1. Copying Data Into and Out of the GFFS .............................................................. 46
   E.3.2. Exporting Local Filesystems to the Grid .............................................................. 50
   E.3.3. How to Mount the GFFS via a FUSE Filesystem ................................................. 50
   E.3.4. Other Staging Methods for Data Files ................................................................. 52
E.4. Grid Commands ......................................................................................................... 52
   E.4.1. Grid Command Set ............................................................................................. 53
   E.4.2. Grid Paths: Local vs. RNS .................................................................................. 53
   E.4.3. Scripting the Grid Client ..................................................................................... 54
   E.4.4. XScript Command Files ....................................................................................... 55
E.5. Submitting Jobs .......................................................................................................... 59
   E.5.1. How to Create a JSDL file ................................................................................... 59
   E.5.2. Using the Job-Tool .............................................................................................. 59
   E.5.3. Submitting a Job to a Grid Queue ........................................................................ 61
   E.5.4. Controlling or Canceling Jobs in a Queue ............................................................ 61
   E.5.5. Cleaning Up Finished Jobs .................................................................................. 62
   E.5.6. The Queue Manager in Client-UI ........................................................................ 62
   E.5.7. Job Submission Point .......................................................................................... 64
   E.5.8. Submitting a Job Directly to a BES ................................................................... 64
   E.5.9. How to Run an MPI Job ....................................................................................... 65
E.6. Client GUI ................................................................................................................... 66
   E.6.1. Client GUI Basics ............................................................................................... 66
   E.6.2. Credential Management ....................................................................................... 67
   E.6.3. Client UI Panels and Menus ................................................................................ 70
   E.6.4. Drag-and-Drop Feature ....................................................................................... 89
E.6.5. File associations

F. Grid Configuration

F.1. Structure of the GFFS

F.2. Deployment of the GFFS

F.2.1. Preparing the Environment for Generating Deployments

F.2.2. Creating the GFFS Root Deployment

F.2.3. Changing a Container’s Administrative or Owner Certificate

F.2.4. XSEDE Trust Store Customization

F.2.5. Detailed Deployment Information

F.2.6. Certificate Revocation Management (CRL files)

F.3. Grid Containers

F.3.1. Container Structure

F.3.2. Where Do My Files Really Live?

F.3.3. Serving GFFS Folders from a Specific Container

F.3.4. Container Network Security

F.3.5. Container Resource Identity

F.3.6. User Quota Configuration

F.3.7. Genesis Database Management

F.4. Grid Queues

F.4.1. Creating a Genesis II Queue

F.4.2. Linking a BES as a Queue Resource

F.5. Basic Execution Services (BES)

F.5.1. How to Create a Fork/Exec BES

F.5.2. Running a BES Container With Sudo

F.6. Grid Inter-Operation

F.6.1. How to Create a BES using Construction Properties

F.6.2. Adding a PBS Queue to a Genesis II Queue

F.6.3. Adding a Unicore6 BES to a Genesis II queue

F.6.4. Adding an MPI Cluster to a Grid Queue

F.6.5. Establishing Campus Bridging Configurations

G. Grid Management

G.1. User and Group Management

G.1.1. Creating Grid Users

G.1.2. Creating a Group

G.1.3. Adding a User to a Group

G.1.4. Removing a User from a Group
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>G.1.5.</td>
<td>Removing a User</td>
<td>134</td>
</tr>
<tr>
<td>G.1.6.</td>
<td>Removing a Group</td>
<td>135</td>
</tr>
<tr>
<td>G.1.7.</td>
<td>Changing a User's Password</td>
<td>136</td>
</tr>
<tr>
<td>G.1.8.</td>
<td>Using a Kerberos STS</td>
<td>136</td>
</tr>
<tr>
<td>G.1.9.</td>
<td>Creating XSEDE Compatible Users</td>
<td>137</td>
</tr>
<tr>
<td>G.1.10.</td>
<td>Configuring Kerberos Authorization on a Container</td>
<td>138</td>
</tr>
<tr>
<td>G.1.11.</td>
<td>Setting Up an InCommon STS</td>
<td>140</td>
</tr>
<tr>
<td>G.2.</td>
<td>Container Management</td>
<td>141</td>
</tr>
<tr>
<td>G.2.1.</td>
<td>How to Stop a Grid Container</td>
<td>141</td>
</tr>
<tr>
<td>G.2.2.</td>
<td>How to Start a Grid Container</td>
<td>142</td>
</tr>
<tr>
<td>G.2.3.</td>
<td>How to Backup a Genesis II Grid Container</td>
<td>142</td>
</tr>
<tr>
<td>G.2.4.</td>
<td>How to Restore a Genesis II Grid Container</td>
<td>143</td>
</tr>
<tr>
<td>G.2.5.</td>
<td>Replication of GFFS Assets</td>
<td>145</td>
</tr>
<tr>
<td>G.3.</td>
<td>RNS &amp; ByteIO Caching</td>
<td>151</td>
</tr>
<tr>
<td>G.4.</td>
<td>Grid Accounting</td>
<td>151</td>
</tr>
<tr>
<td>G.4.1.</td>
<td>Accounting Prerequisites</td>
<td>152</td>
</tr>
<tr>
<td>G.4.2.</td>
<td>Background</td>
<td>152</td>
</tr>
<tr>
<td>G.4.3.</td>
<td>Accounting Database</td>
<td>152</td>
</tr>
<tr>
<td>G.4.4.</td>
<td>Denormalized accounting data for usage graphs</td>
<td>153</td>
</tr>
<tr>
<td>G.4.5.</td>
<td>The denormalization process</td>
<td>153</td>
</tr>
<tr>
<td>G.4.6.</td>
<td>Linking the Accounting Database Into the Grid</td>
<td>155</td>
</tr>
<tr>
<td>G.4.7.</td>
<td>Migrating Accounting Info to a New Grid</td>
<td>155</td>
</tr>
<tr>
<td>G.4.8.</td>
<td>Usage Graph Web Site</td>
<td>156</td>
</tr>
<tr>
<td>G.4.9.</td>
<td>Database Table Structure for Accounting</td>
<td>157</td>
</tr>
<tr>
<td>G.4.10.</td>
<td>Creating the Accounting Database</td>
<td>161</td>
</tr>
<tr>
<td>G.5.</td>
<td>Grid Inter-Operation</td>
<td>164</td>
</tr>
<tr>
<td>G.5.1.</td>
<td>Connecting a Foreign Grid</td>
<td>165</td>
</tr>
<tr>
<td>H.</td>
<td>XSEDE Development with Genesis II</td>
<td>167</td>
</tr>
<tr>
<td>H.1.</td>
<td>Installing Java</td>
<td>167</td>
</tr>
<tr>
<td>H.1.1.</td>
<td>Centos Build Dependencies</td>
<td>168</td>
</tr>
<tr>
<td>H.1.2.</td>
<td>Ubuntu Build Dependencies</td>
<td>168</td>
</tr>
<tr>
<td>H.2.</td>
<td>Getting the Genesis II Source Code</td>
<td>168</td>
</tr>
<tr>
<td>H.3.</td>
<td>Building Genesis II from Source on the Command Line</td>
<td>168</td>
</tr>
<tr>
<td>H.4.</td>
<td>Developing Genesis II in Eclipse</td>
<td>169</td>
</tr>
<tr>
<td>H.4.1.</td>
<td>Getting Eclipse</td>
<td>169</td>
</tr>
<tr>
<td>H.4.2.</td>
<td>Getting Subclipse</td>
<td>169</td>
</tr>
</tbody>
</table>
Appendix 1: FAQs and Troubleshooting Guide

J.1. Grid Client Problems ........................................................................................................185
    J.1.1. What does “Internal Genesis II Error -- Null Pointer Exception ” mean? ......185
    J.1.2. Why Can’t I Login With My Valid XSEDE Portal ID? ......................................185
    J.1.3. Why Does Client-UI Get out of Sync? ....................................................................186
    J.1.4. Why Is My Grid Client Logged in Differently than Client-UI? .........................186
    J.1.5. How Can I Deal With Memory Problems? ...............................................................186
    J.1.6. Why Can’t the Installer or Grid Client Connect to the Grid? ..............................187
    J.1.7. What does “Unable to locate calling context information” mean? .................188
J.2. Problems with the Grid Container ....................................................................................189
K.9.4. Supporting Libraries, Jar Files and Other Software ........................................231
   K.10.1. Grid Resource-Specific Storage.................................................................232
K.11. Cross-Campus Grid (XCG) Global Namespace ...........................................234
K.12. Security in the Grid ....................................................................................235
   K.12.1. Supported Security Tokens .................................................................235
   K.12.2. Genesis II User/Group Resources and GAML Tokens .....................235
L. Appendix 3: XScript Language Reference .........................................................236
   L.1. Introduction – What is XScript? .................................................................236
   L.2. Namespaces ..............................................................................................236
   L.3. Running XScript Scripts ..........................................................................237
   L.4. XScript Variables/Macros .......................................................................237
   L.5. XScript High-level Description ...............................................................238
       L.5.1. Grid Command Elements .................................................................238
       L.5.2. XScript Language Elements .............................................................238
M. GFFS Exports Explicated ................................................................................252
   M.1. What is an Export? ..................................................................................252
   M.2. Types of Exports ......................................................................................253
       M.2.1. ACL Exports ......................................................................................253
       M.2.2. ACLAndChown Exports .................................................................253
       M.2.3. ProxyIO Exports ..............................................................................253
   M.3. Users View - Exporting File Systems Directory Trees to the GFFS .........255
       M.3.1. Creating an Export in the GFFS .........................................................255
       M.3.2. Lightweight vs. Heavyweight Exports ...........................................256
       M.3.3. Setting Extended ACLs for ACL and ACLAndChown Exports ....257
       M.3.4. Preferred Identity Management ......................................................258
   M.4. System Administrator Considerations .....................................................260
       M.4.1. Establishing the Container’s Export Configuration .......................260
       M.4.2. ACL Export Mode Configuration .....................................................260
       M.4.3. ACLAndChown Export Mode Configuration ..................................261
       M.4.4. ProxyIO Export Mode Configuration ..............................................261
       M.4.5. Enabling Export Creation for Grid Users .........................................262
       M.4.6. Configuration of the Grid-Mapfile ....................................................263
       M.4.7. Configuration of the “sudoers” File for Sudo Access ......................263
       M.4.8. Creating Exports for Other Grid Users ............................................265
       M.4.9. Configuration Files and Scripts ........................................................266
N. Central Administrator’s Guide for the XSEDE Production Grid .......................................................... 271
   N.1. Installation Support .......................................................................................................................... 271
   N.2. Deployment Perspective .................................................................................................................. 271
   N.3. Creating an XSEDE GFFS Grid of Four Central Containers ......................................................... 271
      N.3.1. Deployment Prerequisites ......................................................................................................... 271
      N.3.2. Time and Effort Estimates ....................................................................................................... 275
      N.3.3. Deploying the XSEDE GFFS Central Containers .................................................................... 275
   N.4. Updating the XSEDE GFFS Production Grid to SDIAC -149 ............................................................ 282
      N.4.1. Update Installations to RPM Packages ................................................................................. 283
      N.4.2. Enable Users to Store Files on Root and Root Replica Containers ...................................... 284
      N.4.3. Add a Pattern-based ACL for MyProxy users ....................................................................... 284
   N.5. XSEDE GFFS Central Container Administrative Procedures ......................................................... 285
      N.5.1. Container Startup/Shutdown ...................................................................................................... 285
      N.5.2. Revert/Undo/Rollback .............................................................................................................. 286
      N.5.3. Container Backup and Restore ................................................................................................. 286
      N.5.4. Creating XSEDE User Accounts ............................................................................................. 287
      N.5.5. Container log files ..................................................................................................................... 287
      N.5.6. User and group management ................................................................................................... 289
      N.5.7. Prepare for Service Provider GFFS Deployment ...................................................................... 290
      N.5.8. Link an External Grid into XSEDE Grid ................................................................................ 291
O. References ............................................................................................................................................. 292
P. Document History ................................................................................................................................. 293
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Installer Welcome Dialog</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>Installation Location Dialog</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Installation Type Dialog</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>Active Copying of Files</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>Concluding Dialog After Installation</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>Installation Type as Container</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>Container Web-Service Configuration</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>Owner Selection Dialog</td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>Grid Keypair Generation Choice</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>Choosing Password for Generated TLS Keypair</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>Specifying Existing Keypair and Password</td>
<td>29</td>
</tr>
<tr>
<td>12</td>
<td>The client UI with RNS Tree and ACL List</td>
<td>48</td>
</tr>
<tr>
<td>13</td>
<td>Directory Context Menu</td>
<td>49</td>
</tr>
<tr>
<td>14</td>
<td>Job tool basic information tab</td>
<td>60</td>
</tr>
<tr>
<td>15</td>
<td>Job tool data staging tab</td>
<td>60</td>
</tr>
<tr>
<td>16</td>
<td>Job tool resources tab</td>
<td>61</td>
</tr>
<tr>
<td>17</td>
<td>Launching the queue manager</td>
<td>62</td>
</tr>
<tr>
<td>18</td>
<td>Queue manager's job list</td>
<td>63</td>
</tr>
<tr>
<td>19</td>
<td>Queue manager's resource tab</td>
<td>63</td>
</tr>
<tr>
<td>20</td>
<td>Removing a job from the queue</td>
<td>63</td>
</tr>
<tr>
<td>21</td>
<td>Job history detail window</td>
<td>64</td>
</tr>
<tr>
<td>22</td>
<td>Setting matching parameters in resources tab</td>
<td>65</td>
</tr>
<tr>
<td>23</td>
<td>XCG3 viewed in client-ui</td>
<td>67</td>
</tr>
<tr>
<td>24</td>
<td>Credential Management-&gt;Login-&gt;Standard Grid User</td>
<td>68</td>
</tr>
<tr>
<td>25</td>
<td>Showing grid credentials using mouse hover</td>
<td>69</td>
</tr>
<tr>
<td>26</td>
<td>Highlighting a specific credential to logout from</td>
<td>70</td>
</tr>
<tr>
<td>27</td>
<td>Major User Interface Panels</td>
<td>71</td>
</tr>
<tr>
<td>28</td>
<td>Drag RNS Resource to Trash</td>
<td>72</td>
</tr>
<tr>
<td>29</td>
<td>Drag-and-drop a user to ACL list on a resource</td>
<td>73</td>
</tr>
<tr>
<td>30</td>
<td>Dragging ACL Entry Into Trash</td>
<td>74</td>
</tr>
<tr>
<td>31</td>
<td>Changing UI Shell font and size</td>
<td>75</td>
</tr>
<tr>
<td>32</td>
<td>Setting UI to show detailed credential information</td>
<td>76</td>
</tr>
<tr>
<td>33</td>
<td>Viewing detailed credential information</td>
<td>77</td>
</tr>
<tr>
<td>34</td>
<td>Displaying resource information as tree structure</td>
<td>78</td>
</tr>
<tr>
<td>35</td>
<td>File-&gt;Create New File option</td>
<td>79</td>
</tr>
<tr>
<td>36</td>
<td>Job tool, creating simple ls job</td>
<td>80</td>
</tr>
<tr>
<td>37</td>
<td>Job-tool showing project number/allocation</td>
<td>81</td>
</tr>
<tr>
<td>38</td>
<td>Job Tool, Data tab showing Output and Error Files</td>
<td>82</td>
</tr>
<tr>
<td>39</td>
<td>Jobs-&gt;Queue Manager</td>
<td>83</td>
</tr>
<tr>
<td>40</td>
<td>Displaying Job History</td>
<td>84</td>
</tr>
<tr>
<td>41</td>
<td>Job Definition Using Variables</td>
<td>85</td>
</tr>
<tr>
<td>42</td>
<td>Variable Usage in Output Filename</td>
<td>86</td>
</tr>
</tbody>
</table>
Figure 43. Defining Job Variable Values ........................................................................................................87
Figure 44. Queue View with Sweep Jobs ........................................................................................................88
Figure 45. Invoking grid shell via Tools->Launch Grid Shell option .............................................................89
Figure 46. User request form .........................................................................................................................156
Figure 47. Example of daily usage graph .......................................................................................................157
Figure 48 Remote Client Interaction with Container ....................................................................................252
A. Welcome to Grid Computing

Welcome to the world of grid computing! This field combines ideas from high performance computing, web services, networking, and computer security in order to provide the user with a powerful, virtual super-computer for running large workloads across heterogeneous systems and clusters. Grid computing can offer a uniform interface that glosses over the multifarious details of the distributed computing systems that are brought together to form the grid. Users can submit their jobs to grid queues that will automatically parcel out the workload to available execution services. User's computing jobs can rely on data located around the world, brought together by the uniform filesystem view provided by the grid.

The Genesis II GFFS (Global Federated File System) is the topic of this reference manual. The GFFS is part of the XSEDE project (http://xsede.org) and provides a globally accessible grid filesystem as well as grid queues that leverage the high performance computing infrastructure of the XSEDE project.
B. Document Overview

B.1. Intended Readers

The main body of the document consists of the following sections:

1. **Installation**
   Describes both graphical and command-line versions of installers for both the grid client and container.

2. **Grid Usage**
   Surveys the basics of authentication and authorization in the grid, running jobs on compute resources, exporting local file system paths to the GFFS, copying data files into and out of the grid.

3. **Configuration**
   Discusses the deployment of the root GFFS container, the deployment of secondary containers, creation of Basic Execution Services to run jobs, creation of grid queues, and establishing campus bridging configurations.

4. **Management**
   Covers creating users and groups in the grid, removing users and groups, stopping and restarting containers, backing up containers, and restoring containers from a backup.

5. **Development**
   Provides links to the Genesis II code repository, describes the command-line build process, and debugging Genesis II source with the Eclipse IDE.

6. **Testing**
   Discusses how to create a small bootstrapped grid for testing, how to run the XSEDE test scripts, and what results to expect from the tests.

7. **Appendices**
   Contains a FAQ & troubleshooting guide. Also provides a detail-oriented reference for extended deployment issues and other configuration considerations.

This document is intended for the following classes of users (also known as personas):

1. XSEDE System Administrators
2. Scientific Users
3. Campus Grid Administrators
4. Grid Testers
5. XSEDE Developers

Membership in a particular user class does not necessarily limit an individual’s interest in any of the information documented here. That said, the Installation and Grid Usage chapters will be
especially relevant to the Scientific User. The Configuration and Management chapters will be of more interest to the XSEDE System Administrators and Campus Grid Administrators. Finally, the Grid Tester and XSEDE Developer personas each have a chapter devoted to their particular viewpoint.

B.2. Author List

This document is a group effort. It incorporates text from many contributors who, over an extended period of time, wrote various documents about the Genesis II grid functionality. These contributors include:

- Bastian Demuth
- Daniel Dougherty
- Ashwin Raghav Mohan Ganesh
- Andrew Grimshaw
- John Karpovich
- Chris Koeritz
- Duane Merrill
- Mark Morgan
- Michael Saravo
- Karolina Sarnowska-Upton
- Salvatore Valente
- Vanamala Venkataswamy
- Muhammad Yanhaona

Editor: Chris Koeritz

This omnibus document was originally accumulated and edited for “XSEDE Activity 43 – Genesis II Documentation” during the spring of 2012. Chris has served as the Omnibus editor for ongoing edits through XSEDE Increment 5.

Special Thanks

Thanks to Jessica Otey for her help in editing this document.

B.3. Document Notation

In this document, command-lines appear as follows:

- The command-line font in this document will be 9 point monospace bold
- The dollar sign followed by a name specifies an environment variable: $VAR
- Curly brackets indicate a parameter that must be filled in: {parmX}
- Pound signs (#, aka octothorpes) indicate comments that accompany code


A few environment variables are used consistently in this document.
B.4.1. HOME

The HOME variable is expected to already exist in the user environment; this points at the home folder for the current user on Linux and Mac OS X. On Windows, the home directory is composed of two variables instead: ${HOMEDRIVE}${HOMEPath}

B.4.2. TMP

The TMP variable should point at a location where temporary files can be stored. If TMP is not set, the tool and test scripts will default to /tmp on Linux and Mac OS X.

B.4.3. JAVA_HOME

The JAVA_HOME variable is used to specify the top-level of the Java JDK or JRE. This variable is not widely used in the Genesis II software but may be used in a few specific scripts. If the “java” executable is found on the application path, then JAVA_HOME is not usually needed.

B.4.4. GENII_INSTALL_DIR

The GENII_INSTALL_DIR variable is a Genesis II specific variable that points at the top folder of the Genesis II software installation. This variable is not needed by the Genesis II Java software, although it may be relied on by some scripts and is used extensively in this document.

B.4.5. XSEDE_TEST_ROOT

The XSEDE_TEST_ROOT variable points at the top-level of the GFFS tool and test scripts within the Genesis II installation package. It is also not needed by the Java software of Genesis II, but will be relied on heavily within the provided tool and test scripts.

B.4.6. GENII_USER_DIR

The GENII_USER_DIR variable points at the path where client and container state are stored. This is also referred to as the "state directory". This variable is used within the Genesis II Java software and by many of the tool and test scripts. The variable is optional in general and will default to "$HOME/.genesisII-2.0". However, if a Genesis II client or container is intended to use a different state directory than the default, then the variable must be defined before the client or container software is started. It is recommended that any non-default value for the variable be set in the user’s script startup file (such as $HOME/.bashrc) to avoid confusion about the intended state directory.

For users on NFS (Network File System), it is very important that container state directories (aka GENII_USER_DIR) are not stored in an NFS mounted folder. Corruption of the container state can result if this caution is disregarded. To avoid the risk of corruption, the GENII_USER_DIR variable can be set to a directory location that is on a local hard disk.
B.4.7. The “grid” Command

Throughout the document, we will often reference the “grid” command from Genesis II. It is shown as just “grid” in example commands, which assumes that the grid command is in the PATH variable. The path can be automatically updated for Genesis II GFFS by running a script included with the install called “set_gffs_vars”. For example, this loads the important Genesis II variables into the current bash environment:

```
source /opt/genesis2-xede/set_gffs_vars
```

This statement assumes an XSEDE production grid installation of the Genesis II GFFS RPM file; other installations may have a different install path. The above statement loads GENII_INSTALL_DIR, XSEDE_TEST_ROOT, and other important variables as well as putting the Genesis II grid command into the PATH. This statement can be added to .bashrc for automatic execution in each bash shell if desired. There are many other methods for getting the grid command into the path, including Environment Module files, or even just adding the environment variables manually.

To add the GFFS grid command manually, one can set the value of $GENII_INSTALL_DIR and add it into the PATH variable:

```
export GENII_INSTALL_DIR=/opt/genesis2-xcg
export PATH=$PATH:$GENII_INSTALL_DIR
```

This command assumes an RPM install for the Cross-Campus Grid (XCG) at the University of Virginia.

B.5. Glossary

These terms will be used throughout the document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Access Control List. A security feature that specifies a set of rights for particular users. Any object stored in the GFFS has three ACLs (one each for read, write, and execute permissions). Each ACL can have zero or more rights in the set.</td>
</tr>
<tr>
<td>BES</td>
<td>Basic Execution Services. The component that offers computational resources to a grid. A BES can accept jobs, run them on some resource, and then provide the job’s results.</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create Read Update Delete. An acronym for the four most common file operations.</td>
</tr>
<tr>
<td>EMS</td>
<td>Execution Management Services. The general category for grid computation services. This is implemented by the grid’s available BES components, which can all be of different types.</td>
</tr>
<tr>
<td>EPI</td>
<td>EndPoint Identifier.</td>
</tr>
</tbody>
</table>
A short unique pointer (across time and space) to an EPR (see below). EPIs provide for a simple identity comparison, such that if object A has an identical EPI to object B, then they are in fact the same object.

**EPR**  
**EndPoint Reference**

A pointer to a web-service, including network location (such as URL), security policies, and other facts needed for a client to connect to the service.

**Export**

To "export" a file system directory structure is to make it available (subject to access control) to other users in the grid. One exports a local rooted directory tree, e.g., sourceDir and maps it into a target directory in the GFFS directory space, e.g., /home/Alice/project1/sourceDir. The files and directories in “sourceDir” are still accessible using local mechanisms and are also accessible via the grid.

**Export Implementation**

The realization of the export functionality in the Genesis II container, as implemented in source code.

**Export Owner**

The local user at the SP or campus who owns the data being exported.

**Export Owner User ID**

The Unix user ID (also called ‘account name’) of the export owner.

**FUSE mount**  
**File system in User SpacE**

FUSE is a file system driver for Linux and MacOS that allows users to define and write their own user space (non-kernel) file system drivers. Genesis II has a grid-aware FUSE driver that maps the GFFS into the users local file system using a FUSE mount.

**Genesis II**  
**The Genesis System version 2**

A grid computing project developed at the University of Virginia. Genesis II provides the GFFS component for XSEDE.

**Genesis II GFFS Container**

The Genesis II GFFS implements a “Web Services” container architecture. The container is the process running the Genesis II source code in Java with which clients interact. The container receives requests to operate on exported data, and the container’s export implementation carries out those requests (subject to authorization).

**GFFS**  
**Global Federated File System**

The filesystem that can link together heterogenous computing resources, authentication and authorization services, and data resources in a unified hierarchical structure.

**GFFS Container User ID**
The container currently executes with a non-privileged Unix user id. For example, a normal Unix account named 'gffs' might be used to run the GFFS container. In the remainder of the document, the Unix user that is running the container will be referred to as “GffsUser”.

**GORM**  
*Genesis II Omnibus Reference Manual*

This reference manual.

**GIU**  
*Grid Interface Unit*

A Grid Interface Unit (GIU) is the hardware component on which the Genesis II container runs. The required elements of the GIU are defined in the XSEDE Architecture Level 3 Decomposition document (L3D) in section 8.1.2.

**IDP**  
*IDentity Provider*

A service that can create or authenticate user identities.

**L3D**  
*XSEDE Architecture Level 3 Decomposition document.*

**PBS**  
*Portable Batch System*

A queuing service for job processing on computer clusters. PBS queues can be linked to Genesis II grid queues.

**PKCS#12**  
*Public Key Cryptography Standard Number 12*

A file format for storing key-pairs and certificates with password protection.

**PKI**  
*Public Key Infrastructure*

The general category of all services that rely on asymmetric encryption where a key owner has two parts to their key: the public part that can be shared with other users, and the private part that only the owner should have access to. Using the public key, people can send the owner encrypted documents that only he can decrypt. The owner can also create documents using his private key that only the public key can decrypt, offering some proof of the document's origin. With this one essential feature of enabling communication without giving away private keys (unlike symmetric encryption algorithms), a number of important authentication schemes have been developed (such as SSL, SSH, TLS, etc).

**Principle of least privilege**

The term was first coined by Jerome Saltzer, “Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job.” Saltzer, Jerome H. (1974). What this means here is that software that does not need root should not have it. And that if it does need it, then it should have it for the least amount of time in the most encapsulated way (sometimes called privilege bracketing.)

**RNS**  
*Resource Namespace Service*

A web services protocol that provides a directory service for managing EPRs.

**Root Squash**

Because of the way many early distributed file systems handled trust and authentication, processes running as root on file system client hosts may not actually have root privilege with respect to a network mounted file system. When root squash is in effect, the network file server squashes, or
ignores, requests that arrive from clients asserting root privileges. This is done to prevent compromised clients from attacking the file system with root privilege. Note that root squash can be selectively applied by adding exceptions in /etc/exports.

**SSH**  **Secure SHell**

A terminal emulation program that allows users to connect to remote computers while providing an encrypted communication channel that keeps their passwords, command history, and so forth private.

**SSL**  **Secure Socket Layer**

A protocol for connecting to a web service or web site using encrypted transmissions. This protocol is considered deprecated now in favor of TLS.

**STS**  **Secure Token Service**

The STS offers a method for a user to authenticate against a known service in order to log in to the grid. Configuring an STS is usually a task for the grid administrator.

**Sudo privilege**

In Unix there are two important privilege levels, user and root. Root can do anything. Processes with "user" level privilege have very limited capabilities. For example, root can read and write any file, change user id to any user, change file ownership, etc. Users cannot. Sometimes though one wants a user level process to have enhanced capability, without having the infinite capability of root. This is consistent with the principle of least privilege. To support restricted (and temporary) extension of privilege a user may be given sudo (pseudo root) privilege to execute certain commands as either root or as another user.

**TLS**  **Transport Layer Security**

A protocol for connecting to a web service or web site using encrypted transmissions. TLS is the more modern incarnation of SSL.

**Trust Store**  **A set of certificates that are “trusted”**

A trust store can be a file (or directory) with one or more certificates that are trusted for a particular purpose. For example, in Genesis II as of XSEDE Increment 1, there is a trust store in a PFX format file that contains the certificates that a grid client will trust connecting to using TLS. If a container presents an identity that is not present in the trust store and which is not signed by a certificate in the trust store, then no connection will be made.

**Unicore**  **UNiform Interface to COmputing REsources**

The primary EMS for XSEDE is provided by the Unicore software, an EU open source grid computing project initially funded by the German Ministry for Education and Research.
C. Introductory Topics

C.1. Learning the Grid Metaphor

If the reader has not been involved in scientific computing before or would like an overview of the XSEDE GFFS and EMS implemented in Genesis II, this tutorial may be very helpful:

Getting Started with Grids

C.2. System Administrator Tutorial

Readers who are new to administrating a grid or who would like an introduction to the system administrator topics in Genesis II may find this tutorial to be a good introduction:

System Administrator Tutorial

C.3. Public Key Infrastructure (PKI) Tutorial

There are a number of excellent guides that discuss the basic topics of authentication and encryption using modern PKI technologies. The following is just one example:

Everything You Never Wanted to Know About PKI but Were Forced to Find Out

The Genesis II project relies on TLS for authentication and encryption of all grid communications. The following provides a basic summary of TLS:


This is in contrast to other security mechanisms, such as the myproxy server that uses proxy certificates for authentication and authorization. A survey of proxy certificates and related technologies can be found here:

Globus Toolkit Key Security Concepts
[http://www.globus.org/toolkit/docs/4.0/security/key-index.html]

C.4. Documentation Updates

The official html version of the document is available at the following location:


Other formats are available at:

http://genesis2.virginia.edu/wiki/Main/Documentation

This document is also available via a subversion repository and can be downloaded with the following:
D. Genesis II Installation

Genesis II GFFS is a standards-based web-services application. The Genesis II installers can provide both the client-side and server-side of the software. The server side of the web service is called the container. There are interactive installers available for both client and container, and the interactive installer can function with a graphical user interface or in console-mode (text only). The former is intended for most users, while the latter is intended for users who wish to script the install or who do not have access to graphical capabilities during installation.

Genesis II is also available in RPM and DEB package formats on Linux. Unlike the interactive installer, these installation packages are installed by the system administrator once per host. All client and containers configured by users utilize the same installation.

Currently, the container installation is available for 32-bit and 64-bit Linux, and for 32-bit MS-Windows. Client installers are available for 32-bit and 64-bit Linux, for 64-bit Mac OS X (Intel Platform), and for 32-bit MS-Windows.

The Genesis II GFFS software relies on the Java Runtime Engine (JRE) and officially supports Oracle Java 7 (aka version 1.7). The interactive installers include a recent JRE version, but the RPM/DEB packages do not provide a Java JRE.

The Genesis II GFFS software is released under the Apache license agreement, which is available at: http://www.apache.org/licenses/LICENSE-2.0

The installers for Linux, Mac OS X and MS-Windows are available at: http://genesis2.virginia.edu/wiki/Main/Downloads

D.1. Installer User Considerations

The average user who wishes to use the Genesis II container or client does not need to have administrative access to the computer where the installation will occur. In general, a user who has a home directory with write access can just run the installer as their own personal identity, and there are no special permissions required for running either the container or the client on one's own computer.

In some institutional or corporate settings, administrators may prefer to install the software at a single location per computer or per network. This is also supported by the Genesis II installer (in both interactive and Linux package formats). The container owner needs to perform additional tasks to configure Genesis II, which are documented in the sections below.

A common requirement is for the grid tools to be available in the user's application path. One solution for this is a TCL Modules file, and a sample file is provided in the XSEDE tests in the folder called “tools/genesis_module”. There are many other ways to address the path issue, including modifying environment variables (per user or per system).

D.2. Installing the Grid Client Using the Graphical Installer
This section will walk through the installation process for the Genesis II GFFS client using the interactive installer in its GUI mode.

The GUI-mode installer can be launched by double-clicking the installation executable on Windows and Mac OS X. On Linux, the installer can be launched with bash:

```
bash genesis2-gffs-linux64-v2_7_503.bin
```

This will begin an interactive install process where graphical dialogs are displayed to request configuration input.

![Figure 1. Installer Welcome Dialog](image1)

Clicking “Next” leads to picking the installation location.

![Figure 2. Installation Location Dialog](image2)

The next dialog allows one to choose the type of installation, client-only or full GFFS container. Leave it at the default choice of client-only if you do not need a container installation that will
provide GFFS services of your own.

Figure 3. Installation Type Dialog

Note that during an upgrade, the installation type dialog will default to the previously selected choice of client vs. container.

After picking the type of installation to perform, files are copied into place.

Figure 4. Active Copying of Files

Once the Genesis II software files are stored in the target location, the GFFS software will be used to connect to the configured grid. If the grid connection does not succeed and an error message is printed, please refer to the FAQ, Section J, for possible solutions.

When the installation is finished, the completion dialog is displayed.
D.3. Installing the Grid Client at the Command-Line

The console version of the installer is available from the same install program that does the graphical installs. On Linux, the command-line version requires passing a '-c' flag to the installer at run time:

```
bash {installer filename} -c
```

For MS-Windows, run the installer as an exe instead. This assumes the user is in the same directory as the installer:

```
genesis2-gffs-linux64-v2_7_503.exe -c
```

This will begin an interactive install process where prompts are displayed to request configuration input. The same prompts shown in the graphical install dialogs are shown on the console instead.

```
Unpacking JRE ...
Preparing JRE ...
Starting Installer ...
This will install Genesis II GFFS on your computer.
OK [o, Enter], Cancel [c]
  (hit enter)
Where should Genesis II GFFS be installed?
[/home/fred/GenesisII]
  (type a different location or use suggested one, then hit enter)
```
Please Select the Type of Install to Perform
Installing grid deployment for Fred's internal grid
This installation can provide GFFS client-only support or it can function as a GFFS container. Which would you prefer to install?
Client-Only GFFS [1, Enter], GFFS Client and Container [2]
1 (type 1 for client install or just hit enter)
Extracting files ...
  (...filenames flash by...)
Connecting to the Grid
  (slight pause occurs while connecting...)
Setup has finished installing Genesis II GFFS on your computer.
Finishing installation...

If it is important to automatically script the grid client installer, one technique that can help is called a 'here document'. The here document, denoted by the << below, answers the installer prompts using a list of canned responses. The word ‘eof’ below is used to end the stream of commands:

```
bash genesis2-gffs-linux64-v2_7_503.bin -c <<eof
  0
  /home/fred/GenesisII
  1
  eof
```


**D.4. Installing a Grid Container Using the Graphical Installer**

This section will walk through the installation process of the Genesis II GFFS container using the interactive installer in its GUI mode.

The first four dialogs are roughly the same as the client installer. Beginning at the third dialog, the installation diverges. On the installation type dialog, choose a GFFS container install type.
Once the “Next” button is clicked, the files will be installed similarly to the client installer.

After the files are in place, the container installation prompts for items specific to the container configuration. The next dialog requests to know the web services configuration for the container.

The port number is where the container will reside on the current host. This port number should not be in use by any other service, and it must not be blocked by a firewall. The hostname should be a publically visible DNS name (or IP address, although DNS names are preferred). This host must be reachable using that name from potentially anywhere in the world, or the container will not be able to be linked into a grid.

After the web service configuration, the installer will attempt to connect to the configured grid. Once this completes, the container specific configuration continues with a dialog requesting to know which grid user will own the container.
The user specified must be an existing user in the GFFS grid in question (the location of which is packaged in the installer). If you do not currently have a valid grid user, you will need to request one that can own your container.

The grid user in question will completely “own” the container and will be given full administrative rights. This allows the user to add, configure and remove resources on this container. The grid user can also link the container into the grid’s RNS hierarchy in locations where the user has appropriate access rights.

After a valid grid user is provided, the installation offers to generate certificates for the container or to let the user provide her own certificate. This certificate is used for TLS (SSL) communication by the container; all outgoing and incoming web service calls use this certificate for identification and all encryption is done with the associated private key.
If the keypair generating service is used, as depicted, then a certificate for TLS communication is automatically generated. In that case, the next dialog requests to know the password for the generated TLS certificate.

![Figure 10. Choosing Password for Generated TLS Keypair](image)

The default password for the TLS keystore is ‘container’, but this can be changed as desired. After the TLS keystore and certificate are generated, the installer finishes with the final dialog.

Alternately, if one chooses not to use the keypair generator, one must supply a TLS keypair in PFX format that can be used for the communication. The keypair dialog prompts for the keypair and supports browsing for it on the local computer.

![Figure 11. Specifying Existing Keypair and Password](image)

Once an appropriate PFX file has been provided with the proper password, the installation continues to the final dialog.

**D.4.1. OpenSSL Conversion**
The following commands may be helpful for converting between the PEM and PKCS#12 formats.

To convert a certificate from DER format to PEM format:
openssl x509 -inform der -in certificate.cer -out certificate.pem

To convert certificate from PEM format to DER format:
openssl x509 -outform der -in certificate.pem -out certificate.cer

To convert keypair from PKCS#12 format to PEM format:
openssl pkcs12 -nodes -in keypair.pfx -out keypair.pem

To convert PEM format certificate and private key to PKCS#12 format:
openssl pkcs12 -export -out keypair.pfx -inkey private.key -in certificate.pem

If the CA certificate is not in the certificate.pem file, then add this flag: -certfile CA.pem

**D.5. Installing a Grid Container at the Command-Line**

The console mode installation process for the container is very similar to the client install. There are just a few more questions to answer than for the client, all regarding the container configuration.

---

(Installation is shown after container install type is selected and files have been installed.)

### Port Number

By default the XCG container listens for incoming messages on TCP port 18443. You can override this behavior here.

**XCG Container Port Number**

[18443]

*(select a port number and hit Enter)*

Specify the hostname or IP address where your container will run. Hostnames should be globally resolvable via DNS.

**Host Name**

[]

*(type in the world-reachable DNS host name for the computer and hit Enter)*

### Connecting to the Grid

**Owner Information**

Please select a user to manage the container

**User Name**

[]
This service will generate and sign your container keypair with your supplied credentials.

Use Grid Keypair Generating Service?
Yes [y], No [n, Enter]
(choose and hit enter. Remainder assumes keypair was not generated.)

Select path for container keypair (.pfx) to be used for this container (will be copied)
Keypair Path
[
(enter the path to a key-pair to use as the TLS key for the container)

Keystore Password
[
(enter the key-pair and keystore password for the pfx file; these must both be the same password to use the pfx with the GFFS installer.)

Start Container Service?
Yes [y], No [n, Enter]
(hit Y and then Enter to start the container)

Configuring Container
Preparing GFFSContainer Script
Starting Container Service

Setup has finished installing Genesis II GFFS on your computer.
Finishing installation...

Note that the same approach used for scripting the grid client (a 'here document') can be used to script this install.

D.6. Automatic Container Start-up

A system administrator can configure a Genesis II container installation to automatically restart when machines are rebooted.

D.6.1. System-wide Installation Start-up

Genesis II installations provide a sample init.d-style service script in a file called "GFFSContainer". This file can be deployed on some Linux systems in /etc/init.d to automatically restart the container. Once the file is installed, the system administrator must set the script at an appropriate "run level" for starting on reboot.

D.6.2. Personal Installation Start-up

Users who wish to automatically start their personal containers can do so with a "cron job". This method usually does not require administrator assistance. The Genesis II installation provides a script called GFFSContainer which can be used to restart the service if the computer is restarted or...
if the service inadvertently stops. The following is an example cron job that uses the container restart script to launch the container if it is not already running.

```
# checks every 5 minutes for Genesis II container service and restarts it if missing.
GENII_USER_DIR=$HOME/container_state
*/5 * * * * $HOME/GenesisII/GFFSContainer start
```

Example of a cron job that restarts the GFFS container

Cron has a different environment than your normal users, and thus it is important to provide the state directory (GENII_USER_DIR) to the cron job. Otherwise the default state directory ($HOME/.genesisII-2.0) will be used.

D.7. Installation on Linux Using an RPM or DEB Package

The installation of one of the Linux-based packages is much simpler than the interactive process, but mainly because the configuration steps have been moved out to script-based process. This is necessary because the RPM and DEB package formats are intended to be installed once per host, and shared between multiple users for the software. In these package formats, all user and container state must reside in the state directory (unlike the interactive installation, where some of the configuration can reside in the installation directory).

To install or upgrade the Linux RPM for the Genesis II GFFS, use sudo (or login as root) to call the rpm installer:

```
sudo rpm -Uvh genesis2-xede-2.7.503-1.x86_64.rpm
```

To install the Linux DEB package for the Genesis II GFFS, use sudo (or login as root) and run the dpkg program:

```
sudo dpkg -i genesis2-xede-2.7.503-1.x86_64.deb
```

Each of these actions will install the Genesis II GFFS software to “/opt/genesis2-xede” by default when using the XSEDE production grid install package. Installers for other grids will follow a similar form. For example, the European Grid (GFFS.EU) which will install to “/opt/genesis2-european” and the XCG installer installs to “/opt/genesis2-xcg”.

To install to a different location when using RPMs, add the “prefix” flag to the command:

```
sudo rpm -Uvh --prefix {new-location} genesis2-xede-2.7.503-1.rpm
```

To uninstall the RPM or DEB package, use the appropriate package manager’s removal procedure:

```
sudo rpm -e genesis2-xede
or
sudo apt-get remove genesis2-xede
```

If needed, the RPM install can be forced to upgrade a package with identical version information despite already being installed:
The process of configuring container installations and converting older installations is documented in the following sections. The configuration scripts documented below can also be used with interactive installs (on Linux only), which is especially useful when those are installed by the root user for host-wide usage.

### D.7.1. RPM and DEB Upgrade Caution

Multiple containers can be configured on a host using the system-wide RPM or DEB package for Genesis II. This poses an issue at upgrade time, since the running containers will become unavailable when the Java jar files and configuration directories are replaced. The system administrator may want to institute a procedure for alerting users to shut their containers down before the installation and to restart the containers again afterwards. An alternative is to require users to register their container installations in a way that allows a site-implemented, sudo-based process to automatically stop all of them before the installation and start them again afterwards. A mechanism for automating this process may be developed in a future release.

### D.8. Unified Configuration for Containers

The interactive installers provide what is termed a “Split Configuration” installation mode, where the container configuration partially resides in the installation folder itself. In the newer “Unified Configuration” mode, the client-specific and container-specific configuration is stored entirely in the state directory. This is a more flexible configuration, which can operate based on the RPM/DEB packages as well as on the interactive installer. The following sections describe the procedures used for managing containers with the Unified Configuration.

In general, the Unified Configuration is the most useful on Linux when the RPM or DEB package is installed. However, these same approaches can be used directly on Mac OS X also. On MS-Windows, using a Linux compatibility such as Cygwin is required (see section I.1.3 for more information on Cygwin).

In all of the Unified Configuration scripts documented below, the environment variables GENII_INSTALL_DIR and GENII_USER_DIR must be set. The former variable specifies the install location (such as /opt/genesis2-xsede), and the latter specifies the state directory for the container (such as $HOME/.genesisII-2.0). The install directory and the state directory do not need to exist before running the installer, but the two environment variables must be established.

### D.8.1. Creating a New Container That Uses a Unified Configuration

To configure a new container, first install a version of the Genesis II GFFS that provides the Unified Configuration (2.7.500+). Run the configuration script with no parameters to get full help instructions:

```bash
bash $GENII_INSTALL_DIR/scripts/configure_container.sh
```
The instructions provided by the script should be complete, if a bit terse. This section will explain some finer points of the required parameters, but the script’s built-in help should be consulted as the most authoritative and up to date reference.

There are potentially six parameters for the script, and it requires at least five of these. They are:

1. The container host name. This is the globally visible name at which the new container can be reached over the internet. It is alright for this to be an IP address, although a textual host name is preferred.

2. The network port number where the container service provides TLS connection services. This port number must not be blocked by a firewall. The GFFS container must also be the only user of this port number.

3. An existing grid user who will be given complete control over the new container. This should be your grid user name, and it can be an XSEDE-style MyProxy/Kerberos user or a GFFS X509 style user. If you do not have a user name yet, contact your grid administrator to acquire one. In some rare cases, the grid administrator may provide a special user name that will own your container, rather than your own grid user.

4. The keypair file in PFX format (that is, PKCS#12) which holds the TLS certificate and key-pair that the container will use to communicate over the internet. This can either be an already existing PFX file or it can be the word “generate”, which will cause a new TLS keypair to be created by the grid’s certificate generator (where available).

5. The password on the keystore itself. This password is used to unlock the keystore and get at the keypair inside it. If parameter 4 was “generate”, then this password will be used to secure the newly generated keystore.

6. A password for the TLS key within the keystore. This is optional and will only be necessary when the keystore and key password differ. This parameter is not used for the “generate” keystore option.

Note that the script will produce diagnostic output during configuration which can include passwords, so it may be wise to run “clear” or “cls” in that terminal afterwards.

After running the configuration script with the appropriate parameters, the container’s configuration will be built in the GENII_USER_DIR directory. The script prints out a command that will start the container running. For example, the new container might be started up using the default RPM package location:

```
/opt/genesis2-xsede/GFFSContainer start
```

After launching the container, its output can be watched with the “tail” command (assuming the default logging location):

```
tail -f $HOME/.GenesisII/container.log
```

If that shows no errors, then the container is now configured and could be linked into the grid provided by the installer (see Section F.3.1 for more details about linking the container).
D.8.2. Converting a Container in Split Configuration Mode to a Unified Configuration

Users may want to free themselves from the Split Configuration mode after they have previously configured a container with the interactive installer. Typically, this will involve installing an RPM or Deb package to provide the new installation. The existing container can be converted into the Unified Configuration mode with a provided script, which will acquire configuration items from the interactive installation (which must still exist at conversion time). To see the built-in help for the conversion script, run the following:

```bash
$GENII_INSTALL_DIR/scripts/convert_container.sh
```

This will show the required parameters and some example execution sequences. This script is considerably simpler than the configure script (last section), as all of the configuration information should already exist and just needs to be extracted from the old installation directory.

It is important to back up the container state before the conversion process, in order to defend against any unexpected problems during the conversion. Both the installation directory (pointed to by the GENII_INSTALL_DIR variable) and the state directory (specified by the GENII_USER_DIR environment variable or residing in the default location of $HOME/.genesisII-2.0) should be archived. For example, this will create an archive of both directories, assuming the environment variables are set:

```
$ tar -czf container_backup.tar.gz $GENII_INSTALL_DIR $GENII_USER_DIR
```

The most common way to run the container conversion script is to migrate an old interactive installation to using the RPM/DEB package format. It is important to fix the GENII_INSTALL_DIR to point at the newer install location before running the convert script, e.g.:

```bash
export GENII_INSTALL_DIR=/opt/genesis2-xsede
bash $GENII_INSTALL_DIR/scripts/convert_container.sh $HOME/GenesisII
```

The script will produce diagnostic output during the conversion which can include passwords, so it may be prudent to run “clear” or “cls” in that terminal afterwards.

It is possible to convert to a Unified Configuration even if there is only one installation of the newer interactive installer (e.g., if the old installation was upgraded in place). In this situation, pass the current $GENII_INSTALL_DIR as the parameter to the script.

After the conversion script has run successfully, the container’s configuration will be unified under the state directory. The older interactive installation can be removed, and the container will rely on the new package location for the GFFS software.

During the execution of the script, you will be offered a chance to create a copy of your deployment folder from the old installation. This is only necessary if you have manually modified the deployment, or if the deployment is non-standard. This is true for upgrading a source-based container to use an RPM, which is further documented in section D.8.7.
D.8.3. Changing Container’s Installation Source for Unified Configuration

After converting a container to the Unified Configuration, it is sometimes necessary to adapt to changes in the installation location. This may occur if the container was initially converted from an older interactive install to the newer interactive install, but then later the RPM install is used instead. The install also might need to change locations due to organizational or hardware changes.

In these cases where there is no other configuration change required for the container, the location can be fixed with the “update_install_location” script. Running the script prints out the built-in help:

```
bash $GENII_INSTALL_DIR/scripts/update_install_location.sh
```

This is a very simple script. The GENII_INSTALL_DIR should point at the new install location, and the older location is passed on the command line. Below is an example of switching to the RPM package as the new installation source, after having previously relied on the interactive installation to support the container’s Unified Configuration.

```
# if the old installation is still active, stop that container...
$GENII_INSTALL_DIR/GFFSContainer stop

# update the installation directory variable to the new path...
export GENII_INSTALL_DIR=/opt/genesis2-xsede

# fix the install paths...
bash $GENII_INSTALL_DIR/scripts/update_install_location.sh $HOME/GenesisII
```

Again, this is only appropriate for switching the location of a container that already has the Unified Configuration (see prior section for information about converting to the Unified Configuration).

D.8.4. Updating Container’s Grid Deployment for Unified Configuration

A GFFS deployment provides the information needed to connect to a grid, such as the grid location on the internet and the associated certificates for that grid. Occasionally some characteristics of the grid deployment are updated, and these are pushed out in a new deployment package or in a new installer.

For containers with a Split Configuration mode that are set up by interactive installers, this usually poses no problem, as the installer can update the deployment when the new version is installed. But containers with a Unified Configuration are more independent from the installation directory and are not automatically updated to the latest deployment. This is a consequence of the RPM/DEB installation model, where the root user installs the package, but many other users can base their container on the installed package. These types of containers require a deployment update in order to use the latest grid deployment.

The deployment updater has built-in help that can be accessed by running the script with no parameters:

```
bash $GENII_INSTALL_DIR/scripts/update_deployment.sh
```
The script requires two parameters, which are a deployment name and a grid context file name. The named deployment must be an existing folder located under the current deployments folder. The current deployments folder can be pointed at by the “$GENII_DEPLOYMENT_DIR” variable. If that variable is not set, then the deployments folder falls back to the default of “$GENII_INSTALL_DIR/deployments”. The use of a GENII_DEPLOYMENT_DIR variable is uncommon but useful if one’s deployments are not located under the GFFS installation directory. The specified context file must also reside in the named deployment folder.

Once the newer Genesis II GFFS installer with the updated deployment is installed, one’s container configuration can be updated by running:

```
bash $GENII_INSTALL_DIR/scripts/update_deployment.sh current_grid xcg3_context.xml
```

In this case, “current_grid” was already the existing deployment name and the “xcg3_context.xml” file was the existing context file. The names may not always remain the same however, especially if one is updating to a completely different deployment (which is a fairly rare occurrence). After running the script above, the container configuration in $GENII_USER_DIR will be synchronized with the important attributes of the new deployment.

If you do not know the name of the deployment or the context file, it is often enough to look into the deployments folder of your installation; the deployment name will be a folder other than “default”. The context file should reside inside that folder and its filename should end in “.xml”.

**D.8.5. Using a Grid Deployment Override**

There are two methods for using a different deployment than the deployment provided by the Genesis II install package.

The first method is to set the variable GENII_DEPLOYMENT_DIR in the environment before starting the container. This causes the container to use that folder as the root of the deployments hierarchy, rather than the default of $GENII_INSTALL_DIR/deployments.

The second method is to store the specialized deployment hierarchy in a folder called “deployments” under the container’s state directory (in $GENII_USER_DIR). If the container finds a folder named “deployments” in its state directory at start-up, then it will use that one instead of the one stored in the installation directory.

The order of precedence for finding the deployment folder is first to check the GENII_DEPLOYMENT_DIR variable, then to look for “deployments” in the container state directory (GENII_USER_DIR), and finally to look for deployments under the GENII_INSTALL_DIR.

**D.8.6. Unified Configuration Structure**

The unified configuration mode for the installer provides a method for overriding values that were previously always provided by the installed deployment. This allows all of a container’s unique information to be managed in the container’s own state directory.
The unified configuration adds these files and directories to the state directory:

```
installation.properties
certs/
webapps/
wrapper/
deployments/ (optional)
```

D.8.6.1. *installation.properties* file

The installation.properties file provides override values for configuration properties that are otherwise provided by the “configuration” directory of a deployment. This includes the files “security.properties”, “server-config.xml” and “web-container.properties”. The following is an example of a real “installation.properties” file for a container that relies on the installed deployment:

```
gffs-sts.kerberos.keytab.TERAGRID.ORG=KHANDROMA.CS.VIRGINIA.EDU@TERAGRID.ORG
gffs-sts.keytab
gffs-sts.kerberos.principal.TERAGRID.ORG=gffs-sts/KHANDROMA.CS.VIRGINIA.EDU@TERAGRID.ORG
edu.virginia.vcgr.genii.container.external-hostname-override=surya.gruntose.blurgh
edu.virginia.vcgr.genii.container.listen-port=18080
edu.virginia.vcgr.genii.container.security.ssl.key-password=**
edu.virginia.vcgr.genii.container.security.ssl.key-store-password=**
edu.virginia.vcgr.genii.container.security.resource-identity.key-password=**
edu.virginia.vcgr.genii.container.security.resource-identity.key-store-password=**
edu.virginia.vcgr.genii.container.security.resource-identity.container-alias=signing-cert
edu.virginia.vcgr.genii.container.security.certs-dir=/home/fred/.surya_grid_state_dir/certs
edu.virginia.vcgr.genii.container.security.ssl.key-store=tlscert.pfx
edu.virginia.vcgr.genii.container.security.ssl.key-store-type=PKCS12
edu.virginia.vcgr.genii.container.security.resource-identity.key-store=signing-cert.pfx
edu.virginia.vcgr.genii.container.security.resource-identity.key-store-type=PKCS12
edu.virginia.vcgr.genii.gridInitCommand="local:/opt/genesis2-xsede/deployments/surya-grid/surya_context.xml" "surya-grid"
edu.virginia.vcgr.genii.container.deployment-name=surya-grid
```

Note that there will be significantly fewer fields if the container installation carries its own “deployments” folder in the state directory. In that case, the security properties come from the deployments folder rather than the installation.properties file.

As the above shows, the installation.properties is formatted as a java property file, and provides “name=value” definitions of variables. Each of the above entries corresponds to a setting that would otherwise have come from the deployment’s configuration files.
Generally this file should not be hand-edited, but that is always an option if additional overrides are needed or if values must be corrected to adapt to changes.

**D.8.6.2. certs directory**

This directory is used to store container specific certificates and Kerberos keytab files for authentication and authorization. It has a structure mirroring the “security” folder from the installed deployment, and thus can contain a “default-owners” and a “trusted-certificates” directory.

The container configuration and conversion scripts automatically store the container’s certificate files in PFX format in this directory when using the unified configuration mode.

**D.8.6.3. webapps directory**

This directory supports the Apache Axis web services software and provides a storage place for temporary files.

**D.8.6.4. wrapper directory**

Used by the Java Service Wrapper for the container’s service management. This provides the wrapper configuration file in “wrapper.conf”. It also is the location where the service wrapper will track the container’s active process id in “GFFS.pid”.

**D.8.6.5. deployments directory**

If a directory called deployments is found in the state directory, and there is no GENII_DEPLOYMENT_DIR environment variable established, then this folder is used as the deployments folder, rather than the default of $GENII_INSTALL_DIR/deployments. The convert_container script offers to create this directory (as a copy of the previous installation’s deployments folder) during conversion.

**D.8.7. Converting a Source-Based Container to a Unified Configuration**

Converting a container that is built from Genesis II source code is a special case of the conversion process in Section D.8.2. This usually only applies to the bootstrap container for a grid, or to experimental containers used by developers. For these cases, the conversion script should perform the proper actions, but there are a few important choices to make during this process.

To convert the source-based container, follow the steps described above in Section D.8.2 to convert the source folder from “split configuration” to “unified configuration”, but with the following additions:

1. If the source-based container is still running when executing the convert_container script, then the script will show text regarding “There are still Java processes running...” If the script finds any of these processes, then answer “Y” to the question of whether to shut them down. This will only stop Java processes that are detected as running Genesis II containers or clients. Care should be taken if the same user account is running more than one Genesis II container; in that case, stop the source-based container manually.

2. When the convert_container script asks whether to copy a specialized “deployments”
folder, tell it to do so by answering "Y". This is crucial for a root container's specialized deployment to be preserved and is also needed in cases when the deployment generator was used to create the deployments folder.

Both of these choices can be automated by using optional flags to the convert_container script, as in the following script execution example (replace the path for {genesis2-trunk} with your container's source code location):

```bash
# Switch installation dir variable, for example to xsede install location:
export GENII_INSTALL_DIR=/opt/genesis2-xsede
# Perform the conversion:
bash $GENII_INSTALL_DIR/scripts/convert_container.sh {genesis2-trunk} \
   stop+depcopy
```

The "stop" phrase will cause any Genesis II Java processes to be stopped. The "depcopy" phrase causes the deployments folder to be copied from the installation directory into the container state directory.

After the conversion is successful, the source code should no longer be needed to run the container, and it can be removed.

**E. Grid Usage Topics**

This section describes how to get computational work done with a grid based on Genesis II GFFS software (such as the XSEDE and XCG grids). It is assumed that the grid is already configured, and that the user has already been issued a grid user account by the grid administrator.

**E.1. Built-in Help**

Genesis II has built-in help available for most commands. The command grid help prints a list of the available commands. Additionally, each individual grid command has a short help description for usage and also a longer man-page style description.

```bash
# print a list of the available commands.
grid help
# show usage information for a command.
grid help {command}
# show the manual page for a command.
grid man {command}
```

**E.2. Authentication and Authorization**

In the grid, a user's capabilities are based on who they are and what they've been granted permission to do. Authentication is the process that a user goes through to show who they are, at least in terms of an identity that the grid will accept. This proof is limited; the user has merely presented a certificate or a valid login that the grid recognizes. It is not proof that the user actually is a particular person; it just proves that she possesses the credentials associated with that person.

On the other hand, authorization is the full set of capabilities that specify what a particular identity is allowed to do. In the case of the GFFS, the user's authorization is specified by access control lists
on resources that the user has the right to use in some particular manner. For example, the user may have authorization to submit a compute job to a particular queue.

The following sections detail the processes of grid authentication and grid resource authorization.

**E.2.1. Credentials Wallet**

Genesis II uses what is termed a “credentials wallet” to store user identity for grid operations. The wallet contains all the identities that a user has “authenticated” with the grid using a supported protocol, such as by providing a username and password, or by logging into a Kerberos domain.

Users may require a collection of identities for their work, rather than just one. For example, the user may have allocations at a Supercomputing Center as well as having a local campus identity. The credentials wallet allows the user to present all of her valid identities with a single login.

**E.2.1.1. Who Are You?**

A grid client instance that is not connected to a grid container initially has no identity at all. As part of making the secure connection to a grid container, the client creates a self-signed certificate to represent its own identity. Upon attempting to connect to a grid container, the grid client examines the identity of the container and compares it with the client’s own “trust store”. The trust store is a set of server certificates that the grid administrator has instructed the client to “trust”. Trust here just means that the client will connect to containers that identify themselves via one of those certificates, and it will not connect to any containers that are not in the trust store. More details about the trust store are available in the section on GFFS Deployments.

```bash
# show the initial certificate on a client that has never
# authenticated as a user before.
grid whoami
```

When the client has no previously cached identity, this command shows just the certificate that the grid client created to represent its side of the secure TLS connection. This is an example of the “whoami” output for a client in that state.

```text
Client Tool Identity:
(CONNECTION) "Client Cert 90C75E64-D5F9-DCC2-A11F-584339FD425F"
```

Once the client has decided to trust the container (and possibly, based on configuration, the container has made a similar decision to trust the client), the secure TLS connection is made and services can be requested by the grid client. The first of the requested services is generally a login request, because the client must authenticate as an identity of some sort to obtain any authorization for grid resources. Different methods for logging in are discussed in the next section.

**E.2.2. How to Login & Logout**

Genesis II supports a variety of authentication mechanisms, including username & password, Kerberos, MyProxy, InCommon, and direct use of a key-pair. Each of these methods may be appropriate for a different reason. Thanks to the credentials wallet, the user does not need to pick
just one approach, but can attain whatever collection of identities that are needed to get the work done.

E.2.2.1. **Logging Out of the Grid**

Although it may seem counter-intuitive to log out before having logged in, this can be done and is not a null operation; logging out always clears at least the self-signed client certificate. If the user had previously authenticated to any grid identities, those identities are dropped as well.

```bash
# logout of all identities.
grid logout --all
```

It is possible to log out of just one identity by specifying its "alias" on the command-line. Identities each have a unique alias name, and the alias is shown in the whoami listing.

For example:

```bash
# Example of grid whoami result
Client Tool Identity:
 (CONNECTION) "Client Cert 90C75E64-D5F9-DCC2-A11F-584339FD425F"
Additional Credentials:
 (USER) "drake" -> "Client Cert 90C75E64-D5F9-DCC2-A11F-584339FD425F"
 (GROUP) "uva-idp-group" -> "Client Cert 90C75E64-D5F9-DCC2-A11F-584339FD425F"
 (USER) "skynet" -> "Client Cert 90C75E64-D5F9-DCC2-A11F-584339FD425F"
```

This alias can then be used to log the identity out:

```bash
# log out of a user identity.
grid logout --pattern=skynet

# or log out of the group.
grid logout --pattern=uva-idp-group
```

E.2.2.2. **Login with Grid IDP**

The grid's identity provider (IDP) supports standard username and password authentication for users to log in to the grid. The username and password in question must already have been set up by the grid administrator. To log in with a grid user identity, use:

```bash
grid login --username={drake}
```

In a graphical environment, this will pop up a dialog for filling in the password. In a console environment, there will be a prompt asking for the password at the command shell.

Note that the password can be included in the login command if it is absolutely required. This may be needed for scripting a grid login, but it is not generally recommended because the password will be visible in script files or in command history:

```bash
grid login --username={drake} --password={myPass}
```

E.2.2.3. **Login With Kerberos**
For users to log in using a Kerberos STS, the STS must already have been created according to the instructions in the section "Using a Kerberos STS". Once the Kerberos STS exists, users can log in with the following command:

```
grid login rns:{/containers/containerPath}/Services/KerbAuthnPortType/ {userName}
```

This will bring up a password dialog for the userName specified.

### E.2.2.4. Login From a Keystore

In some cases, user identity may need to come from a key-pair stored in a file. This is often the case when a user needs to authenticate as a grid administrator. It is also possible that a key-pair will be issued by a resource owner to control access to the resource. In order to obtain authorization on that resource, merely being logged in as a known grid user would not suffice and the user must add the key-pair credentials to the wallet.

To authenticate using a keystore file (such as a PKCS#12 format PFX file):

```
# using a keystore on a local disk.
grid keystoreLogin local:{/path/to/keyFile.pfx }

# or using a keystore in the grid.
grid keystoreLogin grid:{/home/drake/keyFile.pfx }
```

### E.2.2.5. Login Using xsedeLogin

The xsedeLogin command is a special purpose login for users of the XSEDE grid. It authenticates to the XSEDE Kerberos server and the XSEDE MyProxy server in order to obtain both types of identities for grid services. It is very similar to the simple login command

```
# log in to the grid.
grid xsedeLogin -username={drake}
```

If login is successful, the "whoami" listing may appear as follows:

```
Client Tool Identity:
(CONNECTION) "Drake Valusic"
Additional Credentials:
(USER) "drake" -> "Drake Valusic"
```

In the case of the XSEDE-style login, there is no self-signed certificate for the client. The client's identity is instead dependent on the Kerberos authentication using a real XSEDE portal ID for login.

### E.2.2.6. Logging in with InCommon

The iclogin command uses the Enhanced Client or Proxy protocol (ECP) to authenticate to an InCommon identity provider (IDP), and then use that authentication to acquire grid credentials. Any of the previous STS types may be the target of an InCommon login, as long as it has been set up according to the section "Setting up an InCommon STS" (Section G.1.11).

Once the InCommon STS link exists, users can log in with the following command:

```
grid iclogin
```
There are five parameters to log in using InCommon:

1. The URL of the IDP’s ECP service endpoint,
2. The user id and
3. The password for the user at that identity provider, and
4. (optional) An SSL public/private keypair and
5. (optional) An associated SSL certificate signing request (CSR).

In a graphical environment, dialogs will be displayed to retrieve these parameters. In a console environment, the user will be prompted in the command shell. Alternatively, all of these parameters, or any subset may be specified at the command line, such as follows:

Using a keypair file on a local disk

```
grid iclogin --idp=https://url/of/IDP/endpoint --username=drake \ 
--password=myPass --key=local:/path/to/local/key/file \ 
--csr=local:/path/to/local/CSR/file
```

Using a keypair file in the grid

```
grid iclogin --idp=https://url/of/IDP/endpoint --username=drake \ 
--password=myPass --key=grid:/path/to/grid/key/file \ 
--csr=grid:/path/to/grid/CSR/file
```

If the user does not wish to specify an existing SSL keypair, a new keypair and CSR will be generated by the client. If the user does specify a keypair file, he may also choose to provide a CSR as well or have one generated which contains the provided public key.

The iclogin tool uses the InCommon authentication service at CILOGon.org to generate an authentication request for the provided or generated CSR, forwards the request to the selected IDP with the provided credentials for a signed assertion of identity, and then returns the assertion to CILOGon.org to retrieve a X.509 certificate. As in the xseedLogin, the self-signed session certificate is discarded, and the certificate from CILOGon.org becomes the current client session certificate. Finally, the iclogin tool contacts the STS corresponding to the InCommon credentials provided to acquire additional grid identity certificates, which are delegated to the CILOGon.org session certificate.

### E.2.3. Grid Access Control Lists (ACLs)

Upon authentication, the user may perform all actions she is “authorized” to perform. In Genesis II, authorization is implemented using a technique called Access Control Lists. Every resource in the Genesis II GFFS has three access control lists which are called Read, Write, and Execute ACLs. Each type of ACL can have from zero to an arbitrary number of grid identities listed. This associates the decision-making information about whether a resource is accessible onto the resource itself, rather than associating it with a user or a group (as might be done in a capability model rather than an ACL model).
There are a few generally applicable attributes for the Read, Write and Execute ACLs, but specific resources can vary how these ACLs are interpreted. In general though, Read access grants a user identity the right to see a resource. Without Read access, the user cannot even list the contents of that resource in the GFFS.

Generally speaking, Write access often is considered to grant administrative access to the resource. For example, a queue that lists a user X in its Write ACL is granting user X the right to completely control the queue, even to the extent of removing queued jobs of other users or changing the properties of the queue.

The general interpretation of the Execute ACL is to make a resource available to a user for whatever primary purpose the resource provides. For example, a user with Execute access on a queue is allowed to submit jobs to it, and to cancel her own jobs. That user cannot however manage the jobs of other users or change the attributes of the queue.

**E.2.3.1. How to See ACLs in the Grid Client**

Genesis II provides two ways to display the ACL lists for a resource: the console grid client and the graphical client UI. The graphical client provides a summary of the permissions for user ids, whereas the console client displays the full authorization data (including the EPIs that uniquely describe user identities in the ACLs).

You can show the authorization information for any resource in the GFFS using the grid authz command.

```
grid authz {/path/to/resource}
```

**Example listing of just the Read ACL for a grid path**

Read-authorized trust certificates:

[0] (X509Identity) "CN=EnhancedRNSPortType, SERIALNUMBER=urn:ws-naming:epi:41A37E3B-8E0A-0502-9DDA-BCA21C8E0008, OU=Genesis II, O=GENIITEST, L=Charlottesville, ST=Virginia, C=US"  [06/04/12 11:05:45, 06/05/13 11:05:45]

[1] (X509Identity) "CN=X509AuthnPortType, CN=admin, SERIALNUMBER=urn:ws-naming:epi:B0D0624B-9939-9A8E-4682-52A416657D88, OU=Genesis II, O=GENIITEST, L=Charlottesville, ST=Virginia, C=US"  [06/04/12 11:04:28, 06/05/13 11:04:28]

[2] (X509Identity) "CN=X509AuthnPortType, CN=drake, SERIALNUMBER=urn:ws-naming:epi:2A9784BC-2DF8-42D0-2C34-00CE2857B9D9, OU=Genesis II, O=GENIITEST, L=Charlottesville, ST=Virginia, C=US"  [06/04/12 11:05:52, 06/05/13 11:05:52]

[3] (X509Identity) "CN=X509AuthnPortType, CN=uva-idp-group, SERIALNUMBER=urn:ws-naming:epi:5CF49A70-88F8-C08B-2DAF-ED0029C8D2F5, OU=Genesis II, O=GENIITEST, L=Charlottesville, ST=Virginia, C=US"  [06/04/12 11:04:00, 06/05/13 11:04:00]

[4] EVERYONE

Note that this particular resource allows “everyone” to read it. This is often the case for top-level GFFS folders and other assets that are part of the “grid commons” available to all users. Also of interest are the EPIs (listed after urn:ws-naming:epi: that uniquely specify a particular grid identity.
To use the client-ui for viewing ACLs, launch the client (grid client-ui) and navigate to the file or directory of interest in the RNS Tree. Once an item has been selected (by left-clicking with the mouse), the ACL pane on the right will show the Read, Write and Execute permissions for that resource.

E.2.3.2. Meaning of Read, Write, and/or Execute Permissions

The interpretation of the Read ACL is constant within the grid for all resources. It always specifies visibility of the resource to a particular user.

The Write and Execute permissions can however be interpreted differently by different resources. This section provides a summary of what those permissions mean for the different types.

E.2.3.2.1. ByteIO Files and RNS Directories

For ByteIO files and RNS directories in the GFFS, the write permission simply indicates that a user can change the contents of the file or directory. The execute permission is not really used internally for files and directories, but could be set for use within FUSE mounts (to make a grid file executable when mounted on a Linux filesystem).

E.2.3.2.2. Queue Resources

Having write permission on queue resources indicates that the user is an administrator of that queue. Having execute permission gives the user the ability to submit jobs to the queue.

E.2.3.2.3. BES Resources

Having write permission on BES resources indicates that the user is an administrator of the BES. Having execute permission gives the user the ability to directly submit jobs to the BES. Queues also need execute permission on the BES before they can successfully submit jobs to it.

E.2.3.2.4. IDP Resources

Having write permission on an IDP or other STS object in the GFFS indicates that the user is an administrator of that particular entry (but not necessarily of the server providing security services). Having execute permission enables a user to behave as “a member” of an IDP, which is especially relevant for users being members of groups.

E.3. Data Files

Data files that feed into computational results are an integral component of any grid computing software. Genesis II provides a variety of methods for specifying the locations of data files. Most jobs can rely on stage-in and stage-out files that are available via the GFFS. This section describes a number of methods for loading data into, and retrieving data from, the GFFS.

E.3.1. Copying Data Into and Out of the GFFS
The need to access data files arises when a user's job needs input files for computation and when the job produces output files. There are three main approaches for copying resources in and out of the GFFS: using the command-line grid client, using the graphical grid client, and using a FUSE mounted filesystem.

E.3.1.1. Copying Data Files Using the Console Grid Client

Similar to cp in the UNIX operating system, the grid's cp command can copy the contents of multiple source files and directories to a target location. The source files can be any mix of local and grid locations. The target must be a directory, unless the source is a single file to copy to another location.

```
# copy a file from the local filesystem.
grid cp local:/home/drake/File1.txt grid:/home/drake/File2.txt

# copy a grid file to a local file.
grid cp grid:/home/drake/File2.txt local:/home/drake/File1.txt

# copy a folder from the local filesystem to the grid.
grid cp -r local:/home/drake/myDir grid:/home/drake/newPlace

# copy a folder from the grid to the local filesystem.
grid cp -r grid:/home/drake/myDir local:/home/drake/newPlace
```

Note that many commands, such as cp, assume the “grid:” prefix if is not provided. For local paths, the “local:” prefix (or the synonym of “file:”) must be used.

E.3.1.2. Copying Data Files Using the GUI Client

The grid client-ui tool has recently been updated for a variety of methods of copying data files, including drag&drop functionality. These may be helpful for users more familiar with graphical user interfaces.

To copy files into the grid with the client-ui, first start the GUI:

```
grid client-ui
```

When the graphical client is running, a window similar to the one below is displayed. The window shows a view of the grid filesystem (labeled as RNS Space) and a view of the ACLs for the object currently focused in the tree.
E.3.1.2.1. Drag&Drop Files Into the Grid

The client-ui supports dragging and dropping files into the grid using the standard file browser application for the user's operating system. On Windows, Windows Explorer (explorer.exe) is the recommended browser, and on the Mac, the Finder is recommended. For Linux, the Nautilus or Konqueror applications can be used for file browsing.

Once the file browser has been opened, one performs drag and drop copying by dragging the file or directory of interest out of the file browser and into the grid tree (in the RNS Space tab of the client-ui) at the desired location. A progress dialog will open and show as the files and directories are copied.

E.3.1.2.2. Drag&Drop Files out of the Grid

The grid client-ui can also copy files to the operating system’s file browser via drag&drop. In this case, the user drags the file or directory of interest from the RNS tree view in the client-ui into the desired folder in the file browser.
There is an important caveat for dragging files out of the grid. Drag&drop defines that the drop may only occur when all the files to be dropped are available locally. In the case of the grid’s client-ui, making the files available locally involves copying them to a temporary location in the local filesystem. Once copied, the files can be dropped into the desired location.

This impacts the behavior for drag and drop significantly. The user must wait until the icon changes to the operating system’s “drop okay” icon before letting go of the mouse. If the contents to be dropped are sizeable, then the copy process can take quite a while, and the user must hold the mouse button down that entire time. In the case of larger transfers, it is recommended to use the “Save To” technique from the next section instead of drag&drop.

E.3.1.2.3. Copying Files out of the Grid with “Save To”

Due to the potential for large data files to cause unacceptable delays in a drag&drop operation, the grid client provides another method to copy files and directories in and out of the grid. This feature is used by right-clicking on a grid path (e.g. a directory) that is to be copied and selecting either the “Copy to Local File System From GFFS” or the “Copy From Local File System to GFFS” option. The former will open a directory browser for the local file system. The user selects the target location and hits “save”. When copying to the GFFS a GFFS directory browser is opened and the user selects the target location in GFFS. When the target location is selected, a dialog opens and shows the copy operation's progress.

The advantage of this feature is that the contents do not need to be copied locally before the operation can be started, unlike drag&drop. The user simply selects where the data files should be saved, and the client-ui manages the copying process after that point.

Directory Operations

When a directory is highlighted, the follow options are available from the drop-down Directory Menu:

![Directory Context Menu]

**Figure 13. Directory Context Menu**

E.3.1.3. Copying Data Files Using a FUSE Mount
FUSE is a method for mounting the grid filesystem onto a local path, so that a portion of the grid namespace is available on the user's computer. This enables the user to copy data to and from the mounted grid directory as if it were present in the local filesystem.

Creating a FUSE mount is detailed in the next section. But using a FUSE mounted GFFS to copy data files is very simple. Assuming the grid has been mounted at /home/drake/gridfs, the following will copy a directory tree in or out of the grid:

```bash
# copy a directory hierarchy up into the grid.
cp -r {/a/directory/tree/} {/home/drake/gridfs/home/drake/newDir}
```

Note that when the gridfs is mounted at the root folder of the grid, the extra /home/drake path is necessary to get down to the user's home directory.

```bash
# copy a hierarchy down from the grid to local filesystem.
cp -r {/home/drake/gridfs/home/drake/toCopy} {/local/path/for/directory}
```

Note that the commands above use just cp and not grid cp, because in these cases the operating system's native copy command is used.

### E.3.2. Exporting Local Filesystems to the Grid

The GFFS provides a feature called “exports” for sharing data into the grid. Exports allow data to reside on one's own machine, but be shared with other users and used as staging data for job processing. This may be very helpful for large data sets, where one does not want to make a secondary copy of the data; the original data can be served on demand within the grid.

A simple export command to share a path under one's local home folder might resemble this:

```
grid export --create \n /resources/xsede.org/mason.iu.xsede.org/containers/mason-gffs \n local:/home/xd-fred/myData grid:/home/xsede.org/fred/mason-data
```

In the above, my local path on the Mason machine is “/home/xd-fred/myData”. This folder will show up in the GFFS grid at the path “/home/xsede.org/fred/mason-data”. This is relying on a container that is already established at Mason and which is linked in the grid at “/resources/xsede.org/mason.iu.xsede.org/containers/mason-gffs”.

The GFFS exports feature is supported by two different web services with varying properties and is a fairly large topic. The exports feature is covered in detail in Appendix M.

### E.3.3. How to Mount the GFFS via a FUSE Filesystem

Genesis2 provides a technique for mounting a portion of the grid namespace onto a local computer. This relies on the FUSE subsystem, which allows user-space drivers to manage filesystems, rather than needing the kernel to manage the filesystem. FUSE enables the user to copy files in and out of the mounted directory as if it were simply another directory in the local filesystem.

To fuse mount the top level of the GFFS onto a local path:
grid fuse --mount local:{/local/path} &

This makes the root folder of the GFFS available as the local path specified.

To fuse mount a specific folder in the GFFS locally, use the “sandbox” flag.

grid fuse --mount --sandbox={/path/in/grid} local:{/local/path} &

The “--sandbox=X” portion of the command specifies where the fuse mount should be rooted in the GFFS RNS tree.

After the fuse mount is created, the user can copy files using the /local/path. Most file and directory operations provided by the operating system can be used on the contents of the path.

To unmount the fuse mounted directory:

```
# Unmount using the grid command.
grid fuse --unmount local:{/local/path}

# This alternative OS-level command can be used on Linux.
fusermount -u /local/path
```

E.3.3.1. How FUSE Mounts Are Different From Unix Filesystems

The FUSE mounted grid filesystem does not behave exactly like a standard Unix filesystem. It does support most standard operations (copying files & directories, deleting them, and so forth), but there are a few caveats described in the next sections.

E.3.3.1.1. No Replacements

One important distinction is that the Genesis II FUSE filesystem does not currently support overwriting a directory with a move (mv) operation. Due to the GFFS representation of files and directories as EPRs, the meaning of substituting out an RNS folder in that way is not well defined. Genesis II requires that a directory can only be moved onto a target location in a FUSE mount if that location does not already exist. This may require some special treatment in scripts using FUSE such that the existing directory is deleted before a directory with the same name is moved into that location.

E.3.3.1.2. No Links

The standard Unix filesystem feature of symbolic links does not operate as expected inside of FUSE mounts. This is due to the basic difference in mechanisms providing the filesystem between the Unix local filesystems and the mounted grid filesystem. Links do exist in the grid, but they are an entirely different creature from the filesystem symbolic links.

Due to that implementation difference, making a link from the FUSE client side between grid asset A and grid asset B will not work. Linking local asset A into grid asset B also will not work, because the grid still does not interpret a symbolic link properly in the FUSE mount. But it is possible, however, to link from grid asset A in a FUSE mount into a local filesystem asset B. Asset B will remain usable as long as the FUSE filesystem is mounted.
E.3.3.1.3. Permission Differences

Another important distinction between Genesis II FUSE filesystems and the standard Unix filesystem is that not all permission attributes are used. In the standard filesystem, permission attributes are usually structured as User/Group/Other triples of Read/Write/exec ACL settings (e.g. rwx|rwx|rwx for user|group|other). These control what the user owning the file can do to it, what other members of the file's group can do to it, and what the general populace can do to the file.

In Genesis II FUSE, the “group” RWX is not used at all. The group portion of ls listings will always show up as ‘---’ for the group portion. This is due to the different interpretation in Genesis II of groups versus the Unix interpretation. Group access control is managed uniformly with user access control in Genesis II.

The “other” portion of the permissions is also slightly different. Genesis II uses the other permissions to describe the rights for “everyone” on the file, so that is quite similar to the Unix interpretation. But Genesis II only allows the permissions to be changed if the user who mounted the grid with FUSE has write permissions on the file, whereas merely being the file’s owner enables changing permissions in Unix file systems. Because of this difference, users should never take away their write permissions on their own files and directories in FUSE mounts, or they lose the ability to give write permissions back again.

E.3.3.2. Operating System Dependencies for FUSE

The FUSE file system for the Genesis II GFFS is only available on Linux operating systems.

If FUSE is not provided by the Linux distribution as a default, these steps may be needed to install it:

For Centos:

```
sudo yum install fuse fuse-libs
sudo chmod a+rx /bin/fusermount
```

For Debian/Ubuntu:

```
sudo apt-get install fuse fuse-utils gvfs-fuse libfuse2
```

E.3.4. Other Staging Methods for Data Files

Many compute jobs can rely directly on the GFFS for staging data files. However, there are cases where the data must remain at the original location rather than being copied to or exported from the GFFS. For these cases, the grid’s job-tool application supports other stage-in and stage-out server types. These types include using web servers, ftp servers, and ssh-based servers (with either scp or sftp protocol) for staging in data files. These types also support data file stage-out except for web servers, which only support data file stage-in operations.

More information about creating JSDL files is available in the section on Submitting Jobs.

E.4. Grid Commands
The Genesis II software offers a number of methods for issuing commands to the grid. One method is to run the grid client program (called “grid”) and enter commands manually or via a script. Another method to issue commands is to write an XScript file with grid commands in an XML format.

**E.4.1. Grid Command Set**

There are quite a few commands available to users in the grid client. A list of the available commands can be printed by issuing the command grid help. Many of the commands will be familiar to Unix and Linux users, but some are very specific to the Genesis II grid.


**E.4.2. Grid Paths: Local vs. RNS**

Before discussing the various ways commands may be executed through the Genesis II client interface, it is important to understand the distinction between local resources and grid resources. The grid client can perform many analogous commands on grid resources (like ByteIO and RNS services) and local resources (files and directories). For example, the cat command, which is used to output the contents of a ByteIO resource, can also output the contents of a local file. Similarly, using the ls command on an RNS service will list the RNS entries contained by that service, while that same ls command used on a local directory will list that directory’s contents.

Distinguishing between grid and local resources is accomplished by prefacing the path of the resource with a prefix to denote its location.

**E.4.2.1. Local Resources**

For resources on the local system, preface the path (in the local file system) with local: or file:, as in the following example:

```
ls local:/home/localuser
```

This will cause the ls tool to list the contents of the directory /home/localuser on the local file system. The prefixes local: and file: are interchangeable; that is, they have the same semantic meaning, and users may use either or both according to preference.

**E.4.2.2. Grid Resources**

For resources in the grid namespace (the GFFS), preface the RNS path with grid: or rns:, as in the following example:

```
ls grid:/home/griduser
```

This will cause the ls tool to list the contents of the RNS entry /home/griduser in the grid namespace. As with the local equivalents, the prefixes grid: and rns: are interchangeable; that is, they have the same semantic meaning, and users may use either or both according to preference.
E.4.2.3. Combining Local and Grid Prefixes

Some commands available to the grid client require multiple arguments, and in such cases it may be useful to mix grid and local resource prefixes. For example, suppose the user wishes to copy a file example.txt from the local file system into the grid, creating a new ByteIO resource with the contents of that file. The cp command can be invoked for this purpose as follows:

```
cp local:/home/localuser/example.txt grid:/home/griduser/example-grid.txt
```

This will instruct cp to copy the contents of /home/localuser/example.txt on the local file system into a grid ByteIO resource named example-grid.txt listed in the RNS resource /home/griduser. The semantics of the command will adjust to reflect the locations of the source and destination provided.

Note that the default is the grid namespace, i.e., /home and rns:/home are equivalent.

E.4.3. Scripting the Grid Client

One of the features of the grid client is the ability to invoke the client to execute a single grid command and then exit without further user interaction. For example, from the local command line, the user may enter

```
grid ls /home/griduser
```

This will start the grid client, execute the command ls /home/griduser, and then print the results of the command to the screen and return to the local command line prompt. If the command requires user interaction, the standard input and output streams will work in the standard way; this means that the standard input can be redirected to a file using the local operating system's existing semantics.

This feature is particularly helpful for performing multiple non-interactive commands in succession through scripting on the local command line. The user may write useful scripts, which can invoke commands on both the local system and in the grid, in whatever scripting dialect is already available. Take the following example, written for Linux's bash:

```
#!/bin/bash
# example.sh: scripting local and grid commands
echo "This is a local command"
for I in {1..5}; do
    str="This is grid command number $I"
    grid echo "$str"
done
echo "End of script"
```

In the example script, the local operating system is instructed to print a message, then to loop over the values 1 to 5, assigned to the variable I. For each of these loop iterations, a string variable str is
composed, and a grid command to echo the contents of that variable is invoked. Finally, the local echo command is used to signal the end of the script.

In this fashion, command-line scripting may be employed to create arbitrarily complex series of commands, mixing local and grid commands as needed.

E.4.4.  XScript Command Files

The XScript scripting language is an XML-based scripting language developed by the Virginia Center for Grid Research (then the Global Bio Grid research group) at the University of Virginia for use with Genesis II. Originally the language was designed to support only minimal capabilities – enough to get the project started until something better could be developed – but it has since grown into a more sophisticated and fully featured language in its own right. Today, the XScript language supports many of the language features that are expected from a real programming language, including loops, conditionals, and exceptions.

XScript is used to script commands from within the grid client, as opposed to the previous section which discussed running scripts that repeatedly invoked the grid client to execute commands. This section will provide an overview of the features and use of XScript; a complete documentation of XScript is available in the Documentation section of the Genesis II wiki (which is available at http://genesis2.virginia.edu/wiki/Main/XScriptLanguage Reference).

E.4.4.1.  XScript High-level Description

In XScript, every XML element (other than the root document element) represents a single language statement. These statements may or may not themselves contain other statements depending on the element type in question. For the most part, those statements which can support inner statements are the language feature elements such as conditionals and loops, while those that cannot generally represent simple statement types like echoes, grid commands, and sleep statements.

In XScript, every XML elements falls into one of two categories. The first category is for language elements and uses the first namespace shown in the figure below, abbreviated as gsh. The second category is for Genesis II grid commands and uses the second namespace shown in the figure, abbreviated as geniix. We will use the first of these, gsh, as the default namespace for all XML in this section and thus assume that the root element of all XScript scripts looks like the following:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<gsh:script
    xmlns:gsh="http://vcgr.cs.virginia.edu/genii/xsh/script"
    xmlns:geniix="http://vcgr.cs.virginia.edu/genii/xsh/grid"
    xmlns="http://vcgr.cs.virginia.edu/genii/xsh/script">
    ...
</gsh:script>
```
E.4.4.2. XScript Language Elements

XScript has been designed to include most of the control flow structures used in modern programming languages. There are also command elements common to many scripting languages, such as "echo" and "sleep". The following is a list of the basic control elements and commands available in XScript. Note that this list is subject to change as the language matures or additional features are added.

For usage of a specific element, or the particular semantics of its use, see the external documentation on the Genesis II wiki.

- Echo – Prints to the terminal
- Define – Defines a new variable
- Sleep – Pause script execution
- Exit – Terminate the script
- Param – Indicate parameters to grid command elements
- Comparisons (Equals, Matches, Compare) – Operators for comparing variables
- Conditionals (And, Or, Xor, Not, IsTrue, IsFalse) – Operators for manipulating Boolean variables
- If, Then, Else – Conditional execution
- Switch, Case, Default – Choose from a set of values
- For, Foreach – Loop over an index variable or a set of values
- Throw, Try, Catch, Finally – Exception handling statements
- Function, Call, Return – Defining and using functions/subroutines within the script
- Parallel-job, Parallel – Allow parallel execution of collections of statements

E.4.4.3. Grid Command Elements

The simplest form of statement in an XScript script is a grid command. Grid commands are identified by belonging to the geniix namespace. Any time an XML elements exists in this namespace, the XScript engine attempts to find a grid command with the same name as the element’s local name. If it finds such a command, the statement is assumed to represent that command, otherwise an exception is thrown. Parameters (command-line arguments to the grid command) are indicated with XScript param elements. Below we show example grid commands in the XScript language for the grid commands ls (list the contents of a RNS directory) and cp (copy files/resources to another location).
E.4.4.4. XScript Variables/Macros

Every attribute value and text content node of an XScript script can include a reference to a variable. If included, the value of this variable will be inserted at run time as a macro replacement. Further, variables are scoped by their statement level. This makes it possible to write scripts that contain multiple variables of the same name without additional variable definitions interfering with outer definitions.

Variables in XScript documents are indicated by surrounding the variable name with `${ and }`. Thus, to indicate the value of the NAME variable, the string `${NAME}` should appear anywhere that text was expected (such as for an attribute value or as the text content of an appropriate XScript statement).

Arrays are also supported in the XScript language, though at the time of the writing of this document, only for accessing parameters passed in either to the script itself, or to functions. The length of an array in XScript is indicated with the `${ARRAY_VARIABLE}` expression syntax, while the elements inside of the array are indicated with the `${ARRAY_VARIABLE}[INDEX]` syntax. Thus, to echo all elements of the ARGUMENTS array, the following XScript code can be used:

```
...<for param-name="i" exclusive-limit="${ARGUMENTS}"/>
   <echo message="Argument ${i} is ${ARGUMENTS[${i}]}."/>
</for>
...```

Arguments passed in to the script as well as those passed in to functions are contained in the ARGV array variable (for command-line arguments passed in to the script, the first element is the name of the script file itself).

E.4.4.5. An Example XScript

Below is a complete example XScript script. The functionality of the script is trivial, but the file is syntactically correct, and provides a concrete example of some of the concepts discussed previously in this section. The script takes a single argument from the command line, which it compares to a set of `switch` cases, and then executes a different grid command based on that input (along with a
few **echo** statements for good measure). Note the **if** test at the offset to determine if a command-line argument was provided. We will call this example file **example.xml**.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<gsh:script
    xmlns:gsh="http://vcgr.cs.virginia.edu/genii/xsh/script"
    xmlns:geniix="http://vcgr.cs.virginia.edu/genii/xsh/grid"
    xmlns="http://vcgcr.cs.virginia.edu/genii/xsh/script">
  <condition property="NOARGS">
    <compare numeric="true" arg1="${ARGV}" arg2="2" comparison="lt"/>
  </condition>
  <if test="NOARGS">
    <then>
      <echo message="You must include at least one argument"/>
      <exit exitcode="1"/>
    </then>
  </if>
  <echo message="Starting the script"/>
  <for param-name="i" exclusive-limit="${ARGV}">
    <echo message="Argument $i is ${ARGV[$i]}."/>
  </for>
  <switch value="${ARGV[1]}">
    <case pattern="who">
      <geniix:whoami/>
    </case>
    <case pattern="where">
      <geniix:pwd/>
    </case>
    <case pattern="jobs">
      <geniix:qstat>
        <param>/queues/grid-queue</param>
      </geniix:qstat>
    </case>
    <default>
      <echo message="What do you want to know?"/>
    </default>
  </switch>
  <echo message="Script complete"/>
</gsh:script>
```

**E.4.4.6. Running XScript Scripts**
Before we describe how to execute a script, a word about Genesis II's script handling is in order. Genesis II supports multiple scripting languages through the use of the Java Scripting API. In order to differentiate between the various scripting languages, Genesis II uses filename extensions to determine the correct language to use when running scripts. Thus, to run a JavaScript script, the filename must end in the .js extension. Similarly, to run an XScript script file, the filename must end with the .xml filename extension.

To execute a script within the Genesis II client, use the script command, passing in the path to the script and any parameters to the script. For example, if the example script above were located at the RNS path /home/griduser/example.xml, the following command would launch the script with an input parameter of who:

```
grid script /home/griduser/example.xml who
```

E.5. Submitting Jobs

The main point of any grid software is to provide a means for processing computational jobs on the compute resources that are available in the grid. This is true for Genesis II also; many features are provided for creating jobs in JSDL, sending them to a grid queue or BES, and managing the jobs while queued. This section discusses the basics of creating a job and submitting it for processing.

E.5.1. How to Create a JSDL file

The purpose of a JSDL file is to specify a compute job in terms of the executable that the job should run, the resources that it will consume in terms of memory and CPU, and any special requirements for processor type or other attributes. The JSDL specification requires that the file be stored in XML format with particular elements and attributes for specifying job attributes. This makes it fairly difficult and unpleasant to write JSDL files from scratch. One common way to generate a new JSDL file is to change an existing well-formed JSDL file to fit the purpose under consideration.

A better way to generate a JSDL file is to use the Genesis II JSDL file creation tool to specify the job's requirements. This is available as a standalone install called the Grid Job Tool (located at http://genesis2.virginia.edu/wiki/Main/GridJobTool). This provides versions for most common operating systems. Alternatively, the job-tool is also provided by the Genesis II client installation, and can be executed this way:

```
grid job-tool
```

It can also be executed by right-clicking on an execution service such as a BES or Grid Queue and selecting “create job”.

E.5.2. Using the Job-Tool

From within the client-ui RNS Tree view, select the directory where the JSDL project file should be located, or select the execution container (BES or queue) where the job should be executed. Right click on that location and select 'Create Job'. The tool has provisions to give the job a name and
description. Any arguments that the executable or script needs for running the job can be provided in the first tab (under Basic Job Information).

![Job tool basic information tab.](image1)

In the data tab, the data to be staged in/out can be provided (see figure below). It is worthwhile noting here that data files being staged in&out are usually done via the GFFS, and thus some BESes that do not support the GFFS may need to use other stage-out types than grid: paths (such as data files on a local file system or web server). These can also be specified in the data tab.

![Job tool data staging tab.](image2)

The other major component for the job-tool is the resources tab, where any specific expectations of the job in terms of hardware configurations and preferred operating system can be specified. This is depicted in the figure below.
E.5.3. Submitting a Job to a Grid Queue

The qsub command is used to submit a new job to a queue for processing. Although jobs may be submitted to a BES (and bypass a queue), submitting to queues is recommended since it allows better resource allocation and job handling.

```
# submit a job to the queue, with a job description file.
qsub {/queues/queuePath} local:/path/to/job.jSDL
```

The qsub command returns a job ticket number after successfully submitting the job. This ticket number can be later used to query the job, kill it, and so forth.

E.5.4. Controlling or Canceling Jobs in a Queue

The qkill command allows grid users to terminate any managed job (not already in a final state) that they previously submitted. To kill one job in the queue, use:

```
grid qkill {/queues/theQueue} {jobTicket#}
```

The ticket here is obtained when a job is submitted using anyone of the recommended methods. The qreschedule command is used to return an already-running job back to the queue and ensures it is not rescheduled on the same BES. The slot count for this resource must be manually reset later. This command is useful when the Queue consists of BESes which interface to a queuing system like PBS. A job may be in the Running state on the grid, but in a Queued state on the back-end PBS. Such a job can be moved to an alternate BES where it can be executed immediately. To reschedule a job:
Both qkill and qreschedule have variants that allow multiple job tickets to be killed or rescheduled with one command.

### E.5.5. Cleaning Up Finished Jobs

The queue manages all jobs that are submitted to it from the time that they are submitted until the time that they are executed, or have failed, or are cancelled by the user. Even jobs in the final states of FINISHED, CANCELLED, or FAILED are held onto by the queue until they are cleaned up. The process of cleaning a no-longer active job out of the queue is called 'completing' the job.

Completing a job performs the garbage collection of removing the job from the queue.

```
grid qreschedule {/queues/theQueue} {jobTicket#}
```

Both qkill and qreschedule have variants that allow multiple job tickets to be killed or rescheduled with one command.

### E.5.6. The Queue Manager in Client-UI

After the client-ui has been launched, the "Queue Manager" can be opened to control jobs in the queue or to change the queue’s characteristics given sufficient permissions. The figure below shows the client-ui about to launch the queue manager on a selected queue:

```
grid qcomplete {/queues/queuePath} --all

# Removes all jobs that are in a final state
# (i.e., FINISHED, CANCELLED, or FAILED) from the grid queue.
grid qcomplete {/queues/queuePath} {ticketNumber}

# Removes a specific job from the queue, where the ticketNumber
# is the job-identifier provided at queue submission time.
```

Figure 17. Launching the queue manager.
After launching, the queue manager window will appear as depicted below:

Figure 18. Queue manager’s job list.

In the first tab, called Job Manager, the queue manager shows the current set of jobs that are in the selected queue. The jobs can be in a number of non-final states, such as QUEUED and EXECUTING, or they may be in a final state, such as FINISHED or FAILED.

The second tab of the Queue Manager, called Resource Manager, shows the resources associated with the queue. The view presents what is known about the BES resource, in terms of the operating system and other parameters. This tab can only be modified by a user with permissions on the queue, and the “Max Slots” is the only part of the tab that is modifiable. The number of slots controls how many concurrent jobs the resource is expected to handle, and the queue will allow at most that many jobs onto that particular resource. An example Resource Manager is shown below:

Figure 19. Queue manager’s resource tab.

To control jobs that are in the queue, look at the Job Manager window again. When a job is selected in that view (with a right-click), a context menu for controlling that specific job is displayed. This is shown in the figure below:

Figure 20. Removing a job from the queue.

Using the choices available, a user can stop the job with “End Jobs”, clear up finished jobs with “Remove Jobs”, and examine the “Job History” for the job. Job History brings up the following window with information about the job selected:
This history shows that the job had been added to the queue, and then that it had been submitted to a PBS called “pbs-long-india”. The job was last seen being processed on that BES.

**E.5.7. Job Submission Point**

A user can also submit jobs by copying the jsdl files into the “submission-point” directory under the queue. This is an extremely simple method for job submission, and the jobs submitted this way still show up in the `qstat` command.

```plaintext
grid cp {local:/path/to/job.jsdl} grid:{/queues/queuePath}/submission-point
grid qstat {/queues/queuePath}
```

A sample return using the above method

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Submit Time</th>
<th>Tries</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF2DD56D-B220-FFA8-8D35-589F65E016DE</td>
<td>16:21 EDT 05 Jun 2012</td>
<td>1</td>
<td>QUEUED</td>
</tr>
</tbody>
</table>

**E.5.8. Submitting a Job Directly to a BES**

Another method to run a job is to submit the job directly to the BES. This is a helpful method for testing jsdl files as they are being developed, or when the user is sure that the BES supports the requirements of the job:

```plaintext
grid run --jsdl={local:/home/drake/ls.jsdl} {/bes-containers/besName}
```

The above command is synchronous and will wait till the job is run.

There is an asynchronous variant that will allow job status notifications to be store into a file in the grid namespace. Note that this feature is only available for Genesis II BES currently, and is not
supported on the Unicore BES. The user can check on the status of the job by examining the status file. This is an example of an asynchronous direct submission to the BES:

```
grid run --async-name=/path/to/jobName \ 
   --jsdl={local:/home/drake/ls.jSDL} \ 
   {/bes-containers/besName}
```

In the above, the command returns immediately after submission. The job’s status is stored in the file specified by the grid path /path/to/jobName. Eventually this file should list the job as FINISHED, FAILED or CANCELLED depending on the circumstances.

### E.5.9. How to Run an MPI Job

To run an MPI job, the JSDL file needs to specify that the job requires MPI and multiple processors. The job executable file needs to have been compiled with an MPI library (i.e. MPICH, MVAPICH, OpenMPI).

When using the Genesis II JSDL file creation tool, these job requirements can be specified under the “Resource” tab (depicted in the figure below). The “Parallel Environment” field permits selection of the MPI library (i.e. MPICH1, MPICH2) that the executable was compiled with. The “Number of Processors” field lets the user specify how many total processes are needed to run the job. The “Process per Host” field lets the user specify how many of these processes should be run per one node.

![Image of resource tab settings](image.png)

If the user manually creates a JSDL file, the JSDL SPMD (single program multiple data) Application Extension must be used to define the requirements of the parallel application in JSDL. Please
consult the specification document for details. The SPMD application schema essentially extends the POSIX application schema with four elements: NumberOfProcesses, ProcessesPerHost, ThreadsPerProcess, and SPMDVariation. The NumberOfProcesses element specifies the number of instances of the executable that the consuming system must start when starting this parallel application. The ProcessesPerHost element specifies the number of instances of the executable that the consuming system must start per host. The ThreadsPerProcess element specifies the number of threads per process. This element is currently not supported by the Grid Job Tool. The SPMDVariation element defines the type of SPMD application. An example of a parallel invocation using the “MPICH1” MPI environment is provided below.

```xml
<jsdl:Application>
  ...
  <jsdl-spmd:SPMDApplication>
    <jsdl-posix:Executable>a.out</jsdl-posix:Executable>
    <jsdl-posix:Input>input.dat</jsdl-posix:Input>
    <jsdl-posix:Output>output.dat</jsdl-posix:Output>
    <jsdl-spmd:NumberOfProcesses>8</jsdl-spmd:NumberOfProcesses>
  </jsdl-spmd:SPMDApplication>
  ...
</jsdl:Application>
```

**E.6. Client GUI**

The Client UI is windowed application that provides a graphical view of grid RNS space; it is launched by running `grid client-ui` command.

**E.6.1. Client GUI Basics**

**Terminology**

- RNS – Resource Namespace Service
- XCG3 – Cross Campus Grid, Version 3
- (G)UI – (Graphical) User Interface
- EPR – Endpoint Reference
- ACL – Access Control List
- XML - Extensible Markup Language
- JSDL – Job Submission Description Language
In this document, we will use the XCG3 grid as an example to explain Genesis II client UI features. Refer to Figure 12Figure 23 for an example of the Client UI.

In this window, you will see:

- 7 menus (File, Edit, View, Jobs, Security, Tools and Help) on the top
- Left Panel named RNS Space with grid name space represented as tree structure
- Right Panel with resource information tabs (Security, Resource Properties and EPR Display) in it. Error information text box at the bottom.
- Recycle bin icon in the right panel (Bottom right)
- Tear symbol icons in both right and left panels (Top right, looks like torn paper).
- Username/Password Token Text boxes
- Pattern Icon and Everyone Icon
- Credential Management Button

Tabs, Menus and their options will be represented as Tab/Menu->SubTab/SubMenu->option in this document. Reference to grid commands will be in the format grid command.

E.6.2. Credential Management
A user can manage their grid credentials by clicking on Credential Management button in the client-ui window and selecting appropriate options (Login, Logout or Logout all). Click on Credential Management->Login->Standard Grid User tab, a separate window will pop up prompting for username, password and grid path. This will log you into grid using your grid credentials (not same as grid xsedeLogin), refer to Figure 24. If you select Credential Management->Login->Local keystore tab, you can login using a keystore file. Select the keystore file (usually .pfx form) from your local file system and enter password for it. You can also login using username/password Token by selecting Credential Management->Login->Username/password tab.

Figure 24. Credential Management->Login->Standard Grid User

You can see your current login credentials when you hover the mouse pointer over Credential Management button. Refer to Figure 25.
Figure 25. Showing grid credentials using mouse hover

You can logout of grid by Selecting Credential-Management -> Logout option where you can select which credentials you want to logout as. This is helpful if you have multiple credentials in your credential wallet and you want to logout of specific credential. Refer to Figure 26.
Alternately, you can logout of all your grid credentials by selecting Credential-Management->Logout All option.

E.6.3. Client UI Panels and Menus

E.6.3.1. RNS Space (Left Panel)

Here grid name space is presented as tree structure with root of the name space represented by '/' and other sub-directories below it. You can browse the tree by clicking on toggle symbol next to the resource. You can select a resource simply by clicking on the resource. Clicking on a resource highlights the resource and you can see security information in Right panel change accordingly. You can also view Resource Properties and EPR Display of the resource on the Right panel. You should at least have 'Read' permissions on a resource to view its security and other information. If you do not have at least Read permissions you will get an error in error panel below (Ex. No Authorization info for target path: {grid resource name}). Launch grid client-ui, login as grid user and then browse the RNS tree (highlighted using red box in Figure 27) by clicking on the toggle symbol next to root directory '/' (and then descend down expanding toggle symbol). This will expand the tree; you can now browse to your grid home directory or any or any other grid resource that you have permissions on (at least read permissions). You
can also minimize the tree (if already expanded) by clicking on the toggle symbol next to the grid resource. If you try to browse a resource without read permissions on that resource, you will get an error message in Error Messages Box (Highlighted using Blue box) in Figure 27.

![Figure 27. Major User Interface Panels](image)

**Dragging an RNS Resource to Trash**

To delete a RNS resource (such as files, directories etc), browse the RNS tree structure and select the object you want to delete. Then still holding the mouse click, drag the mouse to recycle bin and release. This is depicted below in the figure where the red-circled resource on the left will be dragged to the trash can on the bottom right.
Figure 28. Drag RNS Resource to Trash
E.6.3.2.  Right panel

Here you will find 3 tabs; Security, Resource properties and EPR Display. This is Highlighted using Green box in Figure 27.

Security tab: This is selected by default when you first open the client-ui. This tab will display read/write/execute ACLs for selected resource. More information on grid ACLs can be found in section E.2.3. If you grant read/write/execute ACLs on a resource and refresh the Client-ui, the new permissions will be seen in respective ACL text box after refresh.

There is also Username/Password Token sub-panel, this is used to issue username/password access on selected resource to users. These users may or may not have a grid account, all they would need is to have Genesis II client installed and username/password information to access that resource (of course if the resource is in the subtree, they should have access to browse to that part of tree structure).

![Drag-and-drop a user to ACL list on a resource](image)

Figure 29. Drag-and-drop a user to ACL list on a resource

Drag and Drop Permission Management

You can give permissions to everyone (grid and non-grid users) on a selected resource by dragging and dropping Everyone icon onto that ACL box ie. read/write/execute text box. You can grant access to individual grid users using two methods, using grid chmod command in grid shell or drag-and-drop
method in UI. In the client-ui window select the resource you want to grant permissions on. You should have write (or admin) permissions on that resource to be able to grant R/W/X access to other users. Locate ‘Tear’ icon on the left panel (tear icon on right top corner, looks like torn piece of paper), left click on it and drag it while you are still clicking on it. This will create another window showing the tree structure, browse to /users directory in the new window and select by left clicking on the username you need. Now drag that username and drop it onto read/write/execute text box in the main client-ui window. In Figure 29, the tear icon, grid resource (hello file) and username (/users/andrew) on new window and write ACL text box are highlighted. You can select the resource in the RNS tree and it should now have the new credentials listed in the corresponding credential text box in the right panel.

**Dragging ACLs to trash**

Browse the RNS tree structure and select the resource (File/Directory) on which you want to modify the ACLs. Then on the right security panel, select the ACLs from read/write/execute and still holding the mouse click, drag the mouse to recycle bin and release.

![Figure 30. Dragging ACL Entry Into Trash](image)

**Resource Properties tab:** will display detailed resource information about a selected resource. It includes, the resource type, Permissions, Creation Time, Resource Endpoint Reference (Address where the resource is physically located) etc.

**EPR Display tab:** will only Address, Reference Parameters and Metadata for that resource.
E.6.3.3. Menus

Figure 31. Changing UI Shell font and size

File Menu: Drops down to present multiple selection options and most options here are intuitive. Selecting File->Preferences option will open another frame where you can set your client-ui and shell preferences. After every change, make sure you refresh the client-ui by selecting View->Refresh. Some of the options include Font size and style in the client-ui's shell window. File->Quit menu option will quit the client-ui window. Selecting File->Make Directory will create a new directory and File->Create New File will create a new file in grid name space. Select File->Preferences->Shell and select the font style and change the font size (up arrow to increase size and down arrow to decrease size) and click on OK. Launch grid shell (Tools->Launch Grid Shell) and type a grid command (Ex. grid ls). You can see the changes in font style and size in this grid shell window. Refer to Figure 31.

You can view a resource’s security ACL information at low, medium or high level. Select File->Preferences->Security->HIGH. Refresh client-ui by selecting View-Refresh (or F5 button on your keyboard). Refer to Figure 32. Select any grid resource that you have at least read permissions on, in the right panel ACL text box you can now see ACL information in more detail.
If you select low level, your ACL information will just list the users in the ACL box. If you select medium you can see additional information like what type of resource it is and some additional information on user's ACLs. By selecting High Level, you can see more information about ACLs like Identity type, Resource Type, DN etc. This is shown in Figure 33.
Select File -> Preferences -> Resource History. Set the job's history information level to desired option Trace, Debug, Information, Warning or Error. Select Trace option, this will provide maximum information about the job. If you just want to see errors or warnings only, select those options. This is useful when a user wants to debug his jobs after submitting them to queue resource. Select a queue resource in the grid RNS name space, select Jobs->Queue Manager. Select a job from the jobs list that you submitted, Right click and select Job History option. In the new window Minimum Event Level is set to Trace (or option that you selected earlier in Step 2).

To get XML display Select File -> Preferences -> XML Display, Either select to view grid resource information as flat XML file or as tree structure. In Figure 34, the Resource Properties are displayed as a tree structure. If you selected File -> Preferences -> XML Display -> Display XML as a tree in step 2 earlier, the information will be displayed as shown.
To Create new file and directory, In RNS tree, select a directory where you have write permissions (Ex. /home/joe). Select File->Create New File option, this will pop up a new window and prompt to enter file name. After entering the file name (Ex. abc.txt), click OK button. New file should be in the RNS directory you selected (Ex. /home/joe/abc.txt). Refer to Figure 35.

Figure 34. Displaying resource information as tree structure
View Menu: View-\>$Refresh$ option can be used to refresh the client-ui window. Click on this option after you make changes to the client-ui window (Ex. changing preferences) to reflect the changes in UI or you create/delete/move/copy new files/directories to grid name space and after refresh client-ui window to reflect the changes. You can also refresh a particular resource by highlighting it and then hitting F5 button in your keyboard but this may depend on how F5 button is configured.

Jobs Menu: From this menu, user can create a new job, view the existing job in the queue resource or view saved job history of a job from the queue. The Jobs-\>$Queue \, Manager$ and Jobs-\>$Create \, Job$ options will be inactive until you select a queue resource from the tree structure in the left panel. Information on how to create job, submit job and check the job status in queue manager can be found in section E.5. You can also create a job using “grid job-tool”, this will open a new window where you can enter you job information. Most fields in the job-tool are intuitive.

To Create Job Select Jobs-\>$Create \, Job$ option. In the new window, create a new job or open an existing one. Submitting a job from this window will submit job to the selected queue resource. Refer to Figure 36 for an example job. Saving new job as project will save the file on your local file system with .gjp file extension.
Figure 36. Job tool, creating simple ls job

Figure 37 below shows how you can enter a project number (such as an allocation number you will get on Kraken) or some other xsede-wide project number. In job-tool, click on the 'Job Projects' text box and you will get a pop-up, click on the ‘+’ sign and you will get another pop-up. Enter the project number and click ‘ok’, click ‘ok’ again and your project number will be in the main job-tool window. Also if you forget to enter a necessary field, such as executable name or data file, you will get a warning/error in the bottom pane of the job-tool window.
In Basic Job Information Tab, Job Name can be any meaning name you want to give for your job. Executable is the executable file that your job will use to run. This can be a system executable like /bin/ls or shell script or MPI program executable, or any other form of executable that can be run (Others include, Java class file, C/C++ executable, Matlab, Namd etc). Arguments list is the list of arguments your executable may need, here it is '-l' option for /bin/ls (essentially /bin/ls -l). You can add arguments by clicking in the '+ ' button in the arguments frame. You can also pass environment variables to your job and this can added by clicking in '+ ' button in Environment frame. If you decide to delete one or more arguments or environment variables after adding them, select that argument/environment variable and click on '- ' button in respective frame.
You can save the job output and error information to files either in grid name space (using grid protocol) or use other protocols (scp/sftp, ftp or mailto) and save them in Data tab of job tool. To save the standard output and standard error from a job, enter the file names in Standard Output and Standard Error text boxes. Refer to Figure 38. Then to save these files to grid or other locations, add the files in Output Stages section and select appropriate Transfer Protocol and corresponding Stage URI path. Note, the file name you enter in Standard output and Standard Error text boxes should match Filename area in Output Stages but these names can change in Stage URI area. The ‘+’ and ‘-‘ buttons are used for adding or deleting an entry. Similarly you can stage in file needed to execute your program in Input Staging frame.
Once a queue resource is selected, you can select Jobs->Queue Manager to view the jobs you submitted to queue and manage resources (if you are the owner of those resources). Selecting Jobs->Queue Manager will open a new window displaying your jobs and resources information. Selecting Jobs->View Job History will open a File browsing frame displaying your machine's local file system (machine where Genesis II client software is running). You should have saved job's history prior to this to be able to select the job history and view it.

To View Jobs, Queue Manager and Job history select the queue resource in the RNS tree and Selects Jobs->Queue Manager, this will open a Queue Manager window showing all the jobs you own on that queue. A new window listing the jobs you submitted or jobs you have permissions to view. Refer to Figure 39.

![Figure 39. Jobs->Queue Manager](image)

To see job history of a particular job, select a job in the Queue Manager window and right click on it. Select the Job history option and you will get a new window with job history for that job. Here you can
select different level of history information from *Minimum Event Level* menu (Trace, Debug, Information, Error or Warning). This can also be set via *File -* > *Preferences-* > *Resource History* tab. Refer to Figure 40.

**Parameter Sweep Job**

To create and submit a parameter sweep job, open the job-tool either by clicking on queue resource or by typing job-tool in grid shell. This will bring up the job tool shown as below.
By default, the tab “Grid Job Variables” is disabled. To add a parameter sweep variable, just use ${var_name} ($ sign followed by open curly brace followed by variable name and close curly brace) in any of the following fields in job-tool.

**Job Name** - e.g. Ls-job-${i}
**Executable arguments** - e.g. /bin/ls dir-name-${i}
**Data Input/Output Stages** - e.g. /home/xsede.org/vana/ls-out-${i}.txt

![Figure 41. Job Definition Using Variables](image-url)
Figure 42. Variable Usage in Output Filename
Once you specify `\{var_name\}` in any one of the above locations, 'Grid Job Variables' tab will be activated and you can define your `var_name` to be either an integer or double or string. You can also specify the starting value, end value and step values (interval) for your variable.

![Figure 43. Defining Job Variable Values](image)

After Submitting the job, the actual values for `\{var_name\}` will be substituted for the `var_name` in all the places (Job name, arguments, File Name) specified in the job. Here's the screen shot of the queue where job-name is substituted for actual integer values (starting from i=1 to i=20). For above example, output files generated will also be `/home/xsede.org/vana/ls-out-1.txt`, `/home/xsede.org/vana/ls-out-2.txt` … `/home/xsede.org/vana/ls-out-20.txt`. 
Tools tab: Selecting *Tools-* > *Launch Grid Shell* will open a shell window where you can run grid shell commands like *ls*, *cat*, *cp* etc. Refer to Figure 45. You can also invoke grid shell directly in command line using *grid shell*. The UI shell interface supports tab completion where as command line shell interface does not support tab completion. More information on grid commands can be found in section E.4.
To launch Shell and list your home directory
Login to the grid using your grid credentials, Launch a grid shell from Tools->Launch grid Shell option. Run `grid pwd` to make sure you are in your home directory (by default you will be in your home directory after you login to grid). Run `grid ls` command and this should list all the files/directories in your grid home directory.

**E.6.4. Drag-and-Drop Feature**

This method can be used for copying data files using the GUI Client. You can select a particular File or directory in the left panel tree structure to copy out of grid, then simply drag it while still clicking and release the mount button to drop File/Directory on to your local computer's File system. Reverse also works, where you can select a file/directory from your local machine and drop it in to your grid name space. For this you will need appropriate permissions on that grid resource i.e write permissions to copy files. Refer to section E.3.1.2 for a detailed explanation.

**E.6.5. File associations**

This helps you to set up the GUI to open files with a particular application. Note that the client-ui has recently been updated to use the launching capabilities of the Operating System. In most cases, the default behavior is sufficient to edit and open assets in the grid. For situations where the default is not sufficient, this section documents how to override the default applications.

The file called `.grid-applications.xml` should go in the user's local file system home directory. This file has the list of programs to launch for some mime types that extend the basic launching support in the
client-ui. On Windows-XP, the home directory will be "c:\Documents and Settings\myUserName" and on Windows7, the home directory will be "c:\Users\myUserName". Note that this file is currently using short names for the first argument, which should be a program name. If you do not have your PDF editor or your Word document editor on the path, you will need to put the full path to the executable for the appropriate editor. The file called .mime.types should also go into the user's home directory. This gives Java an association between file extensions (like .DOC) and the mime type that will be reported for files of that type.

To open a PDF file in grid namespace create .grid-application.xml and .mime.types file, copy them to your $HOME directory or equivalent locations in Mac and Windows. Launch client-ui. Browse the grid RNS space and select a PDF file you wish to open. Double click on the file. File will open Acrobat viewer.

Sample .grid-applications.xml file

```xml
<external-applications>
  <mime-type name="application/acrobat">
    <application-registration type="Common" factory-class="edu.virginia.vcgr.externalapp.DefaultExternalApplicationFactory">
      <configuration name="acroread">
        <argument>acroread</argument>
        <argument>%s</argument>
      </configuration>
    </application-registration>
  </mime-type>

  <mime-type name="application/msword">
    <application-registration type="Common" factory-class="edu.virginia.vcgr.externalapp.DefaultExternalApplicationFactory">
      <configuration name="word">
        <argument>libreoffice</argument>
        <argument>%s</argument>
      </configuration>
    </application-registration>
  </mime-type>
</external-applications>
```

Sample .mime.types file:

```
application/acrobat pdf
application/msword doc docx
```
F. Grid Configuration

This section describes how to create a distributed grid using Genesis II components. The main components of such a grid are: (1) the GFFS, which provides the file-system linking all the components together, (2) the Grid Queues, which support submitting compute jobs to the computational elements in the grid, and (3) the BESes, which represent each computational element. Each of these services lives inside a container, which is a Genesis II installation that provides one or more services to the grid via a web-services interface.

Every GFFS grid has one “root container” that provides the root of the GFFS file system, similar to the traditional Unix file system root of “/”. The remainder of GFFS can be distributed across other containers which are then “linked” into the root container. Usually, the root container serves all of the top-level folders such as /home, /users and /resources.

This chapter will describe the overall structure for the GFFS filesystem and will provide steps for building a new grid, starting with the root container.

F.1. Structure of the GFFS

There is no definite requirement for any particular structure of the GFFS. It starts as a clean slate, with only the root node (’/’). All of the top-level directories are defined by convention and generally exist to provide a familiar structure around the grid resources.

This section will describe the purpose of each of these directories. Note that most of these are created by the GFFS root container deployment, which is described later in this chapter.

The following table documents the basics of the XSEDE namespace that is used for the XSEDE production grid. The full definition of the the XSEDE namespace is provided by SD&I Activity 126 (https://software.xsede.org/viewvc/xsede/sdi/activities/sdiact-126/trunk/Plans/SDIACT-126_XSEDE_Global_GFFS_Namespace_Design-v8final.docx?view=co).

<table>
<thead>
<tr>
<th>XSEDE Grid Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>/resources/xsede.org/containers</td>
</tr>
<tr>
<td>Stores the containers installed on the grid. Each of these is usually a resource fork where a container is linked into the GFFS.</td>
</tr>
<tr>
<td>/etc</td>
</tr>
<tr>
<td>Stores the signing certificate generator for container resource identifiers.</td>
</tr>
<tr>
<td>/etc/resolvers</td>
</tr>
<tr>
<td>Stores the RNS resolvers for the grid that enable fail-over and replication.</td>
</tr>
<tr>
<td>/groups/xsede.org</td>
</tr>
<tr>
<td>Stores group identities for the grid.</td>
</tr>
<tr>
<td>/home/xsede.org</td>
</tr>
<tr>
<td>Stores the home folders for users.</td>
</tr>
</tbody>
</table>
Other grids can use the XSEDE namespace design for their structure, but the portions of the namespace that mention “xsede.org” are replaced by a more locally appropriate name. For example, the new XCG (Cross-Campus Grid) namespace at the University of Virginia has folders for /users/xcg.virginia.edu and /resources/xcg.virginia.edu and so forth. The European GFFS grid has "gffs.eu" in those second tier names. This approach supports federating multiple grids within the same structure; for example, the XSEDE grid can provide a link to /resources/xcg.virginia.edu within the XSEDE grid in order to reach the resources of the XCG grid.

F.2. Deployment of the GFFS

The deployment of the GFFS requires two major components; a set of containers that are deployed on a host or set of hosts, and a deployment configuration package that enables a grid client or container to connect to the GFFS. A “grid deployment package” is a directory of configuration items that is required to connect to an existing grid as a client. This package is also required for configuring a Genesis II Container as a server that allows secure connections. The deployment package is constructed when building the root container. The client is provided a limited version of this package which does not contain any of the private keys used by the root container.

There is a “default” deployment shipped with the source code that contains a basic set of configurations necessary to run Genesis II. A new deployment is created when “bootstrapping” a grid that inherits the “default” deployment. This enables the basic security configuration of “default” to be extended to provide a secure grid.

Below are instructions to create the “Bootstrap” container that serves as the root of the RNS namespace and the primary source of GFFS services. Secondary containers (i.e., not the GFFS root) are created using an installer that contains the deployment package produced during the Boostrap configuration process. Using the installer enables new containers to be deployed very quickly.

Note that the following steps for the Bootstrap Container assume that the grid administrator is working with the Genesis II software as source code, rather than via an installer. When using the Genesis II installer, these steps are not required for setting up clients or secondary containers. Building the installer requires some working knowledge of Install4j, an Install4j license, and the root container’s deployment package (created below). If you would like an installer built for your grid, it is recommended to contact xcghelp@cs.virginia.edu for assistance.

F.2.1. Preparing the Environment for Generating Deployments

The deployment generation process requires a copy of the Genesis II source code (see Section H.2 if you need to obtain the source code and Section H.1 about installing Java and other prerequisites). These steps use the XSEDE tools and test suite for the root container deployment, especially the
deployment generator tool (see Section I for more information about the XSEDE Test Suite). The source code includes a copy of the XSEDE tools and tests (in a folder called “xsede_tools”).

**F.2.1.1. Configuration Variables for Bootstrapping**

The deployment generator uses the same scripting support as the XSEDE Test Suite, although it requires a smaller set of configuration items. This section will describe the critical variables that need to be defined for the bootstrapping process.

The first choice to be made is which namespace the grid should support. In the examples following, we will assume the use of the XSEDE production namespace for bootstrapping the grid. This step copies the example configuration file for the XSEDE namespace into place as the configuration file for the XSEDE tools:

```bash
cd $GENII_INSTALL_DIR/xsede_tools
cp examples/xsede_tools.cfg xsede xsede_tools.cfg
```

Establish these variables in the bash shell environment:

- **GENII_INSTALL_DIR**: point this at the location of the Genesis II source code.
- **GENII_USER_DIR**: set this if you want to store the grid state in a different location than the default. The default state directory is “$HOME/.genesisII-2.0”.
- **JAVA_HOME**: specifies the top-level of the Java JDK or JRE. This is required during deployment for running the keytool file, which is not always on the application path.
- **NEW_DEPLOYMENT**: set this to the intended name of the root container’s deployment. This name should be chosen carefully as a unique and descriptive name for bootstrapping the root container. For example, it could be called “xsede_root” for the root container of the XSEDE grid. It should not be called “default”, “current_grid” or “gffs_eu” which are already in use within the installer or elsewhere.

Important: For users on NFS (Network File System), it is critical that container state directories (pointed at by the GENII_USER_DIR variable) are not stored in an NFS mounted folder. Corruption of the container state can result if this caution is disregarded. Instead, the GENII_USER_DIR should be pointed at a folder that is on local storage to avoid the risk of corruption.

Modify the new xsede_tools.cfg for the following variables:

- **DEPLOYMENT_NAME**: ensure that the chosen NEW_DEPLOYMENT from above is also stored in the configuration file for this variable.

The other variables defined in the xsede_tools.cfg can be left at their existing values (or can remain commented out) when generating the new grid deployment.

The remainder of the chapter will refer to GENII_INSTALL_DIR, GENII_USER_DIR and NEW_DEPLOYMENT as variables defined in the bash shell environment. It is very convenient to load the required environment variables using a script rather than typing them again. Often it
makes the best sense to add the variables to the user’s shell startup script, such as $HOME/.bashrc. Here are some example script commands that set the required variables:

```bash
export GENII_INSTALL_DIR=$HOME/genesis2-trunk
source $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh \ $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh
export NEW_DEPLOYMENT=xsede_root
export GENII_USER_DIR=$HOME/root-state-dir
```

### F.2.2. Creating the GFFS Root Deployment

The six main steps to create the root container for GFFS are: (1) setup the trust store for the deployment, (2) generate key-pairs for the various identities needed for a grid container, (3) start up the GFFS root container, (4) create the root of the RNS name space, (5) archive the deployment, and (6) package the deployment for others. These steps are documented in the following sections.

#### Prerequisites for Generating a Deployment

- These procedures assume that the Genesis II code has been acquired and is already compiled. To build the Genesis II code, refer to Section H.3 on “Building Genesis II from Source on the Command Line” to compile the codebase. It is very important that the unlimited JCE jars are installed on any machine running the GFFS; refer to section H.1 for more information.
- The procedures also require the XSEDE Tests and Tools Suite for execution. The previous section describes configuring the test suite.
- In the following steps, it is crucial that no user state directory exist before the GFFS container creates it. If you have $HOME/genesisII-2.0, then delete it beforehand. (Or if $GENII_USER_DIR points at a different state directory, be sure to delete that.)
- The user state directory must not be stored on an NFS file system. One should point the GENII_USER_DIR at a directory on a local file system.

#### F.2.2.1. Setup the Trust Store

The basic GFFS security configuration for the root container is established in the deployment generator. This involves setting up a resource signing keypair, a TLS keypair, an administrative keypair and the container’s trust store.

The first configuration feature is the “override_keys” folder, which allows the deployment to be built with a pre-existing “admin.pfx” and/or “tls-cert.pfx” file. These files should be in PKCS#12 format with passwords protecting them. If “admin.pfx” is present in “override_keys”, then it will be used instead of auto-generating an administrative keypair. If “tls-cert.pfx” is present, then it will be used for the container’s TLS keypair rather than being auto-generated. The passwords on these PFX files should be incorporated into the “passwords.txt” file discussed in a later section.

The next trust store component is the “trusted-certificates” directory in the deployment_generator. This should be populated with the most basic CA certificates that need to be present in the container’s trust store. The CA certificate files can be in DER or PEM format. Any grid resource
whose certificate is signed by a certificate found in this trust store will be accepted as valid
resources within the GFFS. Also, the GFFS client will allow a connection to any TLS certificate that
is signed by a certificate in this trust store. For example:

```
cd $XSEDE_TEST_ROOT/tools/deployment_generator
cp known-grids/uva_xcg_certificate_authority.cer trusted-certificates
```

The third component of the GFFS trust store is the “grid-certificates” directory, where the bulk of
well-known TLS CA certificates are stored for the grid. This directory will be bound into the
installation program for the GFFS grid, but at a later time, the automated certificate update process
may replace the installed version of those certificates for appropriate clients and containers. The
“grid-certificates” directory can be populated from the official XSEDE certificates folder when
building an XSEDE grid as shown:

```
cd $XSEDE_TEST_ROOT/tools/deployment_generator
cp /etc/grid-security/certificates/* grid-certificates
```

The deployment generator will use the given configuration to create the complete trust store. This
includes generating a resource signing certificate ("signing-cert.pfx") for the grid which is built into
the trust store file ("trusted.pfx"). If not provided, the deployment generator will also automatically
create the root container's TLS certificate ("tls-cert.pfx") and administrative certificate
("admin.pfx"). The trusted-certificates and grid-certificates folders are included verbatim rather
than being bound into trusted.pfx, which permits simpler certificate management later if changes
are needed.

F.2.2.1.1. XSEDE GFFS Root

Building an XSEDE compatible GFFS root requires additional steps. Because the XSEDE grid uses
MyProxy authentication (as well as Kerberos), the deployment generator needs some additional
configuration to support it.

MyProxy Configuration
To authenticate MyProxy logins, an appropriate “myproxy.properties” file must reside in the folder
“deployment-template/configuration” in the deployment generator. Below is the default
myproxy.properties file that is compatible with XSEDE’s myproxy servers; it is already included in
the configuration folder:

```
edu.virginia.vcgr.genii.client.myproxy.port=7514
edu.virginia.vcgr.genii.client.myproxy.host=myproxy.teragrid.org
edu.virginia.vcgr.genii.client.myproxy.lifetime=950400
```

A directory called “myproxy-certs” should also exist under the deployment generator. This
directory should contain all the certificates required for myproxy authentication. The provided
configuration template includes a myproxy-certs directory configured to use the official XSEDE
MyProxy server; this should be replaced with the appropriate CA certificates if the grid is not
intended for use with XSEDE MyProxy.

F.2.2.2. Generate Key-Pairs and Create Deployment
The deployment can be created automatically using the script “populate-deployment.sh” in the deployment_generator folder. **Do not do this step** unless it is okay to completely eliminate any existing deployment named $NEW_DEPLOYMENT (which will be located under $GENII_INSTALL_DIR/deployments).

```bash
cd $XSEDE_TEST_ROOT/tools/deployment_generator
cp passwords.example passwords.txt
cp certificate-config.example certificate-config.txt
```

Edit the passwords specified in “passwords.txt”. These passwords will be used for newly generated key-pairs. These passwords should be guarded carefully.

Edit the certificate configuration in “certificate-config.txt” to match the internal certificate authority you wish to create for the grid. The root certificate created with this configuration will be used to generate all container “signing” certificates, which are used to create resource identifiers inside of containers. Container TLS certificates can also be generated from that root certificate, or they can be provided manually (and their CA certificate should be added to the trust store as described above). Consult the sections “Container Network Security” and “Container Resource Identity” for a discussion of TLS and signing certificates.

The next step generates the necessary certificate files and copies them into the deployment. Again, this step will *destroy* any existing deployment stored in the $GENII_INSTALL_DIR/deployments/$NEW_DEPLOYMENT folder.

```bash
bash populate-deployment.sh $NEW_DEPLOYMENT \ {containerPortNumber} {containerHostName}
```

The container port number is at the installer’s discretion, but it must be reachable through any firewalls if grid clients are to connect to it. The hostname for the GFFS root must also be provided, and this should be a fully qualified DNS hostname. The hostname must already exist in DNS records before the installation.

**F.2.2.3.Starting Up the GFFS Root Container**

These steps can be used to get the GFFS root container service running. They actually will work for any container built from source:

```bash
cd $GENII_INSTALL_DIR
bash runContainer.sh &>/dev/null &
```

Note that it can take from 30 seconds to a couple minutes before the container is finished starting up and is actually online, depending on the host. A container is ready to use once the log file mentions the phrase “Done restarting all BES Managers”. The log file is located by default in $HOME/.GenesisII/container.log.

**F.2.2.4.Create the Root of the RNS Name Space**
The previous steps have set up the deployment and started a container running. That container will now be configured as the root container of the GFFS:

```
cd $GENII_INSTALL_DIR
./grid script local:deployments/$NEW_DEPLOYMENT/configuration/bootstrap.xml
```

Once the bootstrap process has succeeded, it's important to clean up the bootstrap script, since it contains the admin password:

```
rm deployments/$NEW_DEPLOYMENT/configuration/bootstrap.xml
```

At this point, a very basic grid has been established. The core directories (such as /home/xsede.org and /resources/xsede.org) have been created. Any standard groups are created as per the definition of the namespace; for example, the XSEDE bootstrap creates groups called gffs-users for normal users and gffs-admins for administrators. However, there are no users defined yet (besides the administrative keystore login).

The steps above also generate a crucial file called “$GENII_INSTALL_DIR/context.xml”. The context.xml file needs to be made available to grid users before they can connect clients and containers to the new root container. For example, this file could be uploaded to a web-server, or it could be manually given to users, or it could be included in a new Genesis II installation package.

### F.2.2.5. Archive the Root Container Deployment

Now that a deployment has been created, it is important to make an archive of all the generated keys and trusted certificates:

```
# Stop the container.
bash $XSEDE_TEST_ROOT/library/zap_genesis_javas.sh

# Back up the grid key-pairs and certificates.
cd $XSEDE_TEST_ROOT/tools/deployment_generator
bash archive-full-deployment.sh

# Back up the runtime directory for the container.
tar -czf $HOME/grid_runtime_backup.tar.gz $GENII_USER_DIR

# Restart the container.
bash $GENII_INSTALL_DIR/runContainer.sh &>/dev/null &
```

The contents of the root name space for the grid should be backed up regularly; consult the document section How to Backup a Genesis II Grid Container for more details.

These steps will create archives in the $HOME folder. These archives should be stored carefully and not shared with anyone but the grid's administrator group.

### F.2.2.6. Package the Deployment Generator for Others

The grid administrator can make a package from the root container's deployment generator directory that other grid clients and containers can use to connect to the grid. The package will provide the same trust store that the root container uses, and it provides new containers with a TLS certificate that will be trusted by grid clients:
This will create a file called deployment_pack_{datestamp}.tar.gz in the user's home folder. This archive can be shared with other users who want to set up a container or a grid client using the source code. The package includes the container's context.xml file, the trust store (trusted.pfx and other directories), and the admin certificate for the grid.

It is preferred to use a Genesis II installer for all other container and client installations besides the root (bootstrap) container. The above deployment package should be provided to the person building the installer. The package building script uses the deployment package to build an installer that can talk to the new root container.

### F.2.3. Changing a Container's Administrative or Owner Certificate

There is an administrative certificate provided by the installation package for grid containers. Changing the admin certificate has wide-ranging effects: it controls who can remotely administer the grid container and it changes whether operations can be performed on the container by the grid's administrator (such as accounting data collection). Changing the admin certificate for a grid container should not be undertaken lightly.

A related concept to the administrative certificate for a container is the “owner” certificate of the container. The owner is a per-container certificate, unlike the admin certificate that is usually distributed by the installation program. The owner certificate can also be changed from the choice that was made at installation time.

Clients whose credentials contain either the admin or owner certificate are essentially always given permission to perform any operation on any of that grid container's services or on grid resources owned by the container.

For the discussion below, we will refer to the container's security folder as $SECURITY_FOLDER. It will be explained subsequently how to determine where this folder is located.

The grid container admin cert is located in $SECURITY_FOLDER/admin.cer. The .cer file ending here corresponds to a DER-format or PEM-format certificate file. Replacing the admin.cer file changes the administrative keystore for the container.

The container owner certificate is instead located in $SECURITY_FOLDER/owner.cer, and can also be in DER or PEM format.

The owner and admin certificates are also commonly stored in the $SECURITY_FOLDER/default-owners directory. The default-owners directory is used to set default access control for a grid resource during its creation when no user security credentials are present. This is a rather arcane piece of the Genesis II grid container and is mostly used by the grid container during certain container bootstrapping operations.
However, if either certificate is to be changed, then it makes sense to change default-owners too. Otherwise some resources created during container bootstrapping will be “owned” (accessible) by the original certificates. Because of this, if you wish to change the admin or owner certificate for a grid container, it is best to prevent the grid container from starting during installation and to immediately change the admin.cer and/or owner.cer files before starting the grid container for the first time.

If the container has inadvertently been started already but still has no important “contents”, then the default-owners can be changed after the fact. The container should be stopped (e.g. GFFSContainer stop) and the GENII_USER_DIR (by default stored in $HOME/.genesisII-2.0) should be erased to throw out any resources that had the prior administrator certificate associated with them. Again, only do this if there is nothing important installed on this container already! Once the admin.cer and/or owner.cer file is updated, restart the container again (e.g. GFFSContainer start).

If the container has been inadvertently started but does have important contents, then the ACLs of affected resources and services can be edited to remove the older certificate. The easiest method to edit ACLs is to use the client-ui (documented in Section E.6) to navigate to the affected resources and drag the old credential into the trash bin for ACLs it is present in.

Occasionally, a user certificate that owns a container may become invalid or the administrative certificate may need to be swapped out. To swap the certificates into the proper location, we need to resolve the SECURITY_FOLDER variable to a real location for the container. This has become more difficult than in the era when there were only interactive Genesis II installations, because containers can be configured differently when using a host-wide installation of the software. To assist the grid maintainers, a new tool called “tell-config” has been added that can report the security folder location:

```
grid tell-config security-dir
```

Given that one has located the proper SECURITY_FOLDER (and has set a shell variable of that name), these steps take a new certificate file ($HOME/hostcert.cer) and make that certificate both the administrator and owner of a container:

```
# replace administrative certificate:
cp $HOME/hostcert.cer $SECURITY_FOLDER/admin.cer
cp $HOME/hostcert.cer $SECURITY_FOLDER/default-owners/admin.cer
# replace owner certificate:
cp $HOME/hostcert.cer $SECURITY_FOLDER/owner.cer
cp $HOME/hostcert.cer $SECURITY_FOLDER/default-owners/owner.cer
```

If only the admin or only the owner certificate needs to be updated rather than both, then just perform the appropriate section of commands from above.

**F.2.4. XSEDE Trust Store Customization**

When a GFFS container or client is deployed on a host that supports the official XSEDE CA certificates, it is desirable to use the official certificates directory rather than the static copy of the
certificates provided by the install package. This affects two configuration items: the myproxy certificates and the grid’s TLS certificates.

To use the official certificates location for MyProxy, update the “security.properties” file in the container’s deployment configuration folder. The full path to the file should be $GENII_INSTALL_DIR/deployments/current_grid/configuration/security.properties for most installations. Editing this file requires root or sudo permissions if the installation is system-wide (e.g. installed from the RPM). For the root container or other containers with specialized deployments, the path will be based on the active deployment name, such as $GENII_INSTALL_DIR/deployments/xsede_root/configuration/security.properties. The active deployment folder can be shown by running “grid tell-config active-deployment-dir”.

A helper script called “use_official_trust_store.sh” has been developed and is available in “$XSEDE_TEST_ROOT/tools/xsede_admin”. This script performs the necessary edits on the security.properties file given that the GENII_INSTALL_DIR variable is set. Run it without any flags to cause it to point the security.properties at the official certificate locations:

```
bash $XSEDE_TEST_ROOT/tools/xsede_admin/use_official_trust_store.sh
```

After this script is run, the remainder of the edits below are not needed, but the container should be restarted so that it will start using the modified trust store (see section G.2.2 regarding restarting a container). There is a periodic trust store refresh also for containers and clients, but restarting the application will use the new trust store immediately. This command restarts the local container:

```
$GENII_INSTALL_DIR/GFFSCinstaller restart
```

These re-configuration steps are required again after an RPM install is upgraded, since the deployment’s security.properties file will be replaced with the default version.

If for some reason using the script is not appropriate (or if one has an older installation without the script), the trust store modification can also be performed manually with the following steps. Modify security.properties to change this entry from the default:

```
edu.virginia.vcgr.genii.client.security.ssl.myproxy-certificates.location=myproxy-certs
```

Replace “myproxy-certs” above with the official XSEDE certificate directory location:

```
edu.virginia.vcgr.genii.client.security.ssl.myproxy-certificates.location=/etc/grid-security/certificates
```

Similarly, the TLS trust store can be configured to use the official XSEDE CA certificates. The grid-certificates folder is defined in the same security.properties file, where the original entry looks like this:

```
edu.virginia.vcgr.genii.client.security.ssl.grid-certificates.location=grid-certificates
```

The new version should reference the official location of the XSEDE certificates:

```
edu.virginia.vcgr.genii.client.security.ssl.grid-certificates.location=/etc/grid-
```
Afterwards, the container/client install will rely on the official XSEDE certificates for MyProxy and TLS authentication.

F.2.5. Detailed Deployment Information

To create a new deployment from scratch, a directory should be created under “$GENII_INSTALL_DIR/deployments” with the name chosen for the new deployment. That directory should be populated with the same files that the populate-deployment.sh script puts into place. The deployment should inherit from the default deployment.

There are a few important requirements on certificates used with Genesis II:

- Signing certificates (set in security.properties in ‘resource-identity’ variables) must have the CA bit set. Container TLS certificates do not need the CA bit enabled.
- Clients can only talk to containers whose TLS identity is in their trust stores (i.e., the CA certificate that created the TLS certificate is listed).
- When acting as a client, a container also will only talk to other containers whose TLS certificates are in its trust store.

The deployment directory consists of configuration files that specify the container’s properties and trusted identities. For an interactive install using the Split Configuration model (see Section D.8.6), these files are rooted in the deployment directory provided by the installer: $GENII_INSTALL_DIR/deployments/[DEPLOYMENT_NAME]. The following files and directories can usually be found in that folder (although changing properties files can change the name expected for particular files):

- `configuration/security.properties`: Specifies most keystrength and keystore parameters
  - Determines lifetime of resource certificates.
  - Note that variables in the form of `${installer:X}` should have been replaced by the deployment_generator tool.
- `configuration/server-config.xml`: Configuration of the container as an internet server.
  - Provides the hostname on which the container will listen, and which other containers will use in links (EPRs) to that container.
- `configuration/web-container.properties`: Defines the network location of the container.
  - This configures whether to trust self-signed certificates at the TLS layer; it is recommended to leave this as “true” for test environments.
  - If the deployment_generator tool is not used, then the container’s “listen-port” value must be replaced with the appropriate port number.
- `security/admin.cer`: Super-user certificate for the container.
  - This certificate is normally generated by deployment_generator tool.
  - The admin.pfx that corresponds to this certificate is capable of bypassing access control on the container and should be guarded extremely carefully.
Some grids may configure all containers to be controlled by the same administrative certificate.

- **security/owner.cer**: The certificate representing the container's owner.
  - This is the certificate for a grid user who has complete control over the container.
  - Ownership can be changed by swapping out this certificate and restarting the container.

- **security/default-owners**: A directory holding other administrators of this container.
  - Can contain DER or PEM encoded .cer certificates.
  - Any .cer file in this directory is given default permissions on creating services for this container.

- **security/signing-cert.pfx**: Holds the container’s signing key that is used to create resource identifiers.

- **security/tls-cert.pfx**: Holds the TLS certificate that the container will use for all encrypted connections.

- **security/trusted.pfx**: Contains certificates that are members of the container’s trust store.
  - This file encapsulates a set of certificates in PKCS#12 format.

- **security/trusted-certificates**: A directory for extending the container’s trust store.
  - Certificate files can be dropped here and will automatically be part of the container trust store after a restart.
  - This is an easier to use alternative to the trusted.pfx file.

- **security/grid-certificates**: Similar to trusted-certificates, this is a directory that extends the container’s trust store.
  - These certificates are part of the automatic certificate update process.
  - In the XSEDE grid, this directory often corresponds to the /etc/grid-security/certificates folder.

- **security/myproxy-certs**: Storage for myproxy certificates.
  - This directory is the default place myproxyLogin and xsedeLogin use as the trust directory for myproxy integration.

- **configuration/myproxy.properties**: Configuration of the myproxy server.
  - This file is necessary for the myproxyLogin and xsedeLogin commands.

Overrides for the above locations in the Unified Configuration model (for more details see Section D.8.6):

- **$GENII_USER_DIR/installation.properties**: Property file that overrides some configuration attributes in the Unified Configuration model. These include certain elements from security.properties, web-container.properties and server-config.xml.

- **$GENII_USER_DIR/certs**: Local storage of container certificates in Unified Configuration.
  - The certificate (CER) and PFX files for a container with the Unified Configuration are stored here (unless the container uses a specialized deployment folder, see below).
  - The “grid-certificates” folder can be located here, and overrides the “security” folder of the deployment.
  - A “local-certificates” folder can be stored here to contain additional elements of the container’s trust store.
• $GENII_USER_DIR/deployments: Storage for specialized container deployment in the Unified Configuration model.
  o This folder is absent unless the user chose to convert a container with a specialized deployment.
  o This case is similar to the Split Configuration described above, except that it resides under the state directory ($GENII_USER_DIR) rather than the install directory ($GENII_INSTALL_DIR).

F.2.6. Certificate Revocation Management (CRL files)

The Genesis II clients and containers will process Certificate Revocation Lists (CRLs) according to the official certificates directory provided by XSEDE. Genesis II will use CRL files if they are found in the “grid-certificates” trust store folder (see section F.2.2.1). This folder can be pointed at an absolute path, such as the official XSEDE certificates directory (see section F.2.4). The CRL files must end in the characters ".r0" to be recognized as CRL files, and they are expected to be in PEM format as encoded by the fetch-crl tool ([http://linux.die.net/man/8/fetch-crl](http://linux.die.net/man/8/fetch-crl)).

The CRL files found in the configured grid-certificates folder will be loaded and used to block connections to containers that are found to be running one of the revoked certificates. This applies to both a Genesis II client connecting to a container, and also to a container connecting to another container for services.

F.2.6.1. Certificate Package Uploader

On official XSEDE hosts, the grid-certificates configuration should be pointed at the official location, which relies on regular updating of the CRL lists using the fetch-crl tool. On non-XSEDE hosts, the grid-certificates will initially be provided by the install package, but can be caused to update automatically using a copy of the certificates package from within the GFFS grid. The certificates package can be built using the “upload_grid_certs.sh” script provided by the install package. This script creates a copy of the certificates and CRL files found in the official location (/etc/grid-security/certificates) and uploads that package to the GFFS grid (in grid:/etc/grid-security/certificates/grid-certificates-X.tar.gz, where X is replaced by a timestamp). Example usage of the script:

```
bash $XSEDE_TEST_ROOT/tools/xsede_admin/upload_grid_certs.sh
```

The script requires that the logged-in grid user has permission to write the new certificate file as well as permission to create the /etc/grid-security/certificates folder if it does not already exist. The simplest way to obtain the proper rights is to log in as a member of the gffs-admins group, or to request that a grid administrator enable the permission for the particular grid user that will run the upload process.

The upload script can be added to a cron job in order to regularly update the certificate package in the grid. Here is an example cron file that runs the upload script every day at 3am:

```
GENII_INSTALL_DIR=/opt/genesis2-xsede
```

---

Genesis II Omnibus Reference Manual Page 103
This cron job logs in as a grid user with appropriate permissions for running the upload script and then runs the upload script. Afterwards, a new copy of the certificates package should be stored in the grid, and grid clients will periodically update their own copy of the grid-certificates as described in the next section. If any errors occur during the upload process, messages will be printed to the console and the script's exit code will be non-zero.

**F.2.6.2. Automated Certificate Download and Update**

The Genesis II client will periodically check for the presence of a new certificate package file in the grid, and if it is found, the client downloads that file locally and updates the state directory's copy of the grid-certificates folder (in $GENII_USER_DIR/grid-certificates). This folder automatically overrides the shipped version of the grid-certificates in order to use the latest CRL lists.

When the grid-certificates configuration is pointed at an absolute path, the certificate update process will not be performed by the client. This allows the containers and clients to use the official certificates in a local filesystem, as described in section F.2.4. If the grid-certificates configuration is left as a relative path (by default it is just set to “grid-certificates”), then the automatic certificate update process is enabled.

The file “$GENII_USER_DIR/update-grid-certs.properties” tracks the last runtime of the update process and the last package file that was used. To force an update of the grid-certificates for the client, remove that file and run a new instance of the Genesis II client. The state directory's copy of the grid-certificates in “$GENII_USER_DIR/grid-certificates” will have a recent timestamp after the certificates have been updated successfully. One can also examine the client log (in $HOME/.GenesisII/grid-client.log) to see information from the update process.

It is important for the grid-certificates to also be kept up to date on Genesis II containers, if they are not running on official XSEDE hosts. Due to the implementation differences between Genesis II clients and containers, the automated certificate update processing used in the client code cannot be re-used in the container. However, if the grid client updates the local certificates folder, then a container running as the same Unix user can take advantage of this; that is because the state directory is shared between the client and container running on the same account, and the container handles any CRL files found just as the client does. The following cron job uses the grid client to regularly update the grid-certificates for the container:

```bash
GENII_INSTALL_DIR=/opt/genesis2-xede
# m h dom mon dow command
0 5 * * * grid ls /
```

This job just runs the client every day at 5am to list the root directory of GFFS. The side effect is that the client will also test whether there is a new certificates package and update its local copy of the grid-certificates if a new package is found. Afterwards, the container will start using the new grid-certificates, which include the latest CRL files. The container will start using the files after its
process is restarted, but running containers will also periodically reload the trust store (every 4 hours by default).

F.3. Grid Containers

The notion of a container is a simple idea at its root. A container encapsulates the services that a particular Genesis II installation can provide on the web. In other service models, the container might have been referred to as being a "server". However, in the field of web services, any program that can host web services may be called a “container”. This includes application servers like "tomcat", “glassfish” and other programs that can host web services.

In the case of Genesis II, commodity container features are provided by the Jetty Application Server and Apache Axis 1.4, which route requests from clients to the specific services in Genesis II. But in general terms, we refer to any separate Genesis II install that can provide web services to the grid as a “container”. If a Genesis II installation just uses containers and provides no services of its own, then it is referred to as a "client".

F.3.1. Container Structure

Grid containers based on Genesis II have a representation in the GFFS as “resource forks”. The resource fork provides a handle for manipulating the actual internals of the container, but the resource fork resembles a normal file or directory. The notion of resource forks provides an easy to use mapping that represents the capabilities of the container (and other resource types) within the GFFS filesystem.

The top-level resource fork is VcgrContainerPortType, which provides access to the container itself. Once a container is linked into the grid via the VcgrContainerPortType, the other services can be viewed under its Services folder. This command shows the basic step for linking a container into the grid. The target location where the container resides must be writable by the user creating this link:

```
grid ln \
  --service-url=https://{hostname}:{port#}/axis/services/VCGRContainerPortType \ 
  {/target/path/in/grid}
```

# example for a personal container in the user’s home folder.
`grid ln \ 
  --service-url=https://cs.vogon.edu:18443,axis/services/VCGRContainerPortType \ 
  /home/xsede.org/fred/MyContainer`

# show the services available on the container:
`grid ls /home/xsede.org/fred/MyContainer/Services`

The 'ln' command links the container into the grid, at which point the owner or administrator of the container can make the container's services available to other users.

As an example of a container service, the X509AuthnPortType service on a container is where basic X509 grid identities are created. A user on the XSEDE grid (with appropriate permissions) can list the directory contents for that port-type in the resource fork for the primary STS container by executing:
All of the X509 users and groups that have been created based on that port type are shown. An alternative identity is the Kerberos port type, which can be listed similarly:

```
grid ls /resources/xsede.org/containers/sts-1.xsede.org/Services/X509AuthnPortType
```

However, these user entries are not simple files that can be copied to someplace else in the grid and used like normal files are used. They only make sense in the context in which the container manages them, which is as IDP entities. That is why we create a link to the user identity in the /users/xsede.org folder (see the Creating Grid Users section for more details) rather than just copying the identity; the link maintains the special nature of the identity, whereas a copy of it is meaningless.

There are several resource forks under the container's topmost folder. Each item's name and purpose are documented below. Note that visibility of these items is controlled by the access rights that have been set, and not every user will be able to list these folders.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resources</td>
<td>This is a directory that lists all the resources, categorized by their types, that have been created in this container. The directory is accessible to only the administrator of the container.</td>
</tr>
<tr>
<td>filesystem-summary.txt</td>
<td>This file reports the current free space of the container's filesystem.</td>
</tr>
<tr>
<td>Services</td>
<td>This directory holds all of the port-types defined on Genesis II containers. Each port-type offers a different type of service to the grid.</td>
</tr>
<tr>
<td>container.log</td>
<td>This is a mapping for the file container.log under the GENII_INSTALL_DIR. Note that if that is not where the container log is being written, then this GFFS entry will not be accessible. This is mainly a concern for containers that are built from source, as the graphical installer sets up the configuration properly for the container log.</td>
</tr>
</tbody>
</table>

Within the Services directory, one will see quite a few port types:

<table>
<thead>
<tr>
<th>Port Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationDeployerPortType</td>
<td>Deprecated; formerly part of preparing an application to use in the grid (e.g. unpacking from zip or jar file, etc).</td>
</tr>
<tr>
<td>ApplicationDescriptionPortType</td>
<td>Deprecated; associated with ApplicationDeployerPortType.</td>
</tr>
<tr>
<td>BESActivityPortType</td>
<td>OGSA port-type for monitoring and managing a single activity that has been submitted into a basic execution service.</td>
</tr>
<tr>
<td>CertGeneratorPortType</td>
<td>A certificate authority (CA) that can generate new container TLS certificates for use within a grid. Used in XCG, not expected to be used in XSEDE.</td>
</tr>
<tr>
<td>EnhancedRNSPortType</td>
<td>The Resource Namespace Service (RNS) port-type, which provides directory services to the grid. In addition to the standard RNS operations, this port-type supports a file creation service (createFile) that creates a file resource in the same container that the RNS resource resides in.</td>
</tr>
<tr>
<td>ExportedDirPortType</td>
<td>Port-type for accessing and managing a directory that lies inside an exported root. It is not needed when using the light-weight export mechanism.</td>
</tr>
<tr>
<td>ExportedFilePortType</td>
<td>Port-type for accessing a file inside an exported directory.</td>
</tr>
</tbody>
</table>
Like the ExportedDirPortType, it is not needed for the light-weight export.

<table>
<thead>
<tr>
<th>Port-Type Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExportedRootPortType</td>
<td>Port-type for exporting a directory in local file-system in the GFFS namespace. This port-type is extended by its light-weight version.</td>
</tr>
<tr>
<td>LightWeightExportPortType</td>
<td>We recommend using this for exports instead of ExportedRootPortType.</td>
</tr>
<tr>
<td>FSProxyPortType</td>
<td>Similar to the LightweightExportPortType, but allows any filesystem-like subsystem that has a Genesis II driver to be mounted in the grid. Examples include ftp sites, http sites, and so forth.</td>
</tr>
<tr>
<td>GeniiBESPortType</td>
<td>The BES (Basic Execution Management) Service provides the capability to execute jobs submitted to the container.</td>
</tr>
<tr>
<td>GeniiPublisherRegistrationPortType</td>
<td>An extension of the WS-Notification web service allowing data providers for subscriptions to register with the grid. Interoperates with GeniiResolverPortType, GeniiSubscriptionPortType and GeniiWSNBrokerPortType.</td>
</tr>
<tr>
<td>GeniiPullPointPortType</td>
<td>Unimplemented.</td>
</tr>
<tr>
<td>GeniiResolverPortType</td>
<td>A service that provides WS-Naming features to the grid. Clients can query the location of replicated assets from a resolver.</td>
</tr>
<tr>
<td>GeniiSubscriptionPortType</td>
<td>Port-type for managing a subscription for web-service notifications. All Genesis II resources extend WSN NotificationProducer port-type, and, therefore, can publish notifications. This port-type is used to pause, resume, renew, and destroy any subscription for a Genesis II resource's notifications.</td>
</tr>
<tr>
<td>GeniiWSNBrokerPortType</td>
<td>Deprecated; implements a form of subscription forwarding.</td>
</tr>
<tr>
<td>JNDIAuthnPortType</td>
<td>An authentication provider based on Java Native Directory Interface (JNDI) to allow grid logins via LDAP servers and other types that JNDI provides a wrapper for.</td>
</tr>
<tr>
<td>KerbAuthnPortType</td>
<td>This port-type supports Kerberos Authentication.</td>
</tr>
<tr>
<td>LightWeightExportPortType</td>
<td>Port-type for exposing a directory in local file-system into the GFFS namespace. Users with appropriate credentials can access and manipulate an exported directory just like a typical RNS resource. The port-type also support a quitExport operation that detaches the exported directory from the GFFS namespace.</td>
</tr>
<tr>
<td>PipePortType</td>
<td>A non-standard port-type for creating a unidirectional, streamable ByteIO communications channel. Once the pipe is created, the client can push data into one end and it will be delivered to the other end of the pipe. This is a less storage-intensive way to transfer data around the grid, because there does not need to be any intermediate copy of the data stored on a hard-drive or network location.</td>
</tr>
<tr>
<td>QueuePortType</td>
<td>A meta-scheduler port-type for submitting and managing jobs in multiple basic execution services (BES). The user can submit, query status, reschedule, kill, etc. one or more activities through this port-type as well as can configure how many slots of individual BESes will be used by the meta-scheduler.</td>
</tr>
<tr>
<td>RExportDirPortType</td>
<td>An extension of the ExportedDirPortType that supports replication.</td>
</tr>
<tr>
<td>RExportFilePortType</td>
<td>An extension of the ExportedFilePortType that supports replication.</td>
</tr>
</tbody>
</table>
RExportResolverFactoryPortType  This port-type creates instances of the RExportResolver port-type.

RExportResolverPortType  A port-type whose EPR is embedded into an exported directory or file's EPR to support resolving to a replica on failure.

RandomByteIOPortType  Port-Type for accessing a bulk data source in a session-less, random way. The user can read and write blocks of data starting at any given offset. In other words, the port-type exposes a data-resource as a traditional random-access file in a local file-system.

StreamableByteIOPortType  Port-type for accessing a bulk data source via a state-full session resource. It supports the seekRead and seekWrite operations.

TTYPortType  An earlier implementation of the PipePortType that was used at one time for managing login sessions.

VCGRContainerPortType  The container port-type provides the top-level handle that can be used to link containers into the GFFS. This port-type represents the container as a whole. When this port-type is linked into the grid, users can see the container structure under that link (including, eventually, the VCGRContainerPortType).

WSIteratorPortType  Port-type for iterating over a long list of aggregated data, instead of retrieving it in a single SOAP response to its entirety. This port-type is used in conjunction with other port types such as in entry listing of an RNS resource or job listing in a queue. Its interface has exactly one operation: the iterate operation.

X509AuthnPortType  This port-type is the Identity Provider (IDP) for the container. New identities can be created under this resource fork, and existing identities can be listed or linked from here.

F.3.2.  Where Do My Files Really Live?

When a user looks at the contents of the GFFS, the real locations of the files and directories are hidden by design. The EPR for files (ByteIO) and directories (RNS resources) can be queried to find where they really reside, but usually this is not of interest to users during their daily activities in the grid. It is much more convenient to consider the files as living “in the grid”, inside the unified filesystem of the GFFS.

However, this convenient view is not always sufficient, and a user may need to be very aware of where the files really reside. For example, streaming a large data file from the root container of the GFFS to a BES container that is half-way around the world is simply not efficient. In the next section, we describe how to store files on whatever container is desired. But first, it is important to be able to determine where the files really reside. For example, a user is given a home directory when joining a grid, but where does that home directory really live?

To determine which container is providing the storage location for a directory, use the following command:

```
# Show the EPR for a directory:
grid ls -e -d (/path/to/directory)
```
This will produce a lengthy XML description of the EPR, and included in that will be an xml element called ns2:Address that looks like the following:

```
<ns2:Address xsi:type="ns2:AttributedURIType">
https://server.grid.edu:18230/axis/services/EnhancedRNSPortType?genii-container-id=52451897-8A90-5BE4-1FAD-5D983AD2224C
</ns2:Address>
```

This provides several useful pieces of information. The hostname server.grid.edu in this example is the host where the container really lives. The port number after the colon (18230 in this example) is the port where container provides its web-services. In addition, the unique id of the RNS resource itself (the directory being queried) is shown.

A file's presence in a particular directory in the GFFS (i.e., an RNS path) does not necessarily mean that the file actually resides in the same container as the directory. That is because files linked into the directory from other containers are still actually stored in the original location. To show which container a file or directory is really stored on, use the following command to display the item's EPR:

```
# Show the EPR for a file.
grid ls -e {/path/to/file}
```

This produces another EPR dump, which will again have an ns2:Address entry:

```
<ns2:Address xsi:type="ns2:AttributedURIType">
https://wallaby.grid.edu:19090/axis/services/RandomByteIOPortType?genii-container-id=46D0CE0D-7F85-C5BA-5AC8-695A77E7668A
</ns2:Address>
```

Here again, we can see the host and port of the container storing the file. In this case, the file is stored on the container at wallaby.grid.edu on port 19090.

### F.3.3. Serving GFFS Folders from a Specific Container

By default, the mkdir command will create directories using the GFFS root container as the storage location. It is desirable to store files on other containers to reduce load on the root container. It is also faster to access data files when they are closer geographically. To create a directory on a particular container, use the following steps. Afterwards, any files or directories stored in the new folder will be physically stored on the {containerPath} specified:

```
# Create the directory using the container's RNS port-type.
grid mkdir --rns-service={containerPath}/Services/EnhancedRNSPortType{/path/to/newDir}
```

Note that if links are created within the newDir, their storage locations are dictated by where the real file is stored, and not where the newDir is stored. Only the link itself is stored in newDir.

### F.3.4. Container Network Security

All connections between GFFS containers use TLS (Transport Layer Security) to encrypt the SOAP communications and avoid exposing critical data on the network. The basis of the TLS connection
is a certificate file called the “TLS certificate”, which is configured in a container’s deployment in the “security.properties” file.

The TLS certificate represents the container’s identity on the network. Incoming connections to a container will see that certificate as “who” they are connecting to. When the container makes outgoing connections, it will use this certificate as its outgoing identity also.

In the case of some grids, the TLS certificates can be created automatically for a container at installation time using the grid’s Certificate Generator. This is handled automatically by the Genesis II GFFS installer.

Other grids may have stricter security requirements, such that they provide their own TLS certificates from a trusted CA. The installer can support such a container when the user is already in possession of an approved TLS certificate; there is an install dialog for adding the certificate. If the certificate is not yet available, the user can go ahead and generate a temporary TLS certificate, and replace that later with the official certificate when available.

The TLS certificate for a container can be replaced at any time. After switching to a different TLS certificate and updating the configuration for the container, one must restart the container to cause the new TLS certificate to take effect.

A GFFS grid client will only connect to a container if the TLS certificate of the container is known to the client, by its presence in the client’s trust store. This ensures that the container is intentionally part of the grid, rather than being from some unknown source. GFFS containers also follow this restriction when they act as clients (to connect to other containers for services).

Grid clients for a given grid will automatically trust the TLS certificates generated by the grid’s Certificate Generator. If specific TLS certificates are used for each container, then each of the CA certificates that created the TLS certificates must be added to the installation’s trust store. Once those CA certificates are present, grid clients and containers will then allow connections to be made to the affected container. Further information on configuring the TLS certificate is available in Section F.2.5 as well as in the internal documentation in the deployment’s security.properties file.

**F.3.5. Container Resource Identity**

Besides the TLS certificate described in the last section, there is another type of certificate used by containers. This certificate is called the “Signing Certificate” and it is used for generating resource identifiers for the assets owned by a container.

The signing certificate is always created by the grid’s Certificate Generator. It is an internal certificate that will not be visible at the TLS level, and so does not participate in the network connection process. Instead, the Signing certificate is used to achieve a cryptographically secure form of GUID (Globally Unique IDentifier) for each resource in a container. Each resource has a unique identity generated for it by the container using the Signing certificate. This allows a container to know whether a resource was generated by it (e.g., when the resource is owned by the container and “lives” inside of it), or if the resource was generated by a different container.
All of the Signing certificates in a grid are “descended from” the root signing certificate used by the Certificate Generator, so it is also clear whether a resource was generated inside this grid or generated elsewhere.

The resource identifiers created by a container’s Signing certificate are primarily used in SAML (Security Assertion Markup Language) Trust Delegations. Each resource can be uniquely identified by its particular certificate, which allows a clear specification for when a grid user has permitted a grid resource to act on her behalf (such as when the user delegates job execution capability to a Queue resource, which in turn may delegate the capability to a BES resource).

The Signing certificate thus enables containers to create resources which can be described in a standardized manner, as SAML assertions, in order to interoperate with other software, such as Unicore EMS services.

F.3.6. User Quota Configuration

It is possible to restrict the ability of grid users to create any files on a container. It is also possible to permit file creations according to a quota system. Either approach can be done on a per-user basis.

All files stored in a grid container (that is, “random byte IO” files) are located in the $GENII_USER_DIR/rbyteio-data folder. Each grid user’s name is the first directory component under the rbyteio-data folder, allowing individualized treatment of the user’s ability to create files.

F.3.6.1. Blocking user ability to create files on a container:

If a user is to be disallowed from storing any byteio type files on the container, then it is sufficient to change the user’s data file folder permission to disallow writes for the OS account running the container.

For example: The container is running as user “gffs”. The user “jed” is to be disallowed from creating any files in that container. The user’s random byte IO storage folder can be modified like so:

```
chmod 500 $GENII_USER_DIR/rbyteio-data/jed
```

To enable the user to create files on the container again, increase the permission level like so:

```
chmod 700 $GENII_USER_DIR/rbyteio-data/jed
```

F.3.6.2. Establishing quotas on space occupied by user files:

Limits can be set on the space occupied by a user’s random byte IO files, enabling the sysadmin to prohibit users from flooding the entire disk with their data. The following is one approach for establishing a per-directory limit for the user’s data files.

Assuming that a user named “jed” is to be given a hard quota limit of 2 gigabytes, the following steps will restrict jed’s total file usage using a virtual disk approach:
The /var/virtual_disks path above must be accessible by the account running the container.

F.3.7. Genesis Database Management

The Genesis II GFFS containers rely on the Apache Derby Embedded Database implementation for their database support. Much of the time the database engine is trouble free, but occasionally it does need maintenance. This section covers topics related to the management of the GFFS database.

F.3.7.1. Using the Derby Database Tools

The following procedure requires the Java Development Kit (JDK) and the Apache Derby software. Java is available at Oracle at http://www.oracle.com/technetwork/java/javase/downloads/index.html. The derby-db tools can be downloaded from http://db.apache.org/derby/derby_downloads.html

Decompress the derby tools and set the DERBY_INSTALL environment variable to point to path where derby tools reside.

F.3.7.2. Configure Embedded Derby

This link provides the material that is summarized below: http://db.apache.org/derby/papers/DerbyTut/install_software.html#derby_configure

To use Derby in its embedded mode set your CLASSPATH to include the jar files listed below:

derby.jar: contains the Derby engine and the Derby Embedded JDBC driver
derbytools.jar: optional, provides the ij tool that is used by a couple of sections in this tutorial

You can set your CLASSPATH explicitly with the command shown below:

Windows:

C:\> set CLASSPATH=%DERBY_INSTALL%\lib\derby.jar;%DERBY_INSTALL%\lib\derbytools.jar;

UNIX:

$ export
The Derby software provides another way to set CLASSPATH, using shell scripts (UNIX) and batch files (Windows). This tutorial shows how to set CLASSPATH explicitly and also how to use the Derby scripts to set it.

Change directory now into the DERBY_INSTALL/bin directory. The setEmbeddedCP.bat (Windows) and setEmbeddedCP (UNIX) scripts use the DERBY_INSTALL variable to set the CLASSPATH for Derby embedded usage.

You can edit the script itself to set DERBY_INSTALL, or you can let the script get DERBY_INSTALL from your environment. Since you already set DERBY_INSTALL, you don’t need to edit the script, so go ahead and execute it as shown below:

Windows:

```
C:\> cd %DERBY_INSTALL%\bin C:\Apache\db-derby-10.10.1.1-bin\bin>
setEmbeddedCP.bat
```

UNIX:

```
$ cd $DERBY_INSTALL/bin $ . setEmbeddedCP.ksh
```

**F.3.7.3. Verify Derby**

Run the sysinfo command, as shown below, to output Derby system information:

```
java org.apache.derby.tools.sysinfo
```

Successful output will look something like this:

```
------------------ Java Information ------------------
Java Version: 1.7.0_11
Java Vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.7.0_11.jdk/Contents/Home/jre
class path: /Users/me/src:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derby.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derby.war:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_de_DE.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_es.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_fr.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_hu.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_it.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_ja_JP.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_ko_KR.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_pl.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_pt_BR.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_ru.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_zh_CN.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyLocale_zh_TW.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyclient.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbynet.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbyrun.jar:/Users/me/sw/z/10.10.1/db-derby-10.10.1.1-bin/lib/derbytools.jar:/Users/me/sw/db2jcc/lib/db2jcc.jar:/Users/me/sw/db2j
```
The output on your system will probably be somewhat different from the output shown above, but it should reflect the correct location of jar files on your machine and there shouldn't be any errors. If you see an error like the one below, it means your class path is not correctly set:

```
$ java org.apache.derby.tools.sysinfo
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/derby/tools/sysinfo
```

F.3.7.4. Start up ij

Start up ij with this command:

```
java org.apache.derby.tools.ij
```

You should see the output shown below:

```
ij version 10.4 ij>
```

The error below means the class path isn't set correctly:

```
java.org.apache.derby.tools.ij Exception in thread "main"
java.lang.NoClassDefFoundError: org/apache/derby/tools/ij
```

F.3.7.5. Connect to Genesis II database

Start up ij again and connect to the database in Genesis II state directory:

```
java org.apache.derby.tools.ij
```
F.3.7.6. Execute SQL statements

Once you connect to a database, you can execute SQL statements. `ij` expects each statement to be terminated with a semicolon (;); for example:

```
ij> create table derbyDB(num int, addr varchar(40));
```

F.3.7.7. Disconnect from a database

The disconnect command disconnects from the current database:

```
ij> disconnect;
```

F.3.7.8. Exit

The exit command quits out of `ij` and, in embedded mode, shuts down the Derby database:

```
ij> exit;
```

F.3.7.9. Run SQL Scripts to compress Genesis II state directory

Derby does not provide automatic database compaction, and hence the database can grow quite large over months of operation. This section provides the techniques needed to compact the database.

You can execute SQL scripts in `ij` as shown below:

```
ij> run 'compress.db';
```

Here is sample `compress.db` script (script name need not end with `db`, it can be any extension):

```sql
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'ACCOUNTINGRECORDS', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'ACCTCOMMANDLINES', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'ACCTRECCREDMAP', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'ALARMTABLE', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'BESACTIVITYINDEX', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'BESACTIVITYFAULTINDEX', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'BESACTIVITYINDEXPROPERTYINDEX', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'BESPOLICYINDEX', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'CLOUDACTIVITIES', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'CLOUDRESOURCES', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'CONTAINERPROPERTIES', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'CONTAINERSERVICESPROPERTIES', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'CREDENTIALS', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'ENTRIES', 0);
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('SA', 'EXPORTEDDIR', 0);
```
F.4. Grid Queues

The Genesis II system provides a queuing feature for scheduling jobs on a variety of different types of BES services. The queue matches the job requirements (in terms of number of CPUs, required memory, parameters for matching types of service required, and other factors) with a BES that is suited to execute the job. When other jobs are already executing on the necessary resources, the queue keeps the job waiting until the resources become available. Queues also provide services to users for checking on their jobs’ states and managing their jobs while in the queue.

F.4.1. Creating a Genesis II Queue
A queue in the GFFS generally does not do any job processing on its own. It does all of the
processing via the BES resources that have been added to the queue. The following shows how to
create the queue itself; later sections describe how to add resources of different types to the queue:

```bash
# Create the queue resource on a container.
grid create-resource {containerPath}/Services/QueuePortType {/queues/queueName}
```

Note that, by convention, publicly available queues are stored in the /queues folder in GFFS, but
they can be created anywhere in the GFFS that the user has access rights.

```bash
# Give the owner full administrative rights on the new queue.
grid chmod {/queues/queueName} +rwx {/users/ownerName}
# Or, give a group rights to use the queue.
grid chmod {/queues/queueName} +rwx {/groups/groupName}
```

It is generally considered wiser to give a particular group rights to the queue; then members can be
added and removed from the group without causing a lot of repeated maintenance on the queue
itself.

Example of Steps in Action:

```bash
grid create-resource /containers/poe/Services/QueuePortType /queues/poe-queue
grid chmod /queues/poe-queue +rwx /users/drake
grid chmod /queues/poe-queue +rwx /groups/uva-idp-group
```

**F.4.2. Linking a BES as a Queue Resource**

The computational elements in the grid are represented as Basic Execution Services (BES)
containers. These will be discussed more in the next section, but assuming that a BES is already
available, it can be added as a resource on a grid queue with the following steps. Once added as a
resource, the queue can start feeding appropriate jobs to that BES for processing.

Make the BES available on the queue:

```bash
grid ln {/bes-containers/besName} {/queues/queueName}/resources/{besName}
```

The mere presence of the BES as a resource indicates to the queue that it should start using the BES
for job processing.

```bash
# Set the number of queue slots on the resource:
grid qconfigure {/queues/queueName} {besName} {queueSlots}
```

Example of Steps in Action:

```bash
grid ln /bes-containers/poe-bes /queues/poe-queue/resources/poe-bes
grid qconfigure /queues/poe-queue poe-bes 23
```

**F.5. Basic Execution Services (BES)**

To configure Genesis II BES on a Linux machine, the grid administrator should install a Genesis II
container on that machine first. Usually this machine will be a submit node on a cluster, and it
should have a batch job submission system (like Unicore, PBS, SGE, etc) set up on the cluster. Once
the BES is configured, it will talk to the underlying batch job submission system and submits users’ jobs to the nodes on the cluster. Grid admin can also configure attributes specific to that machine while setting up the BES.

**F.5.1. How to Create a Fork/Exec BES**

The native BES type provided by Genesis II is a fork/exec BES. This type of BES simply accepts jobs and runs them, and offers no special functionality or cluster support. It offers a very basic way to build a moderately-sized computation cluster, if needed.

Adding a BES service requires that a Genesis II container already be installed on the host where the BES will be located. To create a BES on that container, use the following steps:

```
# Create the BES on the container.
grid create-resource {containerPath}/Services/GeniiBESPortType {/bes-containers/besName}

# Give the BES owner all rights on the BES.
# (Write permission makes the user an administrator for the BES):
grid chmod {/bes-containers/besName} +rwx {/users/ownerName}

# Give the appropriate queue permissions to use the BES for submitting jobs. #
This includes any queue where the BES will be a resource:
grid chmod {/bes-containers/besName} +rx {/queues/queueName}

# Give the managing group rights to the BES.
grid chmod {/bes-containers/besName} +rx {/groups/managingGroup}
```

Example of Steps in Action:

```
grid create-resource /containers/poe/Services/GeniiBESPortType /bes-containers/poe-bes
grid chmod /bes-containers/poe-bes +rwx /users/drake
grid chmod /bes-containers/poe-bes +rx /queues/poe-queue
grid chmod /bes-containers/poe-bes +rx /groups/uva-idp-group
```

**F.5.2. Running a BES Container With Sudo**

In the previous example, the BES was created on a machine to run jobs submitted by grid users. These jobs execute on the local machine (Fork/Exec) or on the system’s compute nodes through the local queuing system interface (PBS, Torque, etc). From the local machine’s standpoint, all of these jobs come from one user; that is, all of the local processes or job submissions are associated with the same local uid. The (local) user that submits the jobs is the same (local) user that owns the container.

This situation can lead to a security vulnerability: depending on the local configuration of the disk resources, if a (grid) user can submit jobs to the BES to run arbitrary code as the same (local) user as the container, that job will have access to all of the state (including grid credentials, files, etc) stored on the local file system.

To protect the container and other grid resources from the jobs, the BES may be configured to run the jobs as a unique local user account, which has limited permissions within the file system and
execution environment. This account can be configured to have access to only the files and
directories specifically for that job, and thereby protect the container and the local operating
system. This is accomplished using Linux's built-in command "sudo", which changes the effective
user for the process as it runs.

In the following, we will assume that the container was installed and runs as the user "besuser". The
effective user for running jobs will be “jobuser”. Setting up the local system (sudo, users, etc)
requires administrative rights on the local machine, so it is assumed that “besuser” has the ability to
execute administrative commands. All commands that require this permission will start with “sudo”
(e.g: sudo adduser jobuser). Some aspects of configuration are common among any deployment
using this mechanism, while other aspects depend on the type of BES (Fork/Exec or Native Queue)
or the capabilities of the local operating system. These are described below.

F.5.2.1. Common Configuration

To enable execution of jobs as a unique user, first that user must exist on the local operating system.
It is recommended that a new user is created specifically for the task of running jobs, with minimal
permissions on system resources.

Within the local operating system, create “jobuser”:

```
sudo adduser jobuser
```

Set jobuser’s default umask to enable group access, by adding umask 0002” to $HOME/.bashrc or
similar. This will ensure that any files created by a running job can be managed by the container
once the job has terminated.

Grant jobuser access to any shared resources necessary to execute jobs on the current system, such
as job queues, or shared file-systems.

F.5.2.2. Extended ACLs vs. Groups

When a BES container sets up the working directory for a job, the files it creates/stages in are
owned by the besuser. The jobuser must have access permissions to these files to successfully
execute the requested job. There are two mechanisms by which the system may grant these
permissions: groups or extended access control lists (Extended ACLs).

Extended ACLs are the preferred method for extending file permissions to another user, and are
available in most modern Linux deployments. They provide a files owner the ability to grant read,
write, and execute permissions on a per-user basis for each file. Compare this to Linux Groups,
where every user in the group receives the same permissions.

In the following commands, we assume the Job state directory for the BES will be the default
location at $GENII_USER_DIR/bes-activities. If it is configured to be located somewhere else in the
file system, adjust the commands below accordingly.

If the BES is to be configured using Extended ACLs:

```
# Set the default access on the Job state directory and its children,
```
# so permission propagates to new job directories:
```bash
```

# Grant jobuser access to the Job state directory and its existing children:
```bash
sudo setfacl -mR u:besuser:rwx,u:jobuser:rwx $GENII_USER_DIR/bes-activities
```

# If the BES is to be configured using Groups:

# Create a new group, for this example "g2".
```bash
sudo addgroup g2
```

# Add jobuser and besuser to group "g2":
```bash
sudo usermod -aG g2 besuser
sudo usermod -aG g2 jobuser
```

# Depending on the local operating system's configuration, it may be necessary to set group "g2" as the default group for besuser and jobuser:
```bash
sudo usermod -g g2 besuser
sudo usermod -g g2 jobuser
```

# Change the owning group of the Job state directory and its children to group "g2":
```bash
sudo chgrp -R g2 $GENII_USER_DIR/bes-activities
```

# Set the "sticky" bit on the Job state directory,
# so permission propagates to new job directories:
```bash
sudo chmod g+s $GENII_USER_DIR/bes-activities
```

# Grant explicit permission to the job process wrapper executable:
```bash
sudo chmod -f g+rx $GENII_USER_DIR/bes-activities/pwrapper*
```

### F.5.2.3. Fork/Exec vs. Native Queue

Once the Job state directory has been configured, either with groups or Extended ACLs, the ability to execute jobs as the jobuser must be granted to the besuser. This is accomplished using Linux’s built-in “sudo” command. To enable a user to use “sudo”, an administrator must add an entry into the “sudoers” file. This entry should limit the set of commands that the user may execute using “sudo”, or no actual security is gained by creating another user. It is recommended that “sudo” only be granted for the specific commands required to launch the job.

#### F.5.2.3.1. Fork/Exec BES

If the BES is to be a Fork/Exec flavor BES, the ability to run with “sudo” should be granted only to the job process wrapper executable. This executable is included with the Genesis II deployment and is located in the Job state directory. The executable used depends on the local operating system, but the filename will always begin with "pwrapper". To grant “sudo” ability for this executable, add an entry like the following to the file “/etc/sudoers”:

```bash
besuser ALL=(jobuser) NOPASSWD: {jobStateDir}/{pwrapperFile}
```

Where `{jobStateDir}` is the full path to the Job state directory, and `{pwrapperFile}` is the filename of the process wrapper executable. Note that “sudo” does not dereference environment variables, so the full path must be specified in the entry. For example, if the Job state directory is located at “/home/besuser/genesisII-2.0/bes-activities” and the operating system is 32-bit Linux, the process wrapper executable will be “pwrapper-linux-32”, the “sudoers” entry should be:

```bash
besuser ALL=(jobuser) NOPASSWD: /home/besuser/.genesisII-2.0/bes-activities/pwrapper-linux-32
```
Once “sudo” has been granted to besuser, it may be necessary to restart the operating system before the changes take effect.

Once the local configuration is complete, a BES should be created which utilizes the “sudo” capability. This is accomplished by specifying a “sudo-pwrapper” cmdline-manipulator type in the construction properties for the new BES. An example construction properties file is included below, which we will call sudo-pwrapper.xml.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ns2:fuse-directory xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"/>
  <ns3:environment-export xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"/>
  <ns2:cmdline-manipulators>
    <ns4:manipulator-variation name="sudo_pwrapper" type="sudo-pwrapper">
      <ns4:sudo-pwrapper-configuration>
        <ns4:target-user> {jobuser} </ns4:target-user>
        <ns4:sudo-bin-path> {sudo} </ns4:sudo-bin-path>
      </ns4:sudo-pwrapper-configuration>
    </ns4:manipulator-variation>
    <ns4:call-chain>
      <ns4:manipulator-name>sudo_pwrapper</ns4:manipulator-name>
    </ns4:call-chain>
  </ns2:cmdline-manipulators>
  <ns2:post-execution-delay xsi:nil="true"/>
  <ns2:pre-execution-delay xsi:nil="true"/>
  <ns2:resource-overrides/>
</construction-parameters>
```

Note that there are two parameters in the above example that require system-specific values: the “target-user” element which is shown with value {jobuser} and the “sudo-bin-path” element which is shown with value {sudo}. {jobuser} should be the user name of the account under which the jobs will execute (the “jobuser” in all of the examples provided), and {sudo} should be the absolute path to the sudo executable (e.g. “/bin/sudo”).

To create the BES with these properties, execute the following:

```
grid create-resource --construction-properties=local:./sudo-pwrapper.xml
{containerPath}/Services/GeniiBESPortType {/bes-containers/newBES}
```

F.5.2.3.2. Native-Queue BES
If the BES is to be a Native Queue flavor BES, the ability to run with "sudo" should be granted only to the queue executables, e.g. "qsub", "qstat", and "qdel" on PBS-based systems. To grant "sudo" ability for these executables, add an entry like the following to the file "/etc/sudoers:

```bash
besuser ALL=(jobuser) NOPASSWD: /{bin-path}/qsub, /{bin-path}/qstat, /{bin-path}/qdel
```

Where `{bin-path}` is the full path to the directory where the queuing system executables are located, and `{qsub}`, `{qstat}`, and `{qdel}` are the filenames of the queuing system executables for submitting a job, checking a job’s status, and removing a job from the queue, respectively. Note that "sudo" does not dereference environment variables, so the full path must be specified in the entry. For example, if the queue executables are installed in the directory "/bin", and the native queue is PBS, the "sudoers" entry should be:

```bash
besuser ALL=(jobuser) NOPASSWD: /bin/qsub, /bin/qstat, /bin/qdel
```

Once "sudo" has been granted to besuser, it may be necessary to restart the operating system before the changes take effect.

Once the local configuration is complete, a BES should be created which utilizes the “sudo” capability. This is accomplished in the construction properties for the BES by prefacing the paths for the queue executables with the sudo command and parameters to indicate the jobuser. An example snippet from a construction properties file is shown below. Substituting these elements for the corresponding "pbs-configuration" element in the cons-prop.xml shown above will result in a construction properties for a sudo-enabled variant of the native-queue BES, which we will call sudo-native-queue.xml.

```xml
<ns5:pbs-configuration xmlns="" xmlns:ns7="http://vcgr.cs.virginia.edu/native-queue" queue-name="My-Queue">
  <ns7:qsub path="(sudo)">
    <ns7:additional-argument>-u</ns7:additional-argument>
    <ns7:additional-argument>{jobuser}</ns7:additional-argument>
    <ns7:additional-argument>{bin-path}/qsub</ns7:additional-argument>
    <ns7:additional-argument>-W</ns7:additional-argument>
    <ns7:additional-argument>umask=0007</ns7:additional-argument>
  </ns7:qsub>
  <ns7:qstat path="(sudo)"
    <ns7:additional-argument>-u</ns7:additional-argument>
    <ns7:additional-argument>{jobuser}</ns7:additional-argument>
    <ns7:additional-argument>{bin-path}/qstat</ns7:additional-argument>
  </ns7:qstat>
  <ns7:qdel path="(sudo)"
    <ns7:additional-argument>-u</ns7:additional-argument>
    <ns7:additional-argument>{jobuser}</ns7:additional-argument>
    <ns7:additional-argument>{bin-path}/qdel</ns7:additional-argument>
</ns5:pbs-configuration>
```
Note that there are three parameters in the above example that require system-specific values: \{jobuser\}, \{bin-path\} and \{sudo\}. \{jobuser\} should be the user name of the account under which the jobs will execute (the “jobuser” in all of the examples provided), \{bin-path\} should be the absolute path to the directory where the queue executables are located (same as the “sudoers” entry above), and \{sudo\} should be the absolute path to the “sudo” executable (e.g. “/bin/sudo”).

To create the BES with these properties, execute the following:

```
grid create-resource --construction-properties=local:/sudo-native-queue.xml {containerPath}/Services/GeniiBESPortType{/bes-containers/newBES}
```

### F.6. Grid Inter-Operation

One of the strengths of the Genesis II GFFS software is its ability to connect heterogeneous resources into one unified namespace, which provides access to the full diversity of scientific computing facilities via a standardized, filesystem interface. This section describes how to link resources into the GFFS from other sources, such as the Unicore6 BES implementation of EMS and PBS-based queues for job processing.

#### F.6.1. How to Create a BES using Construction Properties

To set up the BES wrapper on a machine that will submit to a queuing system, the user should know properties for the cluster configuration such as memory, number of cores on each node, the maximum slots that can be used to submit jobs on the cluster, and any other relevant options. These properties need to be specified in the construction-properties file which is used while creating a BES resource. Also the grid administrator should have already installed a Genesis II container on the head or submit node of PBS or similar job submission system. A sample construction properties file is below, which we will call cons-props.xml:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
xmlns:ns3="http://vcgr.cs.virginia.edu/GenesisII/bes/environment-export"
xmlns:ns4="http://vcgr.cs.virginia.edu/cmdline-manipulators"
xmlns:ns5="http://vcgr.cs.virginia.edu/native-queue">
  <ns2:fuse-directory xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"/>
  <ns3:environment-export xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"/>
  <ns2:cmdline-manipulators>
    <ns4:manipulator-variation name="default_pwrapper" type="pwrapper"/>
    <ns4:manipulator-variation name="mpich2" type="mpich">
      <ns4:mpich-configuration>
        <ns4:exec-command>mpiexec</ns4:exec-command>
        <ns4:additional-arg>-comm mpich2-pmi</ns4:additional-arg>
      </ns4:mpich-configuration>
      <ns4:supported-spmd-variation>
      </ns4:supported-spmd-variation>
```

The BES can be created using the following command in grid command line:

```
# Create the actual BES resource.
grid create-resource --construction-properties=local://cons-props.xml {containerPath}/Services/GeniiBESPortType {/bes-containers/newBES}

# Link the new BES container into the queue as a resource:
grid ln {/bes-containers/newBES} {queuePath}/resources/{newBES}
```

Once the BES is created, users can be added to the read and execute ACLs of the BES to allow those users to run jobs on that BES.

**F.6.1.1. Job state in shared directory**

In the above construction-properties file, the element `<ns2:nativeq shared-directory="/nfs/shared-directory/bes" provider="pbs">` specifies the shared directory where job state of all the jobs submitted to that BES will be stored. A unique directory gets created for each job when a job gets scheduled on the BES and it is destroyed when the job completes. This path should be visible to all the nodes on the cluster and hence should be on a cluster wide shared directory.

**F.6.1.2. Scratch space in shared directory**
To configure Scratch space on BES, a special file called ScratchFSManagerContainerService.xml specifying path to scratch space should be created in the deployments configuration's cservices directory ($GENII_INSTALL_DIR/deployments/$DEPLOYMENT_NAME/configuration/cservices).

Below is a sample file to configure scratch space:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<container-service class="edu.virginia.vcgr.genii.container.cservices.scratchmgr.ScratchFSManagerContainerService">
  <property name="scratch-directory" value="{/nfs/shared-directory/scratch}"/>
</container-service>
```

If scratch space is configured after BES is configured, then the BES container must be restarted once the scratch directory is set up.

**F.6.1.3. Download Directory (Download manager) in shared directory**

When users submit jobs that stage-in and stage-out files, the BES download manager downloads these files to a temporary download directory. If it is not explicitly configured while setting up the BES, it is created in container's state directory $GENII_USER_DIR/download-tmp. Usually container state directory is stored in local path and download directory should be on a shared directory like Job directory and Scratch directory. Also if $GENII_USER_DIR/download-dir and scratch directory are not on the same partition, BES may not copy/move the stage-in/stage-out files properly between download and Scratch-directory. It is highly advised they be on the same partition.

To configure the download directory, the path should be specified in a special file called DownloadManagerContainerService.xml that is located in the deployment's cservices directory ($GENII_INSTALL_DIR/deployments/$DEPLOYMENT_NAME/configuration/cservices).

Below is a sample file to configure the download directory:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<container-service class="edu.virginia.vcgr.genii.container.cservices.downloadmgr.DownloadManagerContainerService">
  <property name="download-tmpdir" value="/path/to/download-tmpdir"/>
</container-service>
```

If the download directory is configured after the BES is configured, then the BES container must be restarted once the download directory is established.

**F.6.1.4. BES attributes**

A BES can be configured with specific matching parameters to direct jobs that might need these properties specifically to execute the jobs. Example some clusters may support MPI while some clusters may be 32-bit compatible while others can be 64-bit compatible. If jobs need certain requirement to be met, then those jobs will specify the requirements in the JSDL. The queue will
match those jobs to BESes where these attributes are available. To set matching parameter user grid command 'matching-parameters' command.

For example, to add or specify that a particular BES supports MPI jobs, run this command on the queue:

```bash
grid matching-params (/queues/queuePath)/resources/{newBES} "add(supports-mpi, true)"
```

Some other matching parameters that are supported are:

```bash
add(x86_64_compatible, true)
add(GLIBC_2_4_COMPATIBLE, true)
add(blastp-2.2.24, true)
```

### F.6.2. Adding a PBS Queue to a Genesis II Queue

The main consideration for adding a PBS Queue to a Genesis II Queue is to wrap the PBS Queue in a Genesis II BES using a construction properties file. The example cons-props.xml above shows such a file for a PBS Queue with MPI capability. Given an appropriate construction properties file for the system, these steps will create the BES and add it as a resource to the queue.

```bash
# Create the actual BES resource.
grid create-resource --construction-properties=local:/cons-props.xml {containerPath}/Services/GeniiBESPortType {/bes-containers/newBES}

# Link the new BES container into the queue as a resource.
grid ln {/bes-containers/newBES} {/queues/queuePath}/resources/{newBES}

# Adjust the number of jobs that the queue will submit
# to the BES simultaneously.
grid qconfigure {/queues/queuePath} {newBES} {jobsMax}
```

### F.6.3. Adding a Unicore6 BES to a Genesis II queue

#### F.6.3.1. Unicore6 Interoperation with XSEDE approved certificates

The following assumes that the Genesis II Queue container is using an approved XSEDE certificate for its TLS certificate(s) and that only XSEDE MyProxy-based users need to be supported. This also assumes that the Unicore6 gateway, Unicore/X and TSI components are already installed and available for use. The XSEDE UNICORE6 GFFS Installation Guide provides instructions on deploying the Unicore6 installation. The Unicore installation also requires that the Genesis II interopration package "unicore-in-xsede-1.6x.zip" has been installed. These are currently available from the SD&I Activity 123 subversion repository (which is listed on the Activity 123 page at https://www.xsede.org/web/staff/staff-wiki/-/wiki/Main/Activity+-+Integrated+EMS+and+GFFS+Increment+3+Update+-+SDI+Plan+Sep+2012).

Additionally, the user configuring the Unicore BES must be logged in with a grid account that permits creating the necessary links in the GFSS (such accounts include the keystore login for the grid, grid administrator accounts, or groups created for this purpose by the grid administrators).
Add the CA certificate of all valid TLS identities for SAML credentials into the Unicore/X directory-based trust store. This is usually the MyProxy CA certificates at the least:

```bash
# copy the PEM into the Unicore trust store.
cp CAcert1.pem CAcert2.pem ... $UNICORE_INSTALL_PATH/unicorex/conf/trustedIdentityProviders
```

A simpler method than the above can be used to point the Unicore/X trust store at all valid XSEDE CA certs. This involves editing "unicorex/conf/uas.config" to add this line:

```plaintext
genii.trusted.dir=/etc/grid-security/certificates
```

Given that the container’s own TLS certificate is an official XSEDE-approved certificate, the UNICORE gateway trust store should already allow connections from the container. However, the UNICORE Unicore/X component requires one of the above methods to trust the XSEDE user identity certificates before the queue can successfully submit jobs on behalf of XSEDE users.

XSEDE users are mapped to local operating system users as part of Unicore authentication. To enable a new XSEDE user, add the XSEDE portal certificate into the Unicore grid map file. (This may already have been done on official XSEDE hosts.)

```plaintext
# edit the grid-mapfile and add a line similar to this (the grid-mapfile is usually found in /etc/grid-security/grid-mapfile):
"/C=US/O=National Center for Supercomputing Applications/CN={XSEDE_NAME}" {UNIX_USER_NAME}
```

The `{UNIX_USER_NAME}` above is the identity that will be used on the local system for running jobs. This should be the same user that installed the Unicore software. The `{XSEDE_NAME}` above is the XSEDE portal user name for your XSEDE MyProxy identity. This information can be obtained by authenticating with xsedeLogin with the grid client and then issuing a whoami command:

```bash
# user and password will be prompted for on console or in graphical dialog:
grid xsedeLogin
# alternatively they can both be provided:
grid xsedeLogin --username=tony --password=tiger

# try this if there are weird problems with console version of login:
unset DISPLAY
# the above disables a potentially defective X windows display; try logging
# in again afterwards.

# finally... show the XSEDE DN.
grid whoami --oneline
```

Acquire the CA Certificate that generated the certificate being used for the Unicore Gateway and for Unicore/X (this can be one certificate, or two if they are generated separately by different CAs). Add that into the trusted-certificates directory on the Genesis Queue container. Repeat this step on any containers or clients where you would like to be able to directly connect to the Unicore BES. If all users will submit jobs via the queue, then only the queue container needs to be updated:

```bash
# Example with unicore container using u6-ca-cert.pem and a Genesis
# deployment named ‘current_grid’. In reality, this may involve more than
# one host, which would lead to a file transfer step and then a copy.
cp $UNICORE_INSTALL_DIR/certs/u6-ca-cert.pem "$GENII_INSTALL_DIR/deployments/current_grid/security/trusted-certificates"
```
Alternatively, since the Unicore TLS certificate is assumed to be generated using XSEDE CA certificates, then the following step is sufficient (rather than copying individual certificates):

```bash
# assumes the XSEDE certificates are installed at /etc/grid-security/certificates.
mv $GENII_INSTALL_DIR/deployments/current_grid/grid-certificates \
$GENII_INSTALL_DIR/deployments/current_grid/old-grid-certificates
ln -s /etc/grid-security/certificates \
$GENII_INSTALL_DIR/deployments/current_grid/grid-certificates
```

Now that mutual trust between the Unicore6 BES and the Genesis II GFFS has been established, link the Unicore BES into the Genesis II namespace with the following command:

```bash
grid mint-epr --link=/bes-containers/u6-bes \ 
  --certificate-chain=local:{/path/to/unicorex-tls-cert} {bes-url}
```

The unicorex-tls-cert is the certificate used by the Unicore/X container for TLS (aka SSL) communication. Note that this is different from the Unicore CA certificate in the last step; this should be the actual TLS certificate and not its CA. Also, be sure to provide the Unicore/X TLS certificate rather than the Gateway TLS certificate (if these are different); otherwise trust delegations cannot be extended by Unicore/X (in the uas-genesis component) and grid stage-in and stage-out will not work in submitted jobs. The bes-url has the following form (as documented in the UNICORE installation guide, at http://www.unicore.eu/documentation/manuals/unicore6/manual_installation.html):

```bash
https://{u6GatewayHost}:{u6GatewayPort}/{u6XserverName}/services/BESFactory?res=default_bes_factory
```

This is an example of the Unicore EPR minting process combined in one command:

```bash
grid mint-epr --link=/bes-containers/unicore-bes \ 
  --certificate-chain=local:$HOME/unicorex-tls-cert.pem \ 
```

Once the Unicore BES is linked into the grid, it can be linked into an existing queue as a resource:

```bash
grid ln {/bes-containers/u6-bes} {/queues/theQueue}/resources/{resource-name}
```

The resource-name above can be chosen freely, but is often named after the BES that it links to.

Adjust the number of jobs that the queue will submit to the BES simultaneously (where jobsMax is an integer):

```bash
grid qconfigure {/queues/theQueue} {resource-name} {jobsMax}
```

To remove the Unicore BES at a later time, it can be unlinked from the queue by calling:

```bash
grid unlink {/queues/theQueue}/resources/{resource-name}
```

**F.6.3.2. Unicore6 Interoperation with non-XSEDE Certificates**

The steps taken in the previous section are still necessary for setting up a Unicore6 BES when one does not possess XSEDE-approved certificates. However, to configure security appropriately to let users and GFFS queues submit jobs, there are a few additional steps required for non-XSEDE grids.
For each TLS identity that will connect directly to the BES, add the CA certificate that issued the certificate into the Gateway and Unicore trust stores. If there are multiple certificates in the CA chain, then each should be added to the trust stores.

```plaintext
# Add CA certificate to the gateway's trust store folder. Consult the
# gateway/conf/security.properties file to determine the actual location
# of the gateway trust store directory.
cp users-CA-cert.pem {/gateway/cert/store}

# Add the CA certificate to the directory trust store for Unicore/X.
cp users-CA-cert.pem 
   /$UNICORE_INSTALL_PATH/unicorex/conf/trustedIdentityProviders
```

Once the users’ CA certificates and the queues’ CA certificates have been added, the Unicore6 BES can be configured as described in the prior section, and then it should start accepting jobs directly from users as well as from the queue container.

F.6.3.3. **Debugging Unicore6 BES Installations**

If there are problems inter-operating between the GFFS and Unicore, then it can be difficult to determine the cause given the complexity of the required configuration. One very useful tool is to increase logging on the Unicore6 servers and GFFS containers involved.

For Unicore, the "debug" level of logging provides more details about when connections are made and why they are rejected. This can be updated in the gateway/conf/logging.properties file and also in the unicorex/conf/logging.properties file. Modify the root logger line in each file to enable DEBUG logging as follows:

```plaintext
log4j.rootLogger=DEBUG, A1
```

For the Genesis II GFFS, the appropriate logging configuration files are in the installation directory in lib/production.container.log4j.properties and lib/production.client.log4j.properties. For each of those files, debug-level logging can provide additional information about job submissions by changing the rootCategory line accordingly:

```plaintext
log4j.rootCategory=DEBUG, LOGFILE
```

**F.6.4. Adding an MPI Cluster to a Grid Queue**

F.6.4.1. **Create a BES with MPI configuration for the MPI-enabled cluster**

Create a Native Queue BES using a construction properties file (for use with BES creation) that specifies the MPI types supported by the cluster along with the syntax for executing MPI jobs and any special commandline arguments. The “manipulator-variation” structure under the “cmdline-manipulators” structure specifies the MPI related details for the cluster. The “supported-spmd-variation” field gives the SPMD type as per the specification. The “exec-command” field specifies the execution command for running MPI jobs on the cluster. The “additional-arg” field specifies any additional command-line arguments required to run an MPI job on the cluster. An example construction properties file for the Centurion Cluster is provided below.
<?xml version="1.0" encoding="UTF-8"?>
<genii:construction-parameters
    xmlns:genii="http://vcgr.cs.virginia.edu/construction-parameters"
    xmlns:bes="http://vcgr.cs.virginia.edu/construction-parameters/bes"
    human-name="PBS-Centurion BES with MPI and Pwrapper">
    <bes:post-execution-delay>15.000000 Seconds</bes:post-execution-delay>
    <bes:pre-execution-delay>15.000000 Seconds</bes:pre-execution-delay>
    <bes:resource-overrides>
        <bes:cpu-count>2</bes:cpu-count>
        <bes:physical-memory>206000000.000000 B</bes:physical-memory>
    </bes:resource-overrides>
    <bes:nativeq provider="pbs"
        shared-directory="/home/gbg/shared-directory"
        xmlns:nq="http://vcgr.cs.virginia.edu/native-queue">
        <nq:pbs-configuration queue-name="centurion">
            <nq:qsub/>
            <nq:qstat/>
            <nq:qdel/>
        </nq:pbs-configuration>
    </bes:nativeq>
    <bes:cmdline-manipulators
        xmlns:clm="http://vcgr.cs.virginia.edu/cmdline-manipulators">
        <clm:manipulator-variation type="pwrapper">
            <clm:exec-command>mpiexec</clm:exec-command>
            <clm:additional-arg>-p4</clm:additional-arg>
        </clm:manipulator-variation>
        <clm:manipulator-variation type="mpich">
            <clm:exec-command>mpiexec</clm:exec-command>
            <clm:additional-arg>-p4</clm:additional-arg>
        </clm:manipulator-variation>
        <clm:manipulator-name>mpil</clm:manipulator-name>
        <clm:manipulator-name>pwrapper</clm:manipulator-name>
    </bes:cmdline-manipulators>
</genii:construction-parameters>
F.6.4.2. Add MPI-Enabled BES to Queue

Add BES resource to queue like any other BES.

F.6.4.3. Set matching parameters

Set up matching parameters (as described in the matching parameters section) and use them for matching jobs to this BES. This is required because Genesis II queues are not MPI-aware.

F.6.5. Establishing Campus Bridging Configurations

Educational institutions may wish to participate in the XSEDE grid to share computational resources, either by utilizing the resources XSEDE already has available or by adding resources from their campus computing clusters to the XSEDE grid for others’ use. There are a few requirements for sharing resources in this way, and they are described in the following sections.

F.6.5.1. Obtaining an XSEDE Portal ID

One primary requirement for using XSEDE resources is to obtain an XSEDE portal ID. The portal ID can be obtained from the XSEDE website at http://xsede.org. Once the ID is obtained, the user’s grid account needs to be enabled by a grid admin. The XSEDE ID can then be used to log into the XSEDE grid.

A grid user can create files and directories within the GFFS, which is required for adding any new resources to the grid. Further, the XSEDE grid account enables the grid user to be given access to existing XSEDE grid resources.

F.6.5.2. Link Campus Identity Into XSEDE Grid

Another primary requirement for campus bridging is to link the campus identity for a user into the XSEDE grid. After the campus user has obtained an XSEDE grid account, she will have a home folder and a user identity within the XSEDE grid. However, at this point the XSEDE grid has no connection to the user’s identity on campus. Since the campus identity may be required to use the campus resources, it is important that the user's credentials wallet contain both the campus and XSEDE identities.

For example, campus user identities may be managed via a Kerberos server. By following the instructions in the section on “Using a Kerberos STS”, an XSEDE admin has linked the STS for campus user “hugo” at “/users/hugo”. Assuming that the user's XSEDE portal ID is “drake” and that identity is stored in “/users/drake”, the two identities can be linked together in the XSEDE grid with:

```
# give drake the right to use the hugo identity.
grid chmod /users/hugo +rx /users/drake
# link campus id under xsede id.
```
The XSEDE user drake will thus automatically attain the identity of the campus user hugo when drake logs in. After this, drake will seamlessly be able to utilize both the XSEDE grid resources as drake and the campus resources as hugo.

F.6.5.3. Link Campus Resources Into XSEDE Grid

Campus researchers may wish to share their local compute resources with others in the XSEDE grid. In order to do this, the campus user should wrap the resource as a BES service and link it to the grid as described in the section on “How to Create a BES with Construction Properties”. That resource can then be added to a grid queue or queues by following the steps in the section “Linking a BES as a Queue Resource”.

Assuming that the BES is successfully linked to a grid queue, users with rights on the grid queue should be able to send compute jobs to the linked campus resource automatically. If it is desired to give an individual user the privilege to submit jobs directly to the BES, this can be done with the “chmod” tool. For example, the user “drake” could be given access to a newly-linked PBS-based BES as follows:

```
grid chmod /bes-containers/GridU-pbs-bes +rx /users/drake
```

F.6.5.4. Utilizing XSEDE Resources

Campus researchers may wish to use the resources already available in the XSEDE grid. At its simplest, this is achieved by adding the user to a grid group that has access to the queue possessing the desired resources. The user can also be given individual access to resources by using chmod, as detailed in the last section.

This situation can become more complex when the resources are governed by allocation constraints or other jurisdictional issues. This may require the user to obtain access through consultation with the resource owners, or to take other steps that are generally beyond the scope of this document.
G. Grid Management

Once a grid configuration has been established and the queuing and computational resources are set up, there are still a number of topics that come up for day to day grid operation. These include managing users and groups, performing backups and restores on container state, grid accounting and other topics. These will be discussed in the following sections.

G.1. User and Group Management

User and group identities exist in the GFFS as they do for most filesystems. These identities can be given access rights on grid resources using some familiar patterns from modern day operating systems. It is useful to keep in mind that a user or a group is simply an identity provided by an IDP (Identity Provider) that the grid recognizes. IDPs can be provided by Genesis II, Kerberos and other authentication servers.

Creating and managing user identities in the GFFS requires permissions on the {containerPath} in the following commands.

G.1.1. Creating Grid Users

To create a new user that can log in to the grid, use the following steps:

```
# Create the user's identity.
grid create-user {containerPath}/Services/X509AuthnPortType \
  {userName} --login-name={userName} --login-password={userPassword} \ 
  --validDuration={Xyears|Ydays}

# Link the user identity into the /users folder (this is a grid convention).
grid ln {containerPath}/Services/X509AuthnPortType/{userName} \
  /users/{userName}

# Take away self-administrative rights for user.
grid chmod /users/{username} 5 /users/{username}

# Create a home folder for the user's files.
grid mkdir /home/{userName}
grid chmod /home/{userName} "+rwx" /users/{userName}
```

Example of Steps in Action:

```
grid create-user /containers/poe/Services/X509AuthnPortType drake \
  --login-name=drake --login-password=pwdx --validDuration=1years
grid ln /containers/poe/Services/X509AuthnPortType/drake /users/drake
grid chmod /users/drake 5 /users/drake
grid mkdir /home/drake
grid chmod /home/drake "+rwx" /users/drake
```

G.1.2. Creating a Group
Group identities in the grid are very similar to user identities. They can be given access to other resources and identities. They are created using the following steps:

```bash
# Create the group identity
grid idp {containerPath}/Services/X509AuthnPortType {groupName}

# Link the group identity to the /groups folder (by convention).
grid ln {containerPath}/Services/X509AuthnPortType/{groupName} /groups/{groupName}
```

Example of Steps in Action:

```
grid idp /containers/poe/Services/X509AuthnPortType uva-idp-group
grid ln /containers/poe/Services/X509AuthnPortType/uva-idp-group /groups/uva-idp-group
```

### G.1.3. Adding a User to a Group

When they are created, groups have no members. Users and other groups can be added to a group using the following commands:

```bash
# Give user permissions to the group.
grid chmod /groups/{groupName} +rx /users/{userName}

The userName will have read and execute access to the groupName afterwards. Instead of /users/{userName}, a group could be used instead.

# Link the group identity under the user identity to enable automatic login.
grid ln /groups/{groupName} /users/{userName}/{groupName}

The next time the userName logs in, she will also automatically log into the group identity. Note that this will fail if the first step above (to grant access) was not performed.
```

Example of Steps in Action:

```
grid chmod /groups/uva-idp-group +rx /users/drake
grid ln /groups/uva-idp-group /users/drake/uva-idp-group
```

### G.1.4. Removing a User from a Group

Taking a user back out of a group is basically the reverse process of adding:

```bash
# Unlink the group identity under the user's identity.
grid unlink /users/{userName}/{groupName}

# Remove permissions on the group for that user.
grid chmod /groups/{groupName} 0 /users/{userName}
```

Example of Steps in Action:

```
grid unlink /users/drake/uva-idp-group
grid chmod /groups/uva-idp-group 0 /users/drake
```

### G.1.5. Removing a User

Occasionally a grid user needs to be removed. These steps will erase their identity:
# Remove the home folder for the user's files.
grid rm -rf /home/{userName}

Note that this will destroy all of the user's files! Do not do this if their data is intended to be retained.

# Unlink any groups that the user was added to.
grid unlink /users/{userName}/{groupName}

# Unlink the user identity from the /users folder.
grid unlink /users/{userName}

# Delete the user's identity.
grid rm -f {containerPath}/Services/X509AuthnPortType/{userName}

Example of Steps in Action:

```bash
grid rm -rf /home/drake
grid unlink /users/drake/uva-idp-group
grid unlink /users/drake
grid rm -f /containers/poe/Services/X509AuthnPortType/drake
```

G.1.6. Removing a Group

It is much more serious to remove a group than a simple user, because groups can be used and linked in numerous places. This is especially true for resources, which administrators often prefer to control access to using groups rather than users. But in the eventuality that a group must be removed, here are the steps:

Unlink any users from the group:

```bash
grid unlink /users/{userName}/{groupName}
```

Omitting the removal of a group link from a user's directory may render the user unable to log in if the group is destroyed.

```bash
# Clear up any access control lists that the group was involved in.
grid chmod [/path/to/resource] 0 /groups/{groupName}

# Remove the group identity from the /groups folder.
grid unlink /groups/{groupName}

# Destroy the group identity itself.
grid rm -f {containerPath}/Services/X509AuthnPortType/{groupName}
```

Example of Steps in Action:

```bash
# repeat per occurrence of the group under every user...
grid unlink /users/drake/uva-idp-group
grid unlink /users/joe/uva-idp-group

# repeat per occurrence of the group in object ACLs...
grid chmod /queues/poe-queue 0 /groups/uva-idp-group

# finally, remove the group.
grid unlink /groups/uva-idp-group

grid rm -f /containers/poe/Services/X509AuthnPortType/uva-idp-group
```
G.1.7.  Changing a User's Password

It is often necessary to change a user's password after one has already been assigned. For the XSEDE logins using Kerberos and MyProxy, this cannot be done on the Genesis II side; the user needs to make a request to the XSEDE administrators (for more information, see http://xsede.org). But for standard Genesis II grid IDP accounts, the password can be changed using the following steps:

First remove the existing password token using the grid client, started with:

```
grid client-ui
```

Navigate to the appropriate user in the /users folder, and remove all entries that are marked as (Username-Token) in the security permissions.

(Alternatively, this command would normally work for the same purpose, but currently there is a bug that prevents it from removing the existing username&password token:

```
grid chmod {/users/drake} 0 --username={drake} --password={oldPassword}
```

This may be fixed in a future revision, and would work for scripting a password change.)

After removing the previous username token, add a new token for the user that has the new password:

```
grid chmod {/users/drake} +x --username={drake} --password={newPassword}
```

Once this is done, the new login can be tested to ensure it works:

```
grid logout --all
grid login --username={drake} --password={newPassword}
```

G.1.8.  Using a Kerberos STS

The grid administrator can create an STS based on Kerberos that will allow users to use the Kerberos identity as their grid identity. This requires an existing Kerberos server and an identity on that server. To create an STS for the grid that uses the server, do the following:

```
# Create the Kerberos STS.
grid idp -kerbRealm={KERB-REALM.COM} -kerbKdc={kdc.kerb-realm.com} {containerPath}/Services/KerbAuthnPortType {kerberosUserName}
# User can then log in like so...
grid login {containerPath}/Services/KerbAuthnPortType/{kerberosUserName}
```

The first step created an STS object in the GFFS under the specified Kerberos service and user name. This path can then be relied upon for logins as shown. Linking to a /users/kerberosUserName folder (as is done for IDP logins) may also be desired. See the next section for a more complete example of how an XSEDE login is created using both Kerberos and MyProxy.

Example of Steps in Action:
G.1.9. Creating XSEDE Compatible Users

This procedure is used to create user identities that are suitable for use with the XSEDE grid. Users of this type must log in using the “xsedeLogin” command. It is necessary for the user’s account to be enabled on both the XSEDE kerberos server (which requires an XSEDE allocation) and the XSEDE myproxy server.

To create an XSEDE compatible user as an administrator, follow these steps (if there is no administrator for Kerberos Users yet, see the end of this section):

```plaintext
# Login as an existing grid user with appropriate permissions to create new users.
grid login --username={adminUser}

# Create new xsede user STS:
grid idp --kerbRealm=TERAGRID.ORG --kerbKdc=kerberos.teragrid.org
  {containerPath}/Services/KerbAuthnPortType {portalID}

# Link user into grid.
grid ln {containerPath}/Services/KerbAuthnPortType/{portalID} \
  /users/xsede.org/{portalID}

# Take away self-administrative rights for user.
grid chmod /users/xsede.org/{portalId} 5 /users/xsede.org/{portalId}

# Create user's home directory.
grid mkdir /home/xsede.org/{portalID}

# Give user all rights on home directory.
grid chmod /home/xsede.org/{portalID} +rwx /users/xsede.org/{portalID}
```

Example of the steps in action:

```plaintext
grid idp --kerbRealm=TERAGRID.ORG --kerbKdc=kerberos.teragrid.org
  /resources/xsede.org/containers/poe/Services/KerbAuthnPortType drake
grid ln /resources/xsede.org/containers/poe/Services/KerbAuthnPortType/drake \
  /users/xsede.org/drake
grid chmod /users/xsede.org/drake 5 /users/xsede.org/drake
grid mkdir /home/xsede.org/drake
grid chmod /home/xsede.org/drake +rwx /users/xsede.org/drake
```

Once the user identity has been created using the above process, the user can be added to groups or given access rights on resources exactly like other grid users.

These steps have been encapsulated in a script in the xsede_tools for the XSEDE grid:

```plaintext
grid script local:$/XSEDE_TEST_ROOT/tools/xsede_admin/create-xsede-user.xml \
  {portalUserName}
```
The process for removing an XSEDE compatible user is identical to the process for removing a standard grid user (Section G.1.5), except for the last step. For an XSEDE compatible user, the last step is:

```
# delete the user's identity.
grid rm -f {containerPath}/Services/KerbAuthnPortType/{portalID}
```

G.1.9.1. Creating an administrator for XSEDE users

The following is only applicable for grid administrators. To enable an existing user (referred to as {newadmin} below) to create new XSEDE users, follow these steps:

```
# Login as the super-user keystore login for the grid.
grid keystoreLogin local:admin.pfx --password={thePassword}

# Give the user permissions to create Kerberos accounts:
grid chmod /groups/xsede.org/gffs-admins +rx /users/xsede.org/{newadmin}
grid ln /groups/xsede.org/gffs-admins /users/xsedeorg/{newadmin}/gffs-admins
grid chmod /groups/xsede.org/gffs-amie +rx /users/xsede.org/{newadmin}
grid ln /groups/xsede.org/gffs-amie /users/xsedeorg/{newadmin}/gffs-amie
```

After these steps, the user is capable of Kerberos user administration.

G.1.10. Configuring Kerberos Authorization on a Container

By itself, authenticating to a Kerberos KDC for a user is not enough to ensure that the user is properly vetted. Kerberos authorization to a service principal is also needed for the STS to fully authenticate and authorize the user against the Kerberos realm.

This needs to be configured on each container that participates as a Kerberos STS. For a small grid, this may only be the root container of the RNS namespace, or a complex grid may have several STS containers. Each of these STS containers must have a separate service principal created for it, and the container must use the keytab corresponding to that service principal. Both the service principal and the keytab file must be provided by the realm's Kerberos administrator.

Once the keytab and service principal have been acquired, the container owner can set up the container to use them by editing the "security.properties" file found in the deployment’s "configuration" folder. This assumes the container is using the “Split Configuration Model” provided by the interactive installer (see below for configuring RPM installs with the “Unified Configuration Model”). The keytab file should be stored in the deployment’s “security” folder, rather than the “configuration” folder. The security.properties file has internal documentation to assist configuring, but this section will go over the important details.

When using an RPM or DEB install package in the “Unified Configuration Model”, the file “$GENII_USER_DIR/installation.properties” should be edited instead of the deployment's "security.properties”. The storage folder for keytab files based on an RPM or DEB install is “$GENII_USER_DIR/certs” instead of the deployment’s security folder.

When requesting a service principal from the Kerberos realm’s administrator, it is recommended to use the following form:
This naming convention makes it clear that the service in question is “gffs-sts”, or the GFFS Server Trust Store. It includes the hostname of the STS container as well as the Kerberos realm in which the service principal is valid. An example of a “real” service principal is below:

```
gffs-sts/KHANDROMA.CS.VIRGINIA.EDU@TERAGRID.ORG
```

This is the service principal that a testing machine uses to authenticate to the TERAGRID.ORG realm maintained by XSEDE.

The Kerberos administrator will also provide a keytab file for this service principal. It is crucial that this keytab file be used on just a single STS host. This file does not participate in replication of Kerberos STS within the GFFS, and it should not be copied off machine or replicated by other means.

The container’s security.properties file (or installation.properties) records the container’s Kerberos authorization configuration in two lines per STS host. One specifies the service principal name, and the other the keytab file. Each entry has the Kerberos realm name appended to the key name, making them unique in case there are actually multiple Kerberos realms being used by the same container.

The key name “gffs-sts.kerberos.keytab.REALMNAME” is used to specify the keytab file. The keytab should be located in the “security” folder of the deployment.

The key name “gffs-sts.kerberos.principal.REALMNAME” is used to specify the service principal name for the realm.

Here is another real-world example for the “khandroma” service principal (lines have been split for readability, but these should each be one line in the configuration and should not contain spaces before or after the equals sign):

```
gffs-sts.kerberos.keytab.TERAGRID.ORG=KHANDROMA.CS.VIRGINIA.EDU@TERAGRID.ORG.gffs-sts.keytab

gffs-sts.kerberos.principal.TERAGRID.ORG=gffs-sts/KHANDROMA.CS.VIRGINIA.EDU@TERAGRID.ORG
```

The name of the keytab file is provided without any path; the file will automatically be sought in the same deployment’s “security” folder (or state directory “certs” folder for RPM/DEB install).

Testing the Kerberos configuration can be done by creating an XSEDE compatible user (see prior section) and attempting to log in as that user. This requires a portal account with XSEDE and an XSEDE allocation. Warning: it may not be appropriate to test the Kerberos authentication yet if setting up an XSEDE-style grid; testing should not be done until after the STS migration process has occurred.

It may be interesting to note that even after divulging all this critical security information about the khandroma container in the discussion above, no breach of security has been accomplished. This is
true because the keytab for this service principal has not been provided, and one will not be able to successfully authenticate to this service principal without it.

If a keytab is accidentally divulged, that is not a total calamity, but it is important to immediately stop that container from authorizing the Kerberos realm affected by the exposed keytab file and to request a new keytab from the Kerberos administrator. Once the new keytab is deployed in the container, normal authorization can resume. After the Kerberos administrator has generated the new keytab, the older one will no longer authorize properly and so the security risk has been mitigated.

G.1.11. Setting Up an InCommon STS

As described in the section “Logging in with InCommon” (Section E.2.2.6), the iclogin tool allows a user to log in using credentials for an InCommon IDP. In order to accommodate this tool, a link must be established between the InCommon identity and another existing grid identity which has access permissions on the intended resources. The target identity may be any of the existing STS types (Kerberos, X509, etc).

The first step is to determine the InCommon identity's Certificate Subject, as follows:

Navigate a browser to https://cilogon.org, and log on with the InCommon credentials. For example, the user might select the ProtectNetwork identity provider, and then click the “Log On” button. This will redirect the browser to that IDP’s login page. The user will then provide appropriate credentials for that IDP to login. The browser will then redirect to the CILogin page. At the top of the page is listed the Certificate Subject for the current user. For example, for an example user “inCommonTester” this string might be:

```
/DC=org/DC=cilogon/C=US/O=ProtectNetwork/CN=IC Tester A1234
```

This information may also be retrieved from an instance of the user's certificate, if the administrator has been provided with a copy for this purpose.

Next, the administrator should obtain a copy of the CILogin.org “Basic” CA certificate from https://cilogon.org/cilogon-basic.pem. From the command line, run the following command:

```
wget https://cilogon.org/cilogon-basic.pem
```

This will place a copy of the certificate in the current local directory.

Assuming the administrator is currently logged in to his own grid credentials, the next step is to add execute permissions to the target credentials for the CILogin certificate. Using the example certificate subject above, and an example XSEDE STS at “/users/xsede.org/xsedeTester”, the administrator would run the following command:

```
grid chmod /users/xsede.org/xsedeTester +x local:cilogon-basic.pem \ 
--pattern="DC=org,DC=cilogon,C=US,O=ProtectNetwork,CN=IC Tester A1234"
```

Note how the “pattern” string is the Certificate Subject returned by cilogon.org, with the leading slash removed, and all other slashes replaced with commas.
Finally, place a link to the target grid credentials in the InCommon user directory. Using the example credentials, the administrator would run the following command from the command line:

```
grid ln /users/xsede.org/xsedeTester /users/incommon.org/inCommonTester
```

The user may now authenticate using the iclogin tool and their InCommon IDP's credentials.

Note that, at this time, the STS link must be in the "/users/incommon.org" directory, and must be named with the InCommon IDP username used to log in. The iclogin tool assumes this location when looking up the grid credentials once the IDP authentication is complete. A more robust solution for linking InCommon identities with grid credentials is in development.

### G.2. Container Management

The grid containers are important assets that the grid administrator must ensure continue to operate, even in the face of hardware failures. Thus it is important to have backups for the container's run-time state, especially for those containers that hold critical assets for the GFFS. Thus it is especially important that the root container is backed up, because there really is no grid without it. The following sections discuss how to stop a container, how to back it up and restore it, and how to start the container running again. The backup procedure should be done regularly on all critical containers.

#### G.2.1. How to Stop a Grid Container

The grid container process does not have a shutdown command as such, but it responds to the control-C (break) signal and stops operation. There are many different methods that would work to cause the container to shut down. The easiest case is for when the Genesis II Installation Program was used to install the container, but for source-based installs we also document how to use the Linux command-line tools to shut the container down and how to use a script in the XSEDE tests to stop the container.

##### G.2.1.1. Stopping a Container When Using the Interactive or RPM/DEB Installer

If the container was installed via the interactive installation program or from Linux packages, then stopping it is quite simple:

```
$GENII_INSTALL_DIR/GFFSContainer stop
```

##### G.2.1.2. Stopping a Container Using Linux Commands

If the container was installed from source, it can be stopped with this procedure. This queries the process list in the operating system and sends a break signal to the Genesis II Java processes:

```
# List the Java processes that are running:
ps wuax | grep -i java | grep -i $USER

# Among those processes, find ones that mention 'genii-container-application.properties'

# For each of those process numbers, run the command:
```
### G.2.1.3. Stopping a Container Using the XSEDE Tests

If the user has access to the XSEDE tests and tools (see the section on Getting the XSEDE Tests for more details), then terminating the Genesis II processes is a bit easier than the above shell commands:

```bash
bash $XSEDE_TEST_ROOT/library/zapgenesis_javas.sh
```

The above finds the running processes similarly to the manual steps (in the previous section) and stops the processes. To ensure they are gone, one can run this command:

```bash
bash $XSEDE_TEST_ROOT/library/listgenesis_javas.sh
```

If that command has no output, then no Genesis II processes are left.

### G.2.2. How to Start a Grid Container

The method for starting a grid container depends on what type of container is installed. If the container is installed from source, then the commands for starting it are:

```bash
cd $GENII_INSTALL_DIR
bash runContainer.sh &>/dev/null &
```

If instead, the container was installed using the Genesis II installation program or an RPM package, then starting the container is done using this step:

```bash
$GENII_INSTALL_DIR/GFFSContainer start
```

If the container is already running and needs to be restarted, then execute this command:

```bash
$GENII_INSTALL_DIR/GFFSContainer restart
```

### G.2.3. How to Backup a Genesis II Grid Container

Archiving the data from the root GFFS container can take hours, or even days, depending on the amount of data stored on the root. This may need to be taken into account for scheduling the container down time.

#### G.2.3.1. Automated Container Backup

The backup process has been automated in a script available in the XSEDE Tests (documented in section I), located in $XSEDE_TEST_ROOT/library/backup_container_state.sh. The container should manually be stopped before running the script, and manually restarted afterwards. For example:
G.2.3.2.  Manual Container Backup

The procedure below describes how to save a snapshot of a Genesis II container’s run-time state. This includes all of its databases, which in turns contain the RNS folders and ByteIO files that live on the container. These steps should work with any container.

When backing up the root GFFS container’s data, note that this can be a truly huge amount of data. If users tend to rely on storing their data files in their home folder, and that folder is located on the root GFFS container, then the administrator is backing up all of those data files when the root container is backed up. This is one reason it is recommended to share the load for home folders by storing them across other containers (see the section on “Where Do My Files Really Live” for more details).

To backup a container, use the following steps. Note that it is expected that GENII_USER_DIR is already set to the right location for this container:

First, stop the container as described in the previous section.

Then back up the container:

```
# Zip up the container's state directory.
zip -r $HOME/container_bkup_$(date +%Y-%m-%d-%H%M).zip $GENII_USER_DIR
# or use tar instead...
tar -czf $HOME/container_bkup_$(date +%Y-%m-%d-%H%M).tar.gz $GENII_USER_DIR
```

Start up the container again, as described in section G.2.2.

G.2.4.  How to Restore a Genesis II Grid Container

G.2.4.1.  Automated Container Restore

The restore process above has been automated in a script available in the XSEDE Tests (documented in section I), located in $XSEDE_TEST_ROOT/library/restore_container_state.sh. The restore script relies on the backup having been produced by the corresponding backup_container_state.sh script. The container should manually be stopped before running the restore script, and manually restarted afterwards.

There are two restoration scenarios that may be encountered; either the container data has been trashed, or the installation itself has been trashed. This first situation, where only the container data needs to be restored, is taken care of by the basic restoration process:

```
source ~/GenesisII/set_gffs_vars  # replace ~/GenesisII with install path.
$GENII_INSTALL_DIR/GFFSContainer stop
bash $XSEDE_TEST_ROOT/library/restore_container_state.sh \\
$HOME/gffs_state_backup....tar.gz
```
If the installation itself has been damaged, then additional steps may be needed. Note that this should only ever be a concern for an interactive installation with the “Split Configuration” model; for the RPM installation or Unified Configuration installation, the above process is sufficient. But the split configuration approach stores some configuration data for the container in the installation directory, and more steps are needed to completely restore both the damaged data and configuration.

Before the “split configuration” restore is attempted, a healthy installation of the appropriate version of Genesis II GFFS should be installed. This installation does not need to be configured identically to the container being restored, as the configuration information will be put back into place in the next steps. Once the installation is available, these steps should perform a full repair of the configuration:

```
$GENII_INSTALL_DIR/GFFSContainer start

source ~/GenesisII/set_gffs_vars    # replace ~/GenesisII with install path.
$GENII_INSTALL_DIR/GFFSContainer stop
bash $XSEDE_TEST_ROOT/library/restore_container_state.sh \\ $HOME/gffs_state_backup...tar.gz

$GENII_INSTALL_DIR/GFFSContainer start
```

For deployments other than XCG or XSEDE, the actual deployment name of “current_grid” may differ. The real deployment name will be visible in the deployments folder of the install.

### G.2.4.2. Manual Container Restore

When the grid container has been backed up and saved at an external location, the grid administrators are protected from catastrophic hardware failures and can restore the grid to the state of the last backup. This section assumes that the administrator is in possession of such a backup.

First, stop the container as described in the section “How to Safely Stop a Grid Container”.

```
# Make a temporary folder for storing the state.
mkdir $HOME/temporary
cd $HOME/temporary

# Clean up any existing run-time state and recreate the state directory.
rm -rf $GENII_USER_DIR
mkdir $GENII_USER_DIR

# Extract the container's state directory from the archive.
unzip $HOME/container_backup_{backupDate}.zip
# or...
tar -xR $HOME/container_backup_{backupDate}.tar.gz

# Move the files into the appropriate place.
mv {relative/path/to/userDir}/* $GENII_USER_DIR

# Clean up the temporary files.
cd
```
Start up the container again, as described in section G.2.2.

G.2.5. Replication of GFFS Assets

Replication in the GFFS can be used for fault tolerance and disaster recovery. For example, replication can be used to create a fail-over system, where the loss of services of a crucial container does not necessarily mean that the grid is down. Replication can also be used to create a backup system that automatically copies assets that are modified on one container onto a container at a different physical location, ensuring that even the total destruction of the first container's host does not lead to data loss.

This section describes how to set up replicated files and directories, and how to create the resolvers that are used to locate replicated assets.

G.2.5.1. Replicating a New Directory Hierarchy

**USE CASE: The user is creating a new project.** The project starts with an empty home directory, such as /home/project. The project's home directory should be replicated.

In this case, run these commands:

```
mkdir /home/project
resolver -p /home/project /containers/backup
replicate -p /home/project /containers/backup
```

The “resolver” command defines a “policy” that whenever a new file or subdirectory is created under /home/project, that new resource will be registered with a resolver in /containers/backup.

The “replicate” command defines a “policy” that whenever a new file or subdirectory is created under /home/project, that new resource will be replicated in /containers/backup.

That's it. Whenever a file or directory is created, modified, or deleted in the directory tree in the first container, that change will be propagated to the backup container. Whenever a security ACL is modified in the first container, that change will be propagated too. If the first container dies, then clients will silently fail-over to the second container. If resources are modified on the second container, then those changes will be propagated back to the first container when possible.

G.2.5.2. Replicating an Existing Directory Hierarchy

**USE CASE: The project already exists.** There are files and directories in /home/project. These resources should be replicated, as well as any new resources that are created in the directory tree.

In this case, simply add the -r option to the resolver command:

```
resolver -r -p /home/project /containers/backup
```
The “resolver” command registers all existing resources with a resolver, and it defines the policy for new resources. The “replicate” command replicates all existing resources, and it defines the policy for new resources.

**G.2.5.3. Choose Specific Resources to Replicate**

**USE CASE: The user wants to replicate a handful of specific resources.** No new replication policies (or auto-replication of new resources) are desired.

In this case, omit the -p option:

```
resolver /home/project/filename /containers/backup
replicate /home/project/filename /containers/backup
```

This case is only useful for certain unusual setups involving hard links or other rarities.

In general, if fail-over is enabled for some file, then it should also be enable for the file's parent directory. In other words, the directions for replicating an existing directory hierarchy should be used.

**G.2.5.4. Create a Named Resolver**

**USE CASE: The user wants to create a resolver for replicated files and directories.** Or the user wants to give other users access to a resolver, so that those users can create new replicas that can be used for failover.

In this case, create a resolver resource using the create-resource command:

```
create-resource /containers/primary/Services/GeniiResolverPortType /etc/myResolver
# Now, the resolver can be replicated.
replicate /etc/myResolver /containers/backup
# And the resolver's ACLs can be modified.
chmod /etc/myResolver +rwx /users/sal
# To use a named resolver, specify the resolver (by pathname) # rather than the container on the resolver command line.
resolver -p /home/project /etc/myResolver
```

**G.2.5.5. Replicating Top-Level Folders for a Grid**

**USE CASE: The user is configuring a new grid** using Genesis II and would like the top-level folders to be replicated, including the root folder (/) and the next few levels below (/home, /users, /groups, etc.).

Adding a replicated root makes the most important top-level folders available through the resolver. Should the root GFFS container be unavailable, each of the items in the replicated folders is still available from the mirror container. Currently only grid administrators may add replication in the RNS namespace for the XSEDE grid.
Prerequisites for Top-Level Folder Replication

- These steps should be performed on a separate client installation, not on a container, to isolate the new context properly.
- On the separate client install, remove the folder pointed at by $GENII_USER_DIR, which will start the process with a completely clean slate. This is shown in the steps below.
- This section assumes that the root container has already been deployed, and that a mirror container (aka root replica) has been installed, is running, but is not yet configured.
- The user executing the following commands requires administrator permissions via an admin.pfx keystore login. Note that if the admin certificates for the root and replica containers are distinct, then one should login with the keystore file for both the root and the replica container. Only the applicable keystore logins for the containers involved should be performed; do not login as an XSEDE user or other grid user first. For example:

```
grid logout --all
grid keystoreLogin local:$HOME/admin.pfx
```

Steps for Replicating Top-Level Grid Folders

This example sets up replication on the top-level grid folders within the XSEDE namespace. Note that this example uses the official host names for XSEDE hosts (e.g. gffs-2.xsede.org) and the default port (18443). These may need to vary based on your actual setup:

```
# GENII_INSTALL_DIR and GENII_USER_DIR are already established.
# This action is being taken on an isolated client install that points at the new grid;
# do not run this on the root or root replica container!
# Clean out the state directory before hand.
rm -rf $GENII_USER_DIR
# Login as the administrative keystore; repeat for all applicable keys.
grid keystoreLogin local:$HOME/admin.pfx
# Run the replication script; replace hostname and port as appropriate for replica host.
bash $XSEDE_TEST_ROOT/tools/xsede_admin/top_level_replication_setup.sh \
gffs-2.xsede.org 18443
# If no errors occurred, the new replication-aware context file is stored in:
# $HOME/replicated-context.xml
```

Note: allow the containers 5-10 minutes to finish replicating before shutting any of them down.

After replication has finished (and all containers seem to be in a quiescent state), it is important to backup both the root and the mirror container data (see section G.2.3 for backup instructions).

The replicated-context.xml file created by the above steps needs to be made available to grid users within an installation package. It is especially important to use this installation package for all future container installs. Submit the file to UVa Developers (xcghelp@cs.virginia.edu) for binding into an updated version of the grid's installer program. Installations of the GFFS that are created using the new installer will automatically see the replicated version of the root.
Testing Basic Replication

It is important to validate the grid’s replication behavior before attempting to use any replicated resources. The new replication configuration can be tested with the following steps:

- On the host of the root container, stop the root container process:

```
bash $XSEDE_TEST_ROOT/library/zap_genesis_javas.sh
```

- On a different host than the root, use the grid client and log out of all identities (the remainder of the steps also use this client host):

```
grid logout --all
```

- List the root folder in RNS (/). If this does not show the top-level replicated folders, then something is wrong with the replication configuration:

```
grid ls /
```

- If the above test succeeds, try a few other publically visible folders:

```
grid ls /users
grid ls /groups
```

Neither of the above commands should report any errors.

If the commands above work as described, then basic RNS replication is working properly. This is assured by having shut down the root container; the only source of RNS records that is still active in the grid is the mirror container.

G.2.5.6. Replicating User (STS) Entries

The replicated STS feature is used similarly to the replicated RNS & ByteIO feature. Suppose Joe represents a Kerberos or X509-Certificate STS resource created for a user Joe. In addition, assume that Joe has access to a group Group1 (so Group1 should be a sub-directory under Joe in the global namespace). Suppose Joe and Group1 reside in arbitrary containers and we want to replicate them. Then the sequence of steps for replication should be as follows.

1. Associate a resolver with the two resources. This step is not required if the users folder already has a resolver established:

```
grid resolver -p -r /users/xsede.org/Joe /resources/xsede.org/containers/{Container-Hosting-Resolver}
```

2. Replicate the STS resource. This can be done more than once to create multiple replicas:

```
grid replicate -p /users/xsede.org/Joe /resources/xsede.org/containers/{Container-Hosting-Replica}
```

If we want only Joe to be replicated but not Group1 then we drop the -r flag (indicating “recursion”) from the resolver command.
Note that the container hosting the resolver and the container hosting replicas are different in the above example, but they do not have to be different containers. However, neither of them should be the primary container where Joe or Group1 are stored, as that defeats the goal of replication.

To replicate the entire /users/xsede.org users hierarchy, use similar steps:

1. Associate a resolver with the users hierarchy. Skip this step if a resolver already exists on the users hierarchy:

   ```
   grid resolver -p -r /users/xsede.org \ 
   /resources/xsede.org/containers/{Container-Hosting-Resolver}
   ```

2. Replicate the STS resource:

   ```
   grid replicate -p /users/xsede.org \ 
   /resources/xsede.org/containers/{Container-Hosting-Replica}
   ```

G.2.5.7. Serving User and Group Identities from a Replicated STS Container

Initially all user and group identities will be stored on the Root container. The authentication processing for the grid can be migrated to a different container, possibly one reserved for managing STS resources such as users. The following sections describe how to accomplish the move in the context of the XSEDE namespace, where there are two STS containers (a primary and a secondary which replicates the primary).

**Prerequisites for Migrating to an STS Container**

- Ensure the root container's top levels are already replicated (see section G.2.5.5 if that has not already been done).
- The steps for migrating the STS should be executed on an administrative client host that has been installed with the replication-aware installer (produced in section G.2.5.5).
- The primary and secondary STS containers must be installed before executing the migration process (section D.4 or D.5).
- The user executing the following commands requires administrator permissions via an admin.pfx keystore login. One should login with the admin keystore for the root, the root replica, and the STS containers. Only the applicable keystore logins for the containers involved should be performed; do not login as an XSEDE user or other grid user first.

**Steps for Migrating to a New Primary STS Container**

This section brings up both STS containers and configures them before any replication is added:

- Run the script below with the host name and port number for the two STS servers. In this example, we use the official hostnames. Test systems should use the appropriate actual hostnames instead.

```
# variables GENII_INSTALL_DIR, GENII_USER_DIR and XSEDE_TEST_ROOT have # already been established.
# This action is being taken on an isolated client install that points at the new
```
grid.

# Run the STS migration script. The hostnames below are based on the official
# XSEDE hosts and should be modified for a test system.
bash $XSEDE_TEST_ROOT/tools/xsede_admin/migrate_to_sts_containers.sh \
  sts-1.xsede.org 18443 sts-2.xsede.org 18443

# If no errors occurred, then the two STS containers are now handling any new
# user account creations and will authenticate users of these new accounts.

Add an XSEDE User and Test Replication

The XSEDE Tool suite provides scripts for adding users to the namespace and for adding the users to groups. Add an XSEDE MyProxy/Kerberos for testing as follows (this step requires being logged in as an account that can manage other XSEDE accounts, such as by using the administrative keystores for the grid):

```
# create the user.  
grid script \  
  local:$XSEDE_TEST_ROOT/tools/xsede_admin/create-xsede-user.xml \  
  {myPortalID};  
grid script \  
  local:$XSEDE_TEST_ROOT/library/link-user-to-group.xml \  
  /users/xsede.org {myPortalID} /groups/xsede.org gffs-users
```

The user specified in {myPortalID} needs to be an XSEDE user ID that can be logged into by the tester. This account must be able to log in to the MyProxy and Kerberos servers at XSEDE.

Before testing replication below, ensure that the user account is working for login:

```
# relinquish administrative login:  
grid logout --all  
# try to login as the portal user:  
grid xsedeLogin --username={myPortalID}  
# show the result of logging in:  
grid whoami
```

The “whoami” command should print out the actual XSEDE user name that was configured above, and should also show group membership in the “gffs-users” group.

Allow the containers a couple of minutes to finish replicating before shutting any of them down. Once the user and groups have been replicated, the soundness of the replication configuration can be tested with the following steps:

- On the host of the root container, stop the root container process:

```
bash $XSEDE_TEST_ROOT/library/zap_genesis_javas.sh
```

- Run the above command on the primary STS container also, to stop that container.
- On an administrative client install with replication enabled (i.e., not on the primary containers), use the grid client and log out of all identities, then log in to the new user again:

```
grid logout --all  
grid xsedeLogin --username={myPortalID}  
grid whoami
```
If the login attempt above works and whoami still shows the details of the appropriate user and groups, then the STS replication configuration is almost certainly correct. Having shut down the primary STS container, the only source of authentication active in the grid is the secondary STS container. Similarly, all RNS requests must be served by the mirror container, since the root RNS container is down.

There are four cases that completely test the failover scenarios. The above test is listed in the table as test 1. To ensure that replication has been configured correctly, it is advisable to test the remaining three cases also:

1. Root down & sts-1 down
2. Mirror down & sts-1 down
3. Root down & sts-2 down
4. Mirror down & sts-1 down

If these steps are successful, then the new primary and secondary STS containers are now responsible for authentication and authorization services for the grid. Any new STS identities will be created on the STS container rather than on the root container. Even if the root container is unavailable, users will still be able to log in to their grid accounts as long as one root replica and one STS container is still available. (Any other required login services, such as MyProxy or Kerberos, must also still be available.)

G.3. RNS & ByteIO Caching

The Genesis II software supports client-side caching of GFFS resources to reduce the impact on the containers that actually store the files. By enabling this feature on a container, an administrator can allow users to cache files on their own hosts rather than always accessing the files on the container. This actually benefits both the administrator and the user, because the administrator will see fewer remote procedure calls requesting data from their containers and the users will see faster access times for the resources they are using frequently.

To enable the subscription based caching feature, it is necessary to add a permission on the port-type that implements the cache service:

```
grid chmod \\
{containerPath}/Services/EnhancedNotificationBrokerFactoryPortType \\
+x --everyone
```

After the port type is enabled, a grid command's most frequently used files will automatically be cached in memory. If the container's version of the file changes, the client is notified via its subscription to the cached data item, and the cached copy is automatically updated.

G.4. Grid Accounting

The Genesis II codebase offers some accounting features to track usage of the grid queue and the number of processed jobs. This section describes how to set up the accounting features and how to create a web site for viewing the accounting data.
This information is intended for local grid administrators (such as the XCG3 admins) and is not currently in use by the XSEDE project.

### G.4.1. Accounting Prerequisites

- Create a central database, we use MySQL (installed and maintained by CS department)
- Create the following tables in the Database table xcgaccountingrecords, table xcgbescontainers, table xcgcredentials, table xcgareccredmap and table xcgcommandlines (Information on the table columns is found in Appendix A)
- Link the database into grid name space
- Two usernames to access this central DB, one has write privileges and the other has read-only privileges. Username with write privileges will be used to run the accounting tool and username with read-only privileges will be used to get web GUI statistics.
- Web server Eg. webserver provided by CS department to be able to talk to Central database

There are several pieces to collecting and processing job accounting data for the XCG.

### G.4.2. Background

Raw job accounting data is kept on the home container of the BES that runs the job. It will stay there forever, unless someone signals that the container can delete accounting information up to a specified accounting record ID (we call this “committing” the records). We use a grid command line tool named "accounting" to collect accounting records, put the collected data into a database (currently this is the department’s MySQL database), and to commit all records collected on the container so that the container can delete them from its local database. In order to support our on-demand online accounting graphs, the raw accounting data collected from the containers must be processed into a format that the graphing pages can use.

So, overall, the collection process has 2 parts:

1. collect raw data from containers and store in raw accounting database tables.
2. process the data to update tables supporting accounting graphs.

### G.4.3. Accounting Database

The raw accounting data is placed into 5 related tables by the accounting collection tool. In order to make the data easier to process for usage graphs, a stored procedure named procPopulateXCGJobInfo crunches all of the data in the 5 raw accounting data tables and stores them in 2 derived “tmp” tables for use by the accounting graph php pages on the web site.

The vcgr database currently contains 15 tables, only about half related to the new way of doing accounting.

<table>
<thead>
<tr>
<th>Accounting related tables set by accounting collection tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>xcgaccountingrecords</td>
</tr>
</tbody>
</table>
xcgbescontainers Each row holds information about a BES container that has had accounting data collected for it. The key for the BES record in this table is used to match accounting records in the xcgaccountingrecords table to the BES container it ran on. Records in this table are matched during the accounting record collection process based on the BES’s EPI, so re-rolling a BES will cause a new BES entry to appear in this table. The besmachinename field in this table is populated by the machine’s IP address when it is first created and this is used by our accounting graphs as the name of the BES. However, the besmachinename field can be manually updated to put in a more human friendly name and to tie together records from two BES instances that have served the same purpose. This is something I do periodically to make the usage graphs more readable and informative.

xcgcredentials Contains a list of every individual credential used by any job. Multiple jobs using the same credential will share an entry in this table. Since the relationship between jobs (xcgaccountingrecords) and credentials (xcgcredentials) is many-to-many, the xcgareccredmap table provides the relationship mapping between them. The credentialtype field is set to NULL for new entries, but can be set to values “Client”, “Service”, “User” or “Group” manually. I occasionally manually edit this field to set the proper designation for new entries.

xcgareccredmap Associative table between xcgaccountingrecords and xcgcredentials.

xcgcommandlines Contains each portion of a job’s commandline – one entry for argument (including the executable argument). The accounting tool did not work properly at first and only recorded the last argument for jobs. This was fixed sometime after initial rollout of the new accounting infrastructure.

**G.4.4. Denormalized accounting data for usage graphs**

In order to easily support reasonably fast creation of a wide range of usage graphs, we use a stored procedure to create two tables to store pre-processed denormalized information about jobs.

tmpXCGJobInfo This table contains 1 row per job with pretty much everything in it that we can run a report against. This includes our best guess the job’s “owner” in human friendly terms, the bes container’s name, various run time information, and information about the day, week, month, and year of execution.

tmpXCGUsers This table is an intermediate table used by the stored procedure that creates the tmpXCGJobInfo table. It really can be deleted as soon as the stored procedure finishes – not sure why it isn’t...

**G.4.5. The denormalization process**

The denormalization process deletes and re-creates the tmpXCGJobInfo and tmpXCGUsers tables from data in the raw accounting tables. Denormalization is done by running the procPopulateXCGJobInfo stored procedure.
N.B. Besides denormalizing the data so that there is single row per job, it also tries to figure out which user “owns” the job and stores it’s best guess in the username field of the tmpXCGJobInfo table. This field is used by the usage graphs to display/filter who ran a job.

The algorithm for doing so is imperfect, but must be understood to properly understand the usage graph behavior. A job’s owner is determined as the credential with the lowest cid (credential id) associated with the job that is designated as a “User” (it may also require that the credential has the string X509AuthnPortType in it). It then assumes that the credential is in the format of those we mint for ordinary XCg users and extracts the owner’s name from the CN field of the credential.

This process will only work if these conditions are met:

- The job has at least one credential minted as a normal user by XCG in the usual format.
- The credential has been manually marked as a “User” credential in the xcgcredentials table.

Users who use a different type of credential (e.g. username/password or other outside credential), were run only by admin (different cred format) or have not had their entry in the xcgcredentials table manually updated to type “User” will be labeled as owned by “unknown”.

For denormalization, there are two main procedures: 1) collection of raw data and 2) processing the data for the online graphs.

1) The grid tool “accounting” is used to collect raw accounting data and store it in the CS department database. The tool takes several options/arguments:

- --collect: tells the tools to do the actual data collection. NOTE: unless the “--no-commit” flag is specified, the --collect flag will commit all accounting records successfully collected from the target container(s).
- --no-commit: optional flag to stop tool from committing (i.e. erasing) records from the target container(s).
- --recursive: allows the user to specify a directory of containers and the tools will recursively go through all entries in directory and collect from each one.
- --max-threads=<# threads>: allows the collection to be multi-threaded
- <source container> | <source container directory>: which container (directory) to collect.
- <target database>: connect information for database that will store collected data.

Typical use (as admin – other users will not have permission to collect or commit accounting data from most/all containers):

```
accounting --collect --recursive /containers /accounting/CS-MySQL
```

The tool uses grid containers, not BES containers as targets. Even though the accounting records do identify which actual BES container the job was associated with, the tools collect all of the data for all BESes it contains at once.
1. We can use the directory /containers recursively because we try to maintain /containers such that it has all of our useful containers in it and no other extraneous entries. This helps simplify the process significantly.

2. We use the RNS path /accounting/CS-MySQL as the target database. Mark set up this RNS entry with the proper connection information for the vcgr database on the department server to help the collection process. The tool can handle extracting the connection info from the EPR contained by the RNS entry.

3. There will be a prompt for the password for the vcgr_mmm2a account on the CS department database server.

4. The tool will collect data from each container in turn. Note exceptions – there are sometimes entries left for containers that are no longer present in the /containers directory.

2) To process the data for the online graphs:

- Log into the department's PHP MySQL web front end. Use a browser and go to . At the login screen, use the vcgr_mmm2a account and password to login.
- Go to the vcgr database. Just click on the vcgr database entry on the left side of the screen
- Get an SQL window. Click the SQL tab and a text area will appear where SQL statements can be entered.
- Run the stored procedure procPopulateXCGJobInfo (procedure definition found in the section below called Database Table Structure for Accounting). In the SQL text area type "call procPopulateXCGJobInfo" and click the Go button. It will run for several minutes and then the screen will go blank when it is done. The procPopulateXCGJobInfo stored procedure will erase the old tmpXCHJobInfo and tmpXCGUsers tables and will repopulate them after crunching whatever data is in the raw accounting tables. GUI statistics can be looked up here http://vcgr.cs.virginia.edu/XCG/2.0/stats/usage_graphs/

G.4.6. Linking the Accounting Database Into the Grid

To link a Database into Grid name space, use the following command:

```
```

G.4.7. Migrating Accounting Info to a New Grid

For linking the XCG2 accounting info to new XCG, all we need to do is create a DB EPR in the new XCG grid name space. This is done using:

```
```

This will create a new EPR which can be used with 'accounting' tool like this:

```
accounting --collect --recursive /<grid-path>/<containers-directory> /accounting/XCG3-CS-MySQL
```
This will use the same database tables in the existing database, and all the information will be preserved from the previous grid.

**G.4.8. Usage Graph Web Site**

The statistics gathered from the grid can be displayed in a web site that supports queries based on operating system and time ranges. An example implementation is provided within the XSEDE tests repository, which is available at:

```plaintext
```

These files are an example only, and would need to be configured appropriately for the site’s Ploticus installation location, the usage graph site’s location on the web server, and the login information for the statistics database.

This implementation uses PHP and the ploticus application ([http://ploticus.sourceforge.net/doc/welcome.html](http://ploticus.sourceforge.net/doc/welcome.html)) to build graphs per a user request. The figure below shows the running site, with a form for requesting accounting info using the available filters. Given a query with a given date range and a daily report, the output might resemble the Usage Graph in the next figure.

![Figure 46. User request form](image)
**Figure 47. Example of daily usage graph**

### G.4.9. Database Table Structure for Accounting

#### G.4.9.1. USERS Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>name</td>
<td>varchar(128)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>varchar(128)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>department</td>
<td>varchar(128)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>organization</td>
<td>varchar(128)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>varchar(64)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>email</td>
<td>varchar(128)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>varchar(64)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>varchar(128)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>idppath</td>
<td>varchar(256)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>homedir</td>
<td>varchar(256)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

#### G.4.9.2. XCGCREDENTIALS Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>cid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>credential</td>
<td>blob</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>credentialtype</td>
<td>varchar(16)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>credentialdesc</td>
<td>varchar(512)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>credentialhash</td>
<td>int(11)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>timeadded</td>
<td>timestamp</td>
<td>NO</td>
<td></td>
<td>CURRENT_TIMESTAMP</td>
<td></td>
</tr>
</tbody>
</table>

#### G.4.9.3. GENIIJOBLOG Table
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>eventtype</td>
<td>varchar(64)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>identities</td>
<td>varchar(1024)</td>
<td>NO</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>eventtime</td>
<td>datetime</td>
<td>NO</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>epi</td>
<td>varchar(256)</td>
<td>YES</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobid</td>
<td>varchar(256)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>processTime</td>
<td>bigint(20)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>hostname</td>
<td>varchar(256)</td>
<td>NO</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

G.4.9.4.  GROUPS Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>name</td>
<td>varchar(128)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>varchar(512)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>path</td>
<td>varchar(256)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

G.4.9.5.  MEMBERSHIP Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>uid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>gid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

G.4.9.6.  tmpXCGJobInfo Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>arid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>besaccountingrecordid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>besid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>exitcode</td>
<td>int(11)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobuserhrs</td>
<td>decimal(23,4)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobkernelhrs</td>
<td>decimal(23,4)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobwallclockhrs</td>
<td>decimal(23,4)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>maxrssbytes</td>
<td>bigint(20)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>recordtimestamp</td>
<td>timestamp</td>
<td>NO</td>
<td>MUL</td>
<td>CURRENT_TIMESTAMP</td>
<td>on update CURRENT_TIMESTAMP</td>
</tr>
<tr>
<td>jobyear</td>
<td>smallint(6)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobmonth</td>
<td>smallint(6)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobmonthname</td>
<td>char(10)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobday</td>
<td>smallint(6)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobdayofyear</td>
<td>smallint(6)</td>
<td>NO</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobmondaydate</td>
<td>date</td>
<td>NO</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>besmachinename</td>
<td>varchar(256)</td>
<td>YES</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>
### tmpXCGUsers Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>cid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>varchar(256)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

### XCGACCOUNTINGRECORDS Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>arid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>besaccountingrecordid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>besid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>exitcode</td>
<td>int(11)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>usertimemicrosecs</td>
<td>bigint(20)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>kerneltimemicrosecs</td>
<td>bigint(20)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>wallclocktimemicrosecs</td>
<td>bigint(20)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>maxrssbytes</td>
<td>bigint(20)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>recordtimestamp</td>
<td>timestamp</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

### XCGARECCREDMAP Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>mid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>cid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>MUL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>arid</td>
<td>bigint(20)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

### XCBESCONTAINERS Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>besid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>besepi</td>
<td>varchar(256)</td>
<td>NO</td>
<td>UNI</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Null</td>
<td>Key</td>
<td>Default</td>
<td>Extra</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>------</td>
<td>------</td>
<td>---------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>bevmachinename</td>
<td>varchar(256)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>arch</td>
<td>varchar(64)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>os</td>
<td>varchar(64)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>timeadded</td>
<td>timestamp</td>
<td>NO</td>
<td>NULL</td>
<td>CURRENT_TIMESTAMP</td>
<td></td>
</tr>
</tbody>
</table>

**G.4.9.11. XCGCOMMANDLINES Table**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>clid</td>
<td>bigint(20)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>arid</td>
<td>bigint(20)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>elementindex</td>
<td>int(11)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>element</td>
<td>varchar(512)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>
G.4.10. Creating the Accounting Database

This SQL source code will create the temporary job tables after the user has run the accounting tool. The database must already exist and be structured as above.

```
ProcPopulateXCGJobInfo routine_definition
BEGIN

DROP TABLE IF EXISTS tmpXCGUsers;
DROP TABLE IF EXISTS tmpXCGJobOwnerIds;
DROP TABLE IF EXISTS tmpXCGJobOwners;
DROP TABLE IF EXISTS tmpXCGJobInfo;

CREATE TABLE `tmpXCGUsers` (  `cid` bigint(20) NOT NULL,  `username` varchar(256) NOT NULL,  PRIMARY KEY (`cid`)) DEFAULT CHARSET=latin1;
DELETE FROM tmpXCGUsers;
INSERT INTO tmpXCGUsers  (`cid`, `username`)  SELECT  `xcgcredentials`.`cid`, substring_index(substr(`xcgcredentials`.`credentialdesc`,42),',',1) FROM `xcgcredentials` WHERE ((`xcgcredentials`.`credentialtype` = 'User') AND (`xcgcredentials`.`credentialdesc` like '%$X509AuthnPortType%'));

CREATE TABLE `tmpXCGJobOwnerIds` (  `arid` bigint(20) NOT NULL,  `ownercid` bigint(20) NOT NULL,  PRIMARY KEY (`arid`)) DEFAULT CHARSET=latin1;
DELETE FROM tmpXCGJobOwnerIds;
CREATE INDEX `tmpXCGJobOwnerIds_ownercid_Idx` ON `tmpXCGJobOwnerIds` (`ownercid`);
INSERT INTO tmpXCGJobOwnerIds  (`arid`, `ownercid`)  SELECT  `ar`.`arid`, min(`cred`.`cid`) FROM `xcgaccountingrecords` `ar`, `xcgareccredmap` `map`  LEFT JOIN `xcgcredentials` `cred` ON (((`map`.`cid` = `cred`.`cid`)) AND (`cred`.`credentialtype` = 'User') AND (not((`cred`.`credentialdesc` like '%$Admin%'))))
```
WHERE (`ar`.`arid` = `map`.`arid`)
GROUP BY
  `ar`.`arid`;

CREATE TABLE `tmpXCGJobOwners` (
  `arid` bigint(20) NOT NULL,
  `ownercid` bigint(20) NOT NULL,
  `username` varchar(256) NULL DEFAULT NULL,
) PRIMARY KEY (`arid`)
) DEFAULT CHARSET=latin1;

DELETE FROM tmpXCGJobOwners;

CREATE INDEX `tmpXCGJobOwners_ownercid_Idx`
ON `tmpXCGJobOwners` (`ownercid`);

INSERT INTO tmpXCGJobOwners
(
  `arid`,
  `ownercid`,
  `username`
) SELECT
  `jo`.`arid`,
  `jo`.`ownercid`,
  `users`.`username`
FROM
  `tmpXCGJobOwnerIds` `jo`,
  `tmpXCGUsers` `users`
WHERE
  (`jo`.`ownercid` = `users`.`cid`);

DROP TABLE IF EXISTS tmpXCGJobOwnerIds;

DROP TABLE IF EXISTS tmpXCGJobInfo;

CREATE TABLE `tmpXCGJobInfo` (
  `arid` bigint(20) NOT NULL,
  `besaccountingrecordid` bigint(20) NOT NULL,
  `besid` bigint(20) NOT NULL,
  `exitcode` int(11) NOT NULL,
  `jobuserhrs` decimal(23,4) NOT NULL,
  `jobkernelhrs` decimal(23,4) NOT NULL,
  `jobwallclockhrs` decimal(23,4) NOT NULL,
  `maxrssbytes` bigint(20) DEFAULT NULL,
  `recordtimestamp` timestamp NOT NULL,
  `jobyear` smallint NOT NULL,
  `jobmonth` smallint NOT NULL,
  `jobmondaydate` date NOT NULL,
  `besmachinename` varchar(256) NULL DEFAULT NULL,
  `arch` varchar(64) NULL DEFAULT NULL,
  `os` varchar(64) NULL DEFAULT NULL,
  `ownercid` bigint(20) NOT NULL,
  `username` varchar(256) NULL DEFAULT NULL,
  `linuxuserhrs` decimal(23,4) NOT NULL,
  `linuxkernelhrs` decimal(23,4) NOT NULL,
  `linuxwallclockhrs` decimal(23,4) NOT NULL,
  `windowsuserhrs` decimal(23,4) NOT NULL,
CREATE TABLE `tmpXCGJobInfo` (  
`arid` tinyint(3) unsigned NOT NULL,  
`besaccountingrecordid` smallint(5) unsigned NOT NULL,  
`besid` smallint(5) unsigned NOT NULL,  
`besmachinename` varchar(255) NOT NULL,  
`exitcode` smallint(5) unsigned NOT NULL,  
`jobuserhrs` decimal(23,4) NOT NULL,  
`jobkernelhrs` decimal(23,4) NOT NULL,  
`jobwallclockhrs` decimal(23,4) NOT NULL,  
`maxrssbytes` bigint(20) NOT NULL,  
`recordtimestamp` timestamp NOT NULL,  
`jobyear` tinyint(4) unsigned NOT NULL,  
`jobmonth` tinyint(4) unsigned NOT NULL,  
`jobmonthname` varchar(3),  
`jobday` tinyint(4) unsigned NOT NULL,  
`jobdayofyear` tinyint(4) unsigned NOT NULL,  
`jobmondaydate` date NOT NULL,  
PRIMARY KEY (`arid`)  
) DEFAULT CHARSET=latin1;

CREATE INDEX `tmpXCGJobInfo_besmachinename_Idx` ON `tmpXCGJobInfo` (`besmachinename`);
CREATE INDEX `tmpXCGJobInfo_username_Idx` ON `tmpXCGJobInfo` (`username`);
CREATE INDEX `tmpXCGJobInfo_recordtimestamp_Idx` ON `tmpXCGJobInfo` (`recordtimestamp`);
CREATE INDEX `tmpXCGJobInfo_jobmondaydate_Idx` ON `tmpXCGJobInfo` (`jobmondaydate`);

INSERT INTO tmpXCGJobInfo  
VALUES  
(`arid`,  
`besaccountingrecordid`,  
`besid`,  
`exitcode`,  
`jobuserhrs`,  
`jobkernelhrs`,  
`jobwallclockhrs`,  
`maxrssbytes`,  
`recordtimestamp`,  
`jobyear`,  
`jobmonth`,  
`jobmonthname`,  
`jobday`,  
`jobdayofyear`,  
`jobmondaydate`,  
`besmachinename`,  
`arch`,  
`os`,  
`ownercid`,  
`username`,  
`linuxuserhrs`,  
`linuxkernelhrs`,  
`linuxwallclockhrs`,  
`windowsuserhrs`,  
`windowskernelhrs`,  
`windowswallclockhrs`,  
`macosuserhrs`,  
`macoskernelhrs`,  
`macoswallclockhrs`  
);
dayofmonth(`ar`. `recordtimestamp`),
dayofyear(`ar`. `recordtimestamp`),
cast((`ar`. `recordtimestamp` - interval weekday(`ar`. `recordtimestamp`) day) as date),
`bes`. `besmachinename`,
`bes`. `arch`,
`bes`. `os`,
`owners`. `ownercid`,
`owners`. `username`,
``IF(bes.os = 'LINUX', (`ar`. `usertimemicrosecs` / 3600000000), 0),
IF(bes.os = 'LINUX', (`ar`. `kerneltimemicrosecs` / 3600000000), 0),
IF(bes.os = 'LINUX', (`ar`. `wallclocktimemicrosecs` / 3600000000), 0),
IF(bes.os = 'Windows_XP', (`ar`. `usertimemicrosecs` / 3600000000), 0),
IF(bes.os = 'Windows_XP', (`ar`. `kerneltimemicrosecs` / 3600000000), 0),
IF(bes.os = 'Windows_XP', (`ar`. `wallclocktimemicrosecs` / 3600000000), 0),
IF(bes.os = 'MACOS', (`ar`. `usertimemicrosecs` / 3600000000), 0),
IF(bes.os = 'MACOS', (`ar`. `kerneltimemicrosecs` / 3600000000), 0),
IF(bes.os = 'MACOS', (`ar`. `wallclocktimemicrosecs` / 3600000000), 0)
FROM
(`xcgaccountingrecords`.`ar`)
LEFT JOIN `xcgbescontainers`.`bes` ON(`bes`. `besid` = `ar`. `besid`))
LEFT JOIN `tmpXCGJobOwners` `owners` on(`owners`. `arid` = `ar`. `arid`));

DROP TABLE IF EXISTS tmpXCGJobOwners;
END

G.5. Grid Inter-Operation

The Genesis II software fully supports grid federation, where resources can be shared between multiple grids. This enables researchers to connect to a low-latency grid that is geographically convenient while still sharing data and BES resources with researchers on other grids. The XSEDE namespace provides a convenient method to achieve “grid isomorphism”, where the locations of other grids’ resources can be found at the identical location in RNS regardless of which grid one is connected to.

For example, the XSEDE Operations Grid is a Genesis II GFFS grid that is maintained by the XSEDE project. The Cross-Campus Grid (XCG) is also a Genesis II GFFS grid, but it is maintained by the University of Virginia. Despite these grids being within very different administrative domains, the users on XCG grid can log into their accounts and access their home directories on the XSEDE grid. This is accomplished by linking parts of the XSEDE grid into the XCG namespace structure.

The interconnections from XCG to XSEDE were created by the XCG administrator. Each “foreign” grid is given a well-defined location in the /mount directory where the remote grid is linked. For the XSEDE grid, the top-level (/) of the grid has been linked into /mount/xsede.org. Listing the contents of that folder shows the root of XSEDE’s grid; note that this command is executed on an XCG grid client, not an XSEDE grid client:
```
This is the same list of folders one sees if one is connected to the XSEDE grid and lists the top-level of RNS, but in this case, it is visible via the link in the XCG grid.

To gain fully isomorphic grid folders, one makes links for each of the major items in the foreign grid under the appropriate folders in one's own grid. For example, XCG has a folder for its local users called /users/xcg.virginia.edu, but it also has a folder called /users/xsede.org for the remote users that live in the XSEDE grid. From the XSEDE grid's perspective, it would have a link for /users/xcg.virginia.edu that connects to the XCG grid. Using the foreign path, one can authenticate against the XSEDE grid’s STS for a user even though one is connected to the XCG grid. This provides for fine-grained access control across the multiple grids, and ensures that the user can acquire whatever credentials are needed to use the remote grid's resources.

Similarly, the home folders of the XSEDE grid are available in the XCG grid, as /home/xsede.org. This allows a person who is connected to the XCG to access their remote files and directories that reside in their XSEDE grid home directory. Using this capability, researchers can submit jobs that stage files in and out from any grid that they have access to, and can share their data with other researchers on any of these interconnected grids.

G.5.1. Connecting a Foreign Grid

Making a non-local grid available on one's home grid is achieved by the following steps. In the actions below, we are using the concrete example of linking the XSEDE grid into the XCG grid, but any two grids could be linked in this manner. To achieve grid isomorphism, it is important to pick an appropriate name for the foreign grid and for that name to be used consistently across all federated grids. Otherwise, a path on grid X may be named differently on grid Y, which will lead to many problems with creating grid jobs that run seamlessly on either of the two grids (since expected stage-out paths may not be there without consistent naming).

1. Acquire the root EPR of the foreign grid. This can be accomplished when actually connected to the foreign grid, using its installer or grid deployment. The step below is assumed to be running from the XSEDE grid client:

```
grid ls -ed / | tail -n +2 | sed -e 's/^\s//g' >xsede_context.epr
```

2. The above creates a context file that can be used to link the foreign grid. This step must be performed using the grid client for one's local grid (which is about to be augmented with a link to the foreign grid):

```
grid ln --epr-file=local:xsede_context.epr /mount/xsede.org
```

3. Test the new link by listing its contents. It should show the top-level folders of the foreign grid:

```
grid ls /mount/xsede.org
```
4. If the prior step is unsuccessful, then it is possible the local grid does not trust the remote grid. To establish trust between the grids, the CA certificate of the remote grid’s TLS certificate(s) should be added to the local grid’s trust store. Below, it is assumed that “current_grid” is the specific deployment in use in the local grid and that “remoteCA.cer” is a CA certificate that issued the remote grid’s TLS certificates. Adding more than one certificate is fine, and the certificates can either be in DER or PEM format:

```
cp $HOME/remoteCA.cer \ $GENII_INSTALL_DIR/deployments/current_grid/security/trusted-certificates
```

5. After the remote grid can be listed successfully at its location in mount, the remote hierarchies can be added to the local grid. These should continue the naming convention established for the mount, so that isomorphism is maintained between the grids:

```
grid ln /mount/xsede.org/users /users/xsede.org
grid ln /mount/xsede.org/home /home/xsede.org
grid ln /mount/xsede.org/resources /resources/xsede.org
grid ln /mount/xsede.org/groups /groups/xsede.org
```

6. Listing each of the new folders should show the appropriate type of resources. With a successful top-level mount, this step should always succeed.

This procedure can be repeated as needed to federate other grids alongside one’s own grid. A grid structured isomorphically is a joy for researchers to use, since all paths are precisely arranged and named in a way that makes their true home clear. Jobs can be executed on any BES or queue that the researcher has access to, and the staging output can be delivered on any of the connected grids that the researcher desires. In addition, one’s colleagues on other grids can provide access to their research data in a regular and easy to understand structure, even when the grids are in completely different countries and administrative domains.
H. XSEDE Development with Genesis II

This section focuses on building the XSEDE GFFS components from the Genesis II source code. Support for basic EMS (via the fork/exec BES) is included in the Genesis II source; building the Unicore EMS is not addressed here. This section may be found useful by hard-core users who want to run their container from source and by developers who want to fix bugs or add features to the Genesis II software.

Note that at this time, development of the Genesis II software is only supported in Java. The Genesis II components can be controlled by a variety of methods (grid client, Xscript, client-ui), but they can only be extended by writing new Java classes or modifying existing ones.

H.1. Installing Java

The configuration of Java can be quite confusing to a neophyte, but going very deeply into that process is beyond the scope of this document. This section does explain the basics of setting up Java for building and running Genesis II. It is expected that a Genesis II developer has prior training in Java, but normal users of Genesis II should not need Java proficiency.

We recommend using Oracle Java 7 for building and running the Genesis II codebase. Java 6 (aka 1.6) is no longer supported for Genesis II builds. The latest versions of Java can be downloaded at Oracle’s Java Downloads:

The JRE can be difficult to install on Centos, and this guide has been helpful in previous installations: http://wiki.centos.org/HowTos/JavaRuntimeEnvironment If you intend to recompile the Genesis II code base, then download a JDK (Java Development Kit) and not just the JRE (Java Runtime Engine). There is an installation guide at Oracle for the Java 7 JDK here:
http://docs.oracle.com/javase/7/docs/webnotes/install/index.html

Some of the Genesis II scripts rely on the JAVA_HOME variable being set. This should point to the top directory for the JDK being used to build Genesis II. For example, if the Java JDK is installed at /usr/lib/jvm/java-7-oracle, then the JAVA_HOME variable could be set with:

```
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
```

To determine what version of java is in the path, run:

```
java -version
```

If that does not show the appropriate version, the PATH variable may need to be modified. For example:

```
export PATH=$JAVA_HOME/bin:$PATH
```
Enabling Strong Encryption in the JRE

Building and running GFFS containers requires that full-strength JCE security is enabled for the Java Runtime Environment (JRE). Otherwise the JRE will not allow the certificates to be generated with the necessary key length. The unlimited strength JCE jars are available at: http://www.oracle.com/technetwork/java/javase/downloads/index.html

As an example, after downloading the unlimited JCE zip file for Oracle Java 7 on Linux, the security jars might be updated with these steps:

```
unzip UnlimitedJCEPolicyJDK7.zip
sudo cp UnlimitedJCEPolicy/*/jar /usr/lib/jvm/java-7-oracle/jre/lib/security
```

The location of the JVM can vary widely on different platforms, but these jar files generally belong in a subdirectory ending in ‘jre/lib/security’.

H.1.1. Centos Build Dependencies

On Centos 6.4 (or later) Linux, the following packages may be needed to build Genesis II.

- Ant and additional ant packages were installed with:

```
sudo yum install ant ant-contrib ant-nodeps ant-jsch ant-trax ant-junit
```

H.1.2. Ubuntu Build Dependencies

On Ubuntu Linux, the following packages may be needed to build Genesis II.

- Ant was installed with:

```
sudo apt-get install ant
```

H.2. Getting the Genesis II Source Code

Genesis II source is checked into a subversion revision control system. To retrieve the latest version, use the following:

Check out the Genesis II source code:

```
svn co svn://svn.xcg.virginia.edu:9002/GENREPO/GenesisII/trunk
```

Base the GENII_INSTALL_DIR on the new check-out:

```
export GENII_INSTALL_DIR=$HOME/trunk
# location depends on the check-out that occurred in the first step.
```

H.3. Building Genesis II from Source on the Command Line

Ensure that the GENII_INSTALL_DIR has been set to point at the source code location, that Java is installed with unlimited JCE encryption, and that JAVA_HOME is set (see section H.1).
To perform the main build of the Genesis II trunk, change to the source code location and run "ant -Dbuild.targetArch=32 build" (for 32 bit platforms) or “ant -Dbuild.targetArch=64 build” (for 64 bit platforms).

The follow example rebuilds the source code for 64 bit platforms:

```
cd $GENII_INSTALL_DIR
# Set the ANT options to increase the memory limits:
export ANT_OPTS=' -Xms512m -Xmx768m -XX:MaxPermSize=768m'
# clean generated files from any prior builds.
ant clean
# perform the build.
ant -Dbuild.targetArch=64 build
```

The ANT_OPTS above are required because the web-services build requires more memory than the default amount allocated by ant.

It is important to rebuild the source code on the target machine, rather than using a build from someone else, to ensure that any embedded script paths are regenerated properly.

After building the source code, one needs a grid to test against. If you have an existing grid and the necessary deployment information, then that is sufficient. But if you want to test on an isolated grid that is under your control, consult the XSEDE tests chapter I.2 on how to “How to Bootstrap a Miniature Test Grid” for details on setting up a local grid for testing.

**H.4. Developing Genesis II in Eclipse**

Eclipse is an integrated development environment for Java and other languages, and many developers prefer to manage their coding process with Eclipse. These instructions should assist an Eclipse developer to become comfortable with building and debugging the Genesis II codebase.

**H.4.1. Getting Eclipse**

Download the newest version of the Eclipse IDE for Java Developers from [http://www.eclipse.org/](http://www.eclipse.org/). Genesis II currently relies on features found in the “Kepler” version of Eclipse. As well as providing a software development environment, Eclipse can be used for debugging with dynamic code injection, call-hierarchy searching, and auto-formatting.

There is a plugin called Subclipse which integrates Eclipse with SVN with GUI features for diff'ing workspaces, files, etc.

The Genesis II team has had success using a Java profiler called “YourKit Profiler” which can be integrated with Eclipse.

When first running Eclipse, the user will be asked to select a workspace. Do not specify a path that contains spaces (this just generally makes life easier, although it may not be strictly necessary).

**H.4.2. Getting Subclipse**
Subclipse is a useful add-in for Eclipse that provides subversion repository support. To obtain Subclipse, go to [http://subclipse.tigris.org/](http://subclipse.tigris.org/). Click on "Download and Install". Follow the instructions to install the Subclipse plug-in in Eclipse. The best version of the SVN client to use with our SVN server is version 1.6.x.

If Eclipse fails to install Subclipse, then the user may need to install the "Mylyn" plug-in. The Mylyn update site is [http://download.eclipse.org/mylyn/releases/latest](http://download.eclipse.org/mylyn/releases/latest). With most versions of Eclipse, there is no need to worry about this.

If Eclipse complains about not finding JavaHL on Linux, then it may be that /usr/lib/jni needs to be added to the Java build path in Eclipse. This article has more information about this issue: [Failing to load JavaHL on Ubuntu](http://www.example.com/failing-to-load-javahl-on-ubuntu).

### H.4.3. Eclipse Package Explorer

For easier browsing of the Genesis II source code, setup the Package Explorer view with the following options:

Right click on the package explorer menu button (down arrow) and under Package Presentation select Hierarchical.

Right click on the package explorer menu button (down arrow) and select Filters. Add the "Libraries from external" filter.

#### H.4.3.1. Projects to Load in Eclipse

There is a main trunk project for Genesis II called Genesis II-trunk. Once you have downloaded the Genesis II project source code, you can load this using Eclipse's "Import Existing Projects into Workspace" choice. Browse to the folder where the trunk resides in the "Select root directory" field. Enable the option "Search for nested projects". Disable the option to "Copy projects into workspace". Select "Finish" to complete the project import. This should now show several projects in the package explorer.

Loading the project will cause Eclipse to build its representation of the Java classes. This will fail if an ant build has not been done before (see above section for building from the command line). Once an ant build has been done, select the "Project | Clean" menu item and clean all projects; this will cause Eclipse to rebuild the classes.

#### H.4.3.2. Setting the "derived" Type on Specific Folders

Eclipse will search for classes in any directory that is listed in a project. This sometimes is irksome, as it will find matches for X.class as well as X.java, but X.class is a compiled class output file and is not useful for reading or setting breakpoints. Luckily eclipse also provides an approach for forcing it to ignore file hierarchies. Eclipse ignores any folder that has the "derived" flag set on it. This can be applied to a directory by right-clicking on it and selecting "Properties". The resulting pop-up window will show a Resource tab for the folder, with a check-box for a derived attribute.
Note that each developer must set his own “derived” attributes on folders, since these attributes are not stored in the project file (they live in the user’s workbench).

It is recommended that any project’s generated file folders be marked as derived, which includes the following folders:

```
bin.ant (in every project)  
codegen (in gffs-webservices) 
genned-obj (in gffs-webservices)  
libraries (in Genesis II-trunk)
```

The “libraries” folder is not generated, but its contents are already provided by other projects.

After marking all of the folders that contain generated .class files as derived, future searches for classes in eclipse should only match .java files. This can be tested with the “open reference” command (Ctrl-Shift-r).

**H.4.4. Ant Builds**

To build Genesis II, we use Ant. The two Ant targets that are most often used: build and clean.

The “ant build” target performs the following activities:

1. Creates directories for generated sources
2. Normalizes our extended form of WSDL (GWSDL) into proper service WSDL
3. Runs Axis WSDL2Java stub generation on our service WSDL to create the Java stub classes used by client tools and the data-structure classes for representing operation parameters within both client and server code.
4. Copies the generated .wsdd files into the "deployments/default/services" directory, so that the Axis web application can find the Java classes that implement the port types
5. Compiles both the generated and pre-existing sources
6. Archives the bytecode class files into their respective utility, client, and container .jar files.
7. Creates shell scripts for starting the container and the command-line tools in the base Genesis II directory

To create these Ant targets, go to the Run menu and select External Tools. Create Ant Builds with the following specifications.

**H.4.4.1. Clean Target**

**Name**: Clean Build  
Under the Main tab: set the Buildfile. Browse Workspace and select build.xml  
Under the Target tab: select only clean

**H.4.4.2. Code Generator Target**

**Name**: Generate Code  
Under the Main tab: set the Buildfile. Browse Workspace and select build.xml
Under the Target tab: select only build
Under the JRE tab: Under VM Arguments add: -XX:MaxPermSize=512M

After running Build Codegen targets, refresh the project (F5) so that Eclipse will compile everything that must be compiled.

### H.4.5. User Directory for Genesis II Grid State

When running, the Genesis II container stores its database state and its RNS state in a directory called the state directory. Also, the command-line client tool(s) store their session state (transient login credentials, current working directory, etc.) in this directory. By default this directory is located at:

<table>
<thead>
<tr>
<th>Environment</th>
<th>State Directory Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux and Mac OS X</td>
<td>$HOME/.genesisII-2.0</td>
</tr>
<tr>
<td>Windows XP</td>
<td>C:/Documents and Settings/{username}/.genesisII-2.0</td>
</tr>
<tr>
<td>Windows Vista/7/8+</td>
<td>C:/Users/{username}/.genesisII-2.0</td>
</tr>
</tbody>
</table>

The user can change the location of the state directory by setting the GENII_USER_DIR environment variable before running the container or the client. Genesis II always respects this environment variable, regardless of whether Linux or Windows is being used, and regardless of whether Genesis II is run from Eclipse or from a shell.

In Windows, to set an environment variable, go to System Properties, open the Advanced tab, open the Environment Variables dialog, and create a new variable:

```
Name: GENII_USER_DIR
Value: <location of dir> (i.e. C:\genesis_state)
```

If you wish to wipe out a GFFS container and start from a totally clean state, then move or delete the GENII_USER_DIR contents and the container will regenerate its database with the default empty contents. This removes any prior configuration for the client and container!

### H.4.6. Run Configurations

To run Genesis II within Eclipse, two Run Configurations need to be configured. To create a new configuration, right-click on the project in the Package or Navigator view. In the context menu that appears, select "Run As". In the submenu that appears, select "Run Configurations..." In the resulting dialog window, select "Java Application" and click the "New launch configuration" button.

#### H.4.6.1. Creating a Profile for Running the Container

1. Set up a launch configuration with the following values:

```
Name: Container
In the Main tab, select the Project
In the Main tab, set the Main class to:
    edu.virginia.vcgr.genii.container.Container
In the Arguments tab, add the following VM arguments:
```
Note: The java.library.path should vary based on your intended platform, such as:

```
-Djava.library.path=${workspace_loc:GenesisII-trunk/jni-libs/lin32}
-Djava.library.path=${workspace_loc:GenesisII-trunk/jni-libs/win32}
-Djava.library.path=${workspace_loc:GenesisII-trunk/jni-libs/osx64}
```

2. In the Classpath tab, add the top-level folder for the gffs-structure and the gffs-webservices library projects to the classpath via the “Advanced” button and the “Add folders” option.

3. To change the default location of the state directory, create a new variable in the Environment tab called GENII_USER_DIR, and set the new location. Note that you can just inherit this from your shell environment if it is already set.

After this configuration is saved, it can be run either (a) from the Run Configurations dialog box, or (b) from the Debug Configurations dialog box, or (c) from the Run menu’s "Run As" or "Debug As" submenus, or (d) from the Run tool or the Debug tool in the toolbar.

**H.4.6.2. Creating a Profile for Running the Client**

This will create a launch configuration that starts the CLI tool called the grid shell. The grid shell can be used to run all of the sub-tools.

1. Set up a launch configuration with the following values:

   **Name: Client**
   
   **In the Main tab, select the Project**
   
   **In the Main table, set the Main class to:**
   
   ```
   edu.virginia.vcgr.genii.client.cmd.Driver
   ```
   
   **In the Arguments tab, add the following VM arguments:**
   
   ```
   -Dlog4j.configuration=build.container.log4j.properties
   -Djava.library.path=${workspace_loc:GenesisII-trunk/jni-libs/lin64}
   -Dedu.virginia.vcgr.genii.install-base-dir=${workspace_loc:GenesisII-trunk}
   -Djava.net.preferIPv4Stack=true
   -Djava.net.preferIPv6Addresses=false
   ```

2. In the Classpath tab, add the top-level folder for the gffs-structure and the gffs-webservices library projects to the classpath via the “Advanced” button and the “Add folders” option.

3. To change the default location of the state directory, create a new variable in the Environment tab called GENII_USER_DIR, and set the new location. Note that you can just inherit this from your shell environment if it is already set.
H.4.7. Running Genesis II

Once the above is set up, here are the steps to start running Genesis II:

- Run the Build Codegen target in order to generate Java files from WSDL files.
- Refresh the project in order to compile the source files and the generated files.
- Run the Container profile.
- Run the Client profile.

At this point, the Grid shell should be running and it should be ready to accept commands.

H.5. Building Genesis II GFFS Installers

To rebuild the installers for all platforms, you will need the following:

- Genesis II GFFS Source Code (see section H.2 and H.3).
- XSEDE Tests and Tools Suite (see section I.1).
- Install4j installation program.
- JRE Bundles for the versions referenced in the Install4j installer file. (Install4j has documentation here about [JRE bundles](#))

There is no need to build the GFFS code, as that is done by the automated installer build script.

Follow these steps to build the installers:

- Set up the environment variable GENII_INSTALL_DIR.
- Load the XSEDE Tests environment:

  ```
  source $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh \
  $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh
  ```

- Run the automated installer build script:

  ```
  bash $XSEDE_TEST_ROOT/tools/installer/build_installer.sh {installer-config}
  e.g., bash $XSEDE_TEST_ROOT/tools/installer/build_instiller.sh xcg3.config
  ```

- That command should complete without any errors reported.
- The installers are built in $HOME/installer_products. Platform specific versions are available for Linux, Mac OS X and Windows.

H.6. Genesis Debugging Tips

Debugging Java code is generally a lot “nicer” than debugging C or other natively compiled languages, but it can also be very complex depending on the bug being attacked. This section lists a few useful tools and tactics for more effective debugging.
H.6.1.   Jstack

The jstack tool can print out the stack trace for a running java program. This is very useful when one desires to know where the process is currently executing. For example, a deadlocked Java program may become much easier to debug when one can tell which functions are involved. Jstack is included in the Oracle JDK for Java.

To use jstack, one needs to determine which Java process one cares about, and provide the process ID for that process to jstack, e.g.:

```
jstack 3212
```

This will show the stack of the process with a PID of 3212. Note that one must have permission to examine the process's memory, which will be permitted for any of one’s own processes.

H.6.2.   Yourkit Profiler

While eclipse is an excellent debugging tool, it does not provide built-in methods to track memory allocations or function call frequency. These are traditionally the purview of a profiler tool. The Genesis team has successfully used the Yourkit Profiler for Java for tracking memory leaks and finding “hotspots” where the program spends too much of its time.

The Yourkit Profiler is available at: http://www.yourkit.com/java/profiler/index.jsp
I. The XSEDE Tests and Genesis II Testing

A collection of tests and tools were created for exercising the XSEDE GFFS and EMS during the first increment of the XSEDE project. These tests have also been found very useful as a toolkit for quickly creating different grid configurations. In addition, the XSEDE tools and tests provide some scripts that are extremely useful for different types of grid operations; for example, the deployment generator scripts that are used in the Section F.2 are included in the tools package.

I.1. Getting the XSEDE Test Scripts

The XSEDE tests and tools are provided within the Genesis II installer for all platforms; they reside in a folder called “xsede_tools” under the top-level of the installation.

The tools reside in the Genesis II subversion repository and can be separately downloaded using this command if the latest version is desired:

```
# Check out the tests to a folder named ‘xsede_tools’
svn co svn://svn.xcg.virginia.edu:9002/GENREPO/GenesisII/trunk/xsede_tools \ xede_tools
```

I.1.1. Preparing the XSEDE Tests on Linux

This is the easiest environment for running the XSEDE tests, given that the tests were built using the bash shell for a Linux environment. If some of the packages used in the tests are missing (such as expect and gnu awk), these may need to be installed from the appropriate repository for your Linux distribution. Most distributions included these packages automatically however.

Linux is currently the only platform that supports the FUSE driver for mounting the grid file system; running those tests on other platforms causes them to be skipped. For FUSE installation information, see section E.3.3.2.

One test case (GFFS_Tests/Functional_Tests/gffsFileOpsTest.sh) uses the GNU GCC C++ compiler to build an application. This can be installed with “yum install gcc-c++” on Centos platforms and “apt-get install g++” on Ubuntu platforms.

I.1.2. Preparing the XSEDE Tests on Mac OS X

The test suite runs well on modern Macs with Intel CPUs. Due to some differences in the versions of a few applications on the Mac, we have built the corresponding GNU versions of those applications and they are included in the test suite. To see the full list that we override, look in the xsede tests folder called bin/macosx. These binaries will automatically be used instead of the existing system versions when the test suite is run.

There is currently no Fuse driver for the grid file system (GFFS) on the Mac OS X platform, so those tests are currently skipped in the test suites.

I.1.3. Preparing the XSEDE Tests on MS-Windows
The Cygwin Unix emulation system is required to run the XSEDE test suite on Windows. This package is available at: http://cygwin.com/install.html

The default packages selected by Cygwin are the starting point of the install. In addition to those packages, the following packages are also required (see list below). Rather than using the cygwin setup program for this task, the next section describes how to install Cygwin with the apt-cyg tool. Apt-cyg is the preferred method, since it involves less interaction with the somewhat clunky Cygwin installer. If necessary, it is possible to install all the packages without apt-cyg just by using the Cygwin setup program. To find each of these packages more easily, try switching the “View” button on the Cygwin setup program to “Full” to get an alphabetized list.

```
bc
crypt
cygutils
ewmacs
email
expect
gcc-g++
git
gitk
gvim
inetutils
less
make
mutt
ncftp
openssh
perl
procps
python
sharutils
shutdown
time
unzip
util-linux
vim
wget
xinit
xterm
zip
```

### Apt-cyg Installation Process

The apt-cyg program brings the convenience of the Debian and Ubuntu installer application (apt-get) to Cygwin. This program does require a couple of additional setup steps. (This material is drawn from the apt-cyg home page: http://code.google.com/p/apt-cyg/)

1. Install the basic Cygwin packages with setup.exe (rather than the long list above), but add these two packages which are not selected by default:
   a. subversion
   b. wget
2. Download and install the apt-cyg program from within a Cygwin bash prompt:
3. Install the packages required for the XSEDE test suite:

```bash
svn --force export http://apt-cyg.googlecode.com/svn/trunk/ /bin/
chmod +x /bin/apt-cyg
```

```
apt-cyg install bc crypt cygutils emacs email expect gcc-g++ git gitk gvim \ inetutils less make mutt ncftp openssh perl procs python sharutils \ shutdown time unzip util-linux vim xinit xterm zip
```

4. The installation will run for a while but then should conclude with all required packages installed.

**Windows Testing Notes**

There is currently no Fuse driver for the grid file system (GFFS) on the Windows platforms, so those tests are currently skipped in the test suite.

It is also highly recommended to install the Genesis II client or container to a simple location without spaces in the name. For example, if you install the grid client to “C:/GenesisII”, then the GENII_INSTALL_DIR for the test’s configuration file (see below) will be “C:/GenesisII”.

The default value used for the state directory (GENII_USER_DIR) by the Genesis II Java software on Windows is “C:\Users\{current-user}\genesisII-2.0” on Windows Vista and later, where {current-user} would be replaced by the logged-in user’s name. On Windows XP and earlier, the default directory is “C:\Documents and Settings\{current-user}\genesisII-2.0”. The user’s home directory can be uniformly referred to in both of these cases by the expression “${HOMEDRIVE}${HOMEPATH}” using the built-in variables on Windows.

For testing with the grid client on Windows, it is recommended to use a simple user state directory for the GENII_USER_DIR environment variable when running test scripts. We recommend something such as: “C:/Genesis-State”.

If one has installed a container on Windows, the container service will use the default directory based on the user that owns the container service as described above (in other words, “${HOMEDRIVE}${HOMEPATH}/.genesisII-2.0”). To change this directory for the container service, one must set the environment variable GENII_USER_DIR in the control panel’s advanced settings for the “System” rather than for the current user.

**I.1.4. Grid Permissions Required for Running the Tests**

Some of the tests require permissions that are not necessarily available by default. These are summarized below.

*I.1.4.1. Allow new RNS directories to be created*

This is used in the test GFFS_Tests/Performance_Tests/localAndRemoteData.sh.
The above may just be required on BootstrapContainer in a one container system, rather than both gffs1.xsede.org and gffs2.xsede.org.

I.1.4.2. Make an export directory on the container under test

Make a directory to serve as the value of the EXPORTPATH variable (in xsede_tools.cfg file, discussed later) on the container specified by CONTAINERPATH. This directory must exist in the Unix account where the container actually runs, and the xsede_tools.cfg file must be updated to point at that exact folder path for export tests to be successful. This is a little odd, since clients may be referencing a folder on a remote machine in xsede_tools.cfg, but it is necessary because the export is relative to the container that creates it. This is used in GFFS_Tests/Functional_Tests/gffsGridCommands.sh and directoryTree.sh tests.

I.1.4.3. Allow exports by gffs-users members

This is used in GFFS_Tests/Functional_Tests/gffsGridCommands.sh and directoryTree.sh tests.

grid chmod /resources/xsede.org/containers/gffs-1.xsede.org/Services/LightWeightExportPortType +rx /groups/xsede.org/gffs-users
grid chmod /resources/xsede.org/containers/gffs-2.xsede.org/Services/LightWeightExportPortType +rx /groups/xsede.org/gffs-users

The container path may differ depending on the grid under test.

I.2. Running the XSEDE Tests on a Grid

For the following instructions, we will give an example command line for each step.

I.2.1. Setting up the test suite

The test suite uses a configuration file called “xsede_tools.cfg”. This file is the main switchboard that defines where the tests will find users, home directories, queues, containers, and so forth. The testing configuration file can be specified via the environment variable “XSEDE_TOOLS_CONFIG_FILE”. This variable can be set to any location, enabling the configuration file to reside in a directory other than the xsede_tools folder. If the variable is not defined, then the testing config file defaults to “$XSEDE_TEST_ROOT/xsede_tools.cfg”.

To prepare the xsede_tools.cfg using the provided example file for the XSEDE grid, do the following for the interactive installer (which allows the installation directory to be modified):

```
# Change into the xsede_tools directory:
cd $GENII_INSTALL_DIR/xsede_tools

# Copy the example XSEDE configuration file to xsede_tools.cfg:
cp examples/xsede_tools.cfg-xsede xsede_tools.cfg
```
Use the following steps for configuring the test suite’s config file on a system-wide installation (such as an RPM install or system-wide deployment of the interactive installer):

```
# Copy the example XSEDE configuration file to xsede_tools.cfg:
cp $GENII_INSTALL_DIR/xsede_tools/examples/xsede_tools.cfg $HOME/xsede_tools.cfg
# Configure the config file environment variable to point at the config file:
export XSEDE_TOOLS_CONFIG_FILE=$HOME/xsede_tools.cfg
```

If you are creating a new bootstrapped miniature test grid, then edit the xsede_tools.cfg to match the type of bootstrapped grid desired.

If you are running tests against an existing grid, then the configuration file must match the resources available in the grid. The grid administrators or help staff can often provide an appropriate xsede_tools.cfg. There are example files for other grids besides the XSEDE grid stored in the “examples” folder in the test suite.

These environment variables should be defined ahead of time before running scripts from the tools and test suite:

- **GENII_INSTALL_DIR**: point this at the location of the Genesis II installation.
- **JAVA_HOME**: This should point to the top-level of the installed JDK. Mainly used in the deployment generator tool and during bootstrapping of test grids.
- **GENII_USER_DIR**: set this if you want to store the grid state in a different location than the default. The default state directory is “$HOME/.genesisII-2.0”.

Modify the new xsede_tools.cfg for the following variables:

- **DEPLOYMENT_NAME**: set this to the deployment name for the installation. In the XSEDE and XCG installers, the deployment name is “current_grid”. The installer for GFFS.EU uses a deployment name of “gffs.eu”. A simple bootstrapped grid would have a deployment name of “default”.
- **BASE_USER**: This should be set to the grid user name that the tester is logged in as. The user’s STS and home directory paths will be based on this variable. If one is bootstrapping a test grid, then this user will automatically be created.
- **EMAIL**: An email address that can receive the grid job stage out report for certain of the EMS tests.
- **QUEUE_NAME**: The queue where jobs will be submitted. This should be the queue’s short name, assuming it is located in the grid’s resource area, e.g. /resources/xsede.org/queues. If the queue is stored under a different path, adjust the QUEUE_PATH variable instead. When bootstrapping a test grid, the path specified here will be created as a queue with one BES. For testing on an existing grid, the appropriate existing queue should be used.

The XSEDE_TEST_ROOT variable is frequently referred to in command examples. It is set up by the prepare_tools.sh script (see next section).

### 1.2.2. Initializing the test environment
Start using the tests by running the test preparation script:

```
source $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh \
   $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh
```

This step is needed before running other scripts from the test suite.

(The script is mentioned twice on the command line to get around bash's 'source' command having no script name parameter ($0) to intuit the working directory from. The second script path provides that missing information.)

**I.2.2.1. Loading the tools configuration as environment variables**

The variables in the xsede_tools.cfg file can be loaded into environment if desired. This is not usually needed, unless one wants to run individual functions from the xsede tools library. Modify the xsede_tools.cfg configuration file as described previously. Then to load the variables in the current shell, execute the following steps (this depends on GENII_INSTALL_DIR having been previously defined):

```
# Load the tools environment initialization script.
source $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh \
   $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh

# Load the functions for managing the tools configuration file.
source $XSEDE_TEST_ROOT/library/process_configuration.sh

# Instantiate all the variables from the config file into the environment.
define_and_export_variables

# Show the important variables.
var GENII_INSTALL_DIR GENII_USER_DIR XSEDE_TEST_ROOT DEPLOYMENT_NAME
```

One should perform these steps again after making any changes to xsede_tools.cfg. These steps are also required again if one exits from the bash shell. If running the test suite a lot, we recommend putting these steps into a small script for repeat use (e.g. in the $HOME/.bashrc file for the user).

**I.2.3. How to Bootstrap a Miniature Test Grid**

This is the quick start process for creating a grid for testing. This grid will have a grid queue and a BES for running jobs on. Note that this type of test grid is not suitable for deployment in a production capacity. This step creates the test grid based on the current xsede_test.cfg:

```
bash $XSEDE_TEST_ROOT/library/bootstrap_quick_start.sh
```

This gets a container running in one step, based on the variables defined in the XSEDE tools configuration file. The user defined in BASE_USER will be created during the bootstrapping. Note that the user's password is set to FOOP by default. The default passwords can be adjusted by creating a folder $HOME/.secrets and populating it with a “grid_passwords.txt” file. For example:

```
# password for the admin user:
ADMIN_ACCOUNT_PASSWD=xyzzy
# password for the normal user (USERPATH):
NORMAL_ACCOUNT_PASSWD=notfoop
```

**N.B.** The bootstrap process requires the “unlimited” JCE encryption jar files that are discussed in the section “Unlimited Strength JCE Files”. The certificates used for the miniature grid’s security are
all created on the fly during the bootstrap process, and they use larger key sizes than supported by the default JCE jars.

I.3. Running the XSEDE Regression Test

To run the entire test suite: bash regression_test.sh

To run just GFFS tests: bash regression_test.sh GFFS

To run just EMS tests: bash regression_test.sh EMS

I.3.1. What to Expect From the Test Run

The regression test script will output a few informative lines of text before printing a table of the tests that it intends to run. After this, all of the tests listed will be executed and they will each produce output. Each individual test suite (separate bash script) produces a summary at the end of its run with a count of tests and a report of the tests success or failure.

At the end of all the regression tests, the table of tests is printed again with the results of each test suite. For example, this is a full run that had no errors in any test suite:

Results table for this test run:
01: OKAY -- GFFS_Tests/Functional_Tests/gffsGridCommands.sh
02: OKAY -- GFFS_Tests/Functional_Tests/gffsFuseCommands.sh
03: OKAY -- GFFS_Tests/Functional_Tests/gffsFileOpsTest.sh
04: OKAY -- GFFS_Tests/Functional_Tests/gffsReplCommands.sh
05: OKAY -- GFFS_Tests/Performance_Tests/directoryTree.sh
06: OKAY -- GFFS_Tests/Performance_Tests/localAndRemoteData.sh
07: OKAY -- GFFS_Tests/Performance_Tests/gnuMake.sh
08: OKAY -- GFFS_Tests/Performance_Tests/largeRNSDirectory.sh
09: OKAY -- GFFS_Tests/Scalability_Tests/multiuser-gffs-submit.sh
10: OKAY -- GFFS_Tests/Functional_Tests/rnsBearTrap.sh
11: OKAY -- EMS_Tests/besStatus/bes-attributes-and-activities.sh
12: OKAY -- EMS_Tests/queueFunctionalityTests/queue-submission-test.sh
13: OKAY -- EMS_Tests/fileStagingTests/protocols-test.sh
14: OKAY -- EMS_Tests/besFunctionality/bes-submission-test-sync.sh
15: OKAY -- EMS_Tests/multiUserTests/multiuser-3users-manyjobs.sh
16: OKAY -- EMS_Tests/multiUserTests/multiuser-10users-fewjobs.sh
19: OKAY -- EMS_Tests/queueParallelJobs/queue-parallel-jobs-test.sh
20: OKAY -- EMS_Tests/faultJobsTests/bes-submission-sync-fault.sh
21: OKAY -- EMS_Tests/rnsEnabledBESTests/bes-test-async.sh
22: OKAY -- EMS_Tests/rnsEnabledBESTests/bes-test-submission-point.sh
OK: All 22 Tests Ran Successfully.

A test with a failure in it will have “FAIL” next to the test that failed, and the final output line will start with “FAILURE”. For example:

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Status</th>
<th>Test Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>FAIL</td>
<td>GFFS_Tests/Functional_Tests/gffsGridCommands.sh</td>
</tr>
<tr>
<td>02</td>
<td>OKAY</td>
<td>GFFS_Tests/Functional_Tests/gffsFuseCommands.sh</td>
</tr>
<tr>
<td>03</td>
<td>OKAY</td>
<td>GFFS_Tests/Functional_Tests/gffsFileOpsTest.sh</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>OKAY</td>
<td>EMS_Tests/rnsEnabledBESTests/bes-test-submission-point.sh</td>
</tr>
<tr>
<td></td>
<td>FAILURE</td>
<td></td>
</tr>
</tbody>
</table>

I.3.2. Reporting Bugs Seen in a Test

If there is a need to report a bug in Genesis II that affects XSEDE, please email to help@xsede.org for assistance in filing the bug. If one has a Jira account for XSEDE, then bugs can be filed at https://jira.xsede.org/secure/Dashboard.jspa

If there is a bug in Genesis II and the user is working in the XCG environment or another grid, please send an email to the XCG Developer's List: xcgdevel@virginia.edu

In either case, please include as much detail about the Genesis II problem as possible, and which of the XSEDE tests reported the problem. Including a log of the console output from that test is very important, as is including relevant pieces of the grid-client.log or container.log (stored by default in $HOME/.GenesisII). If any extra steps were required to reproduce the problem, those are especially valuable (for example, container was shut down, or resource was unavailable at the time, etc).

I.4. Helpful Notes for the Tests

The regression test is a good reference for our entire list of tests.

Some of the individual tests require changing to their directory before running them in isolation. The regression suite does this automatically.

If assistance running the tests is needed, users can email the XCG Mailing List at xcghelp@cs.virginia.edu.

I.5. More Information on the Bootstrap Process

<Placeholder section>

I.6. Grid Administrator Steps to Enable Testing

This section documents tasks that the grid administrator should take before testing begins. They establish the proper environment for testing by including a grid queue, a BES for job execution, a group with access to the queue and BES, and adding the actual users who will test.

The following assumes that a basic grid has already been bootstrapped and is available for configuration. See Deployment of the GFFS for details on deployment and setting up grids.
• Create a BES for testing. See Basic Execution Services (BES) for details on adding a BES.
• Create a Queue for testing. See Grid Queues for details on creating queues.
• Add the BES to the Queue. See Linking a BES as a Queue Resource for more information.
• Set the number of the slots for the BES to be large, e.g. 200-400. This lets the tests run at a reasonable clip. See Linking a BES as a Queue Resource for info on queue slots.
• Create a Group for testing. See User and Group Management for details about group creation.
• Create the Users who will be running tests. See User and Group Management for information about creating the users.
• Create a test “xsede_tools.cfg” that lists all resources for this grid. See Setting up the test suite for more details on the input file format.

I.6.1. Cloud VM for Replication Tests

Recently a cloud-based virtual machine for testing replication was deployed on the Amazon cloud, as it allows us to conveniently manage the Genesis II container state to prove that replication is working. These are additional steps needed to set up that VM’s container and get it ready for a replication test.

1. Set up the VM on amazon’s AWS cloud, some other cloud service, or as a VM running under your control (e.g. using virtualbox).
2. Install the Genesis II container on the VM.
3. Link the container into the grid.
4. Give Read and Execute permissions to the uva-idp-group on the container’s ports:
   a. VCGRCContainerPortType
   b. EnhancedRNSPortType
   c. RandomByteIOPortType
   d. X509AuthnPortType
   e. KerbAuthnPortType
   f. GeniiResolverPortType
5. Ensure that the test user is listed as a member of the uva-idp-group.
J. Appendix 1: FAQs and Troubleshooting Guide

J.1. Grid Client Problems

If the solutions here do not lead to success in solving your problem, the mailing list can be asked for more in-depth help at xcghelp@cs.virginia.edu.

J.1.1. What does “Internal Genesis II Error -- Null Pointer Exception” mean?

When this error occurs while running the grid client, a frequent cause is that the container serving the client is not currently running. Ensure that the login container is running, such as by doing a ping on it. This example checks that the main XSEDE STS server is available:

```
./grid ping /resources/xsede.org/containers/sts-1.xsede.org
```

Another potential cause for the NullPointerException is a bug in the credential expiration handling that affects some versions of Genesis II. If a simple “whoami” command displays a NullPointerException, then try logging out of all grid credentials:

```
./grid logout --all
```

Afterwards, try to log in again:

```
./grid login --user=userX  # ...or...
./grid xsedeLogin --user=userX
# Now the whoami command should not complain:
./grid whoami
```

If there is still an exception printed after logging out of all credentials, then please contact the xcghelp@cs.virginia.edu mailing list about this issue.

J.1.2. Why Can't I Login With My Valid XSEDE Portal ID?

The Kerberos authentication used for XSEDE logins requires that there be very little time skew between the Kerberos server and the clients using it. If there is a large difference in perceived clock times, then the credentials may be considered already invalid or not yet valid, simply because the time range for which they are valid is not agreed upon between the client and the server.

This error can manifest in the following way when login is attempted:

```
grid xsedeLogin
```

may display the following:

```
Replacing client tool identity with credentials for "CN=Drake Valusic, O=National Center for Supercomputing Applications, C=US".
Error authenticating to Kerberos domain.; nested exception is:
Error authenticating to Kerberos domain.
```
To avoid this problem, it is important to deploy a time synch service such as ntp in order to keep the Kerberos server and client hosts in agreement about the time.

**J.1.3. Why Does Client-UI Get out of Sync?**

A user may notice that the client-ui does not automatically reflect changes made in a grid client or another client-ui. This is intentional, and is due to the potential communication delays in querying assets to keep them up to date. To have the client-ui subscribe to all files and directories in order to update for changes is simply not feasible.

The client-ui does provide a refresh feature that makes it update whatever is being shown in the user interface. This is accessible on the View menu as the Refresh menu item in newer versions of the client-ui. In both the older and newer clients, the “F5” key may be used to refresh the RNS Tree in the client-ui (although apparently on some keyboards this key is consumed before the client-ui is allowed to see it).

**J.1.4. Why Is My Grid Client Logged in Differently than Client-UI?**

The grid client maintains a user context with logins that have succeeded (the credentials wallet), and all console-mode grid clients running as the same user ID (and using the same GENII_USER_DIR for the runtime state directory) will see the same authentication information. This allows users to log in once with their client and to re-use the authentication information between runs of the grid client.

The client-ui will initially start with that same login identity as the grid client. However, the client-ui then lives in its own separate user context, and the UI can be logged out or logged in as another user without affecting the grid client's user context. This is a feature of the client-ui that separates the credentials wallets so that grid clients can continue operating in their singular context, while possibly many running client-ui’s can operate as wholly different users.

There is simply no way to distinguish the grid client’s wallet as easily as the separate windows sessions of the client-ui allow, and usually no reason to do so. But it is often very convenient for a grid administrator to be able to authenticate as different users when running the client-ui, then perform operations as those users, and not affect the administrator's command-line grid clients (which could be in the background churning on scripts under the previous user context).

**J.1.5. How Can I Deal With Memory Problems?**

**J.1.5.1. Client or Container has insufficient memory**

There are some use-cases when the grid client or client-ui require more memory than they are configured to use by default (for example, submitting 1000 jobs at once). If there are complaints about memory or memory exceptions in the grid-client.log, then it may be necessary to give the client more memory. In the event of a memory issue, shut any running grid clients down before attempting the following.
Giving the grid client more memory can be accomplished using the “pass-through” command supported by the Install4j-based installers. To increase the memory size allowed for the grid client, start it with this option:

```
# Example of starting the grid client with 2 gigabytes of memory.
grid -J-Xmx2G
```

This may need to be tuned for your particular application. It is important to not grant more memory to the client than the target computer can support, or performance will be adversely affected.

Note: if you are running the grid command from source code, then omit the -J in the above command, e.g. “grid -Xmx2G”. (The -J denotes a pass-through option that the install4j installer will give to the actual program being launched by its wrapper.)

The container memory allowance can be adjusted by editing the file JavaServiceWrapper/wrapper/conf/wrapper.conf in the installation. The field called wrapper.java.maxmemory specifies the amount of memory granted to the container in megabytes. This can be increased as needed.

**J.1.5.2. Installer requests too much memory**

If the installer will not start, then it is possible that Install4j has automatically allocated too much memory. If this is the case, then the memory problem is usually mentioned in the errors shown by Install4j. Memory may be limited by system or user level policies. Try launching the installer with a different memory configuration, for example:

```
bash genesis-xsede-v2.7...script -J-Xmx2g
# above allows installer to use 2 gigabytes; below only allows 512 megabytes.
bash genesis-xsede-v2.7...script -J-Xmx512m
```

The memory value above can be chosen based on knowledge of the limits that are in effect. On Unix, those may be revealed with ulimit, e.g.:

```
ulimit -a
```

Ultimately, the installer needs a certain amount to run successfully. 512 megabytes appears to be sufficient, but lower amounts are not tested. If the imposed limits are not sufficient, another approach is to request that the system administrator raise the limit to allow the installer to run.

**J.1.5.3. Client or Container requests too much memory**

The container or client may automatically be requesting more memory than permitted by the system. This is the same issue that can affect the Install4j installer (discussed in item J.1.5.2). The solution is to adjust the amount of memory requested on startup of the client or container as per FAQ item J.1.5.1 above, but to reduce the value to a suitable amount for the system.

**J.1.6. Why Can’t the Installer or Grid Client Connect to the Grid?**
This problem can occur while installing the Genesis II software for a number of reasons.

- The most obvious thing to check is whether the grid is actually online or not (see FAQ item J.1.1 regarding how to ping the grid). This requires allowing the installer to continue and finish, despite the connection error. For example, this command pings the root container:

```
grid ping /
```

If that command fails to run successfully, the root of the grid namespace might actually be down. Try connecting to the grid (next step) and the ping (above) again to see if the problem was a disconnected grid client.

- Try reconnecting to the grid using the installer’s configuration; this may succeed in getting past a connection problem. The command takes two parameters: a context.xml file representing the grid’s root EPR and a deployment name. For example, the XSEDE production grid can be reconnected to using this command (assuming an interactive installer). This command will also work for the XCG Grid (Cross Campus Grid):

```
cd $GENII_INSTALL_DIR;
grid connect local:deployments/current_grid/context.xml current_grid
```

The proper context.xml file can be found under the deployments folder of the installation. For example:

```
$ ls $HOME/GenesisII/deployments/
current_grid/ default/
```

The ‘default’ deployment is overridden by the real deployment folder, which may be called ‘current_grid’ or ‘gffs_eu’ or another name, depending on the actual grid in use. A file called ‘context.xml’ will be located in that real deployment folder. The connect command requires the path to that file (prefixed by ‘local:’) and the name of the real deployment folder.

- In some cases, an existing user/container state directory can cause problems for the client, especially if it is for a different grid than the one provided by the installer.
  - For a client installation in this situation, try removing the user state directory (located in $HOME/.genesisII-2.0 by default).
  - For a container installation with existing resources, do not remove the state directory. Instead, remove the files in the state directory that start with “user-“, which are typically user-config.xml, user-context.xml, and user-transient.dat.

- If the GENII_INSTALL_DIR variable has at some point been out of synch with the real installation location, then problems can result. In this case, try removing the directory residing in $TMP or $TEMP or /tmp that has a name “osgi-genII-$USER” (where $USER is the user name under which Genesis II is running).

J.1.7. What does “Unable to locate calling context information” mean?

The grid client may report this error when it has not yet been connected to a grid. See the previous item (Why Can’t the Installer or Grid Client Connect to the Grid?) for more information.
J.2. Problems with the Grid Container

If the solutions here do not lead to success in solving your problem, the mailing list can be asked for more in-depth help at xcghelp@cs.virginia.edu.

J.2.1. Why is My Container Consuming So Much CPU?

Genesis II containers will generally have reasonable CPU behavior, but we have seen at least one odd system condition contribute to the Java process for Genesis using a very large amount of CPU in a mostly quiescent system. The following article describes a problem seen for the “qpidd” process as well as for Java daemons:

http://mattiasgeniar.be/2012/07/02/centos-6-usrsbinqpidd-eating-up-all-cpu-at-100

The steps to fix this issue are quite simple:

```
$ sudo /etc/init.d/ntpd stop
$ sudo date -s "\`date\`"
$ sudo /etc/init.d/ntpd start
```

If your system was experiencing the problem described in the article, these steps will immediately calm the qpidd process down to near zero CPU usage. They also calmed down the Java process for the container that had previously been overactive. No system restart or process kill was required besides the above steps.

J.2.2. How Can I Improve File Transfer Bandwidth?

One aspect to file transfer performance is the size of the network buffers used for the transmission. We have found that by increasing the buffer memory allowance, we can increase the file transfer bandwidth, but only to a point. For example, we tested with the default buffer size, with a 2 megabyte buffer, and with a 4 megabyte buffer. For the particular data set in the test, the 2 megabyte buffer provided approximately a 70% increase in bandwidth, while the 4 megabyte buffer was also faster than the default but slightly slower than 2 megabyte buffers.

The xsede test suite provides two tools which can help in performing bandwidth testing and in adjusting the Linux kernel TCP configuration:

- `show_tcp_memory.sh` Displays the current buffer settings and other properties for TCP networking.
- `set_tcp_config.sh` Reconfigures the TCP buffer sizes (according to a buffer size constant defined inside the script).

These are not production-quality scripts, but they do provide a starting point for a container owner or grid administrator to begin their bandwidth tuning. There are no hard/fast answers here for how to configure the network buffers, because the bandwidth depends on many external factors, such as network topology, size of data sets, latency, and so forth.

J.2.3. How Do I Open a Port in the Firewall for My Container?
Determining if a firewall is blocking you

The GFFS container program opens a server port on the host where it runs, and this port needs to be accessible for clients or other containers to access it. Often firewalls will block the port and an exception must be added. To check if a firewall is currently blocking the container port, try this command from a different host than the container host:

```
telnet {containerhostname} {port}
```

# for example, the xcg root container:

telnet root.xcg.virginia.edu 18443

If telnet shows text like this, then the connection is working and a firewall is not blocking the container:

```
Trying 128.143.231.208...
Connected to xcg-server2.uvacse.virginia.edu.
Escape character is '^]'.
```

If telnet only shows “Trying {ip address}…” and sits there, then either a firewall is blocking the container or the container is not actually running on that host and port.

Modifying the firewall using iptables

There are many variations of firewalls on all the platforms, and a full description of these is not possible here. If one has access to a program such as “ufw”, this enables easy modification of the firewall. Absent an easy approach, modifying the firewall at a low-level is possible on Linux if necessary, using the following process.

1. Edit the file `/etc/sysconfig/iptables`
2. Add a line like this after the “:INPUT ACCEPT [0:0]” line and before any “-A INPUT -j REJECT...” lines, where `{portNumber}` is replaced with your container’s service port:

   ```
   -A INPUT -p tcp --dport {portNumber} -j ACCEPT
   
   # example using the standard port:
   -A INPUT -p tcp --dport 18443 -j ACCEPT
   ```

3. Then restart the iptables service:

   ```
   /sbin/service iptables restart
   ```

Test the container again with the telnet approach, and it should now provide the expected “Connected to” message.

**J.2.4. How Can an Expired Container TLS Certificate Be Replaced?**

Many organizations issue short term certificates that require yearly replacement. When these are used for the GFFS container’s TLS certificate, the container must be updated with a new certificate upon expiration of the existing certificate. This section describes how to re-use the existing private key (if desired) as well as providing the steps necessary to replace the existing TLS certificate.
J.2.4.1. Prepare GFFS variables

Load the variables as appropriate for your installation, e.g.

```
source ~/GenesisII/set_gffs_vars
# or
source /opt/genesis2-xsede/set_gffs_vars
```

This step must be taken again if you leave the current shell.

J.2.4.2. Extract private key and certificate from existing pfx file

This step assumes that you’re starting with an existing PFX file that contains an expired TLS certificate. If you already have a private key and certificate file to use for TLS connections, then skip to the next section.

The TLS certificate and private key are stored in a file called “tls-cert.pfx”. The location of this file can vary depending on the type of configuration in use for the container (containers can be installed with the original split-style of configuration or with the newer unified configuration model, and additionally the unified configuration supports a specialized deployment configuration). To free the user from needing to know the details of the configuration, a tool called “tell-config” can provide the location of the TLS keypair file for the installation.

Run tell-config to determine where the keypair file is located:

```
$ grid tell-config tls-keypair
Container TLS Keypair stored in: '/home/gffs1/root-state-keep/deployments/xsede_root/security/tls-cert.pfx'.
```

Copy that keypair file to a new location in order to extract its private key, which will be needed to create a new Certificate Signing Request (CSR) file.

```
cp /home/gffs1/root-state-keep/deployments/xsede_root/security/tls-cert.pfx ~/tls-cert-old.pfx
```

The following steps extract all of the necessary information from the pfx file. These steps require knowing the password of the PFX file. (If the password is not known, read step 0 below to locate the password in the container’s configuration.)

First, the private key is extracted with openssl:

```
$ openssl pkcs12 -in tls-cert-old.pfx -out tls-cert.privkey -nodes -nocerts
Enter Import Password:
MAC verified OK
```

Then the certificate is extracted from the pfx file:

```
$ openssl pkcs12 -in tls-cert-old.pfx -out tls-cert.certificate -nodes -nokeys
Enter Import Password:
MAC verified OK
```

Once the certificate file is available, the existing Subject information can be retrieved from it. This information is important for generating the CSR file later:
Save the subject information (all of the line after the subject= above) for use in generating the CSR.

**J.2.4.3. Generating the Certificate Signing Request (CSR) file**

Once you have the private key and the subject name to be used in the new certificate, you can generate a CSR file that will be submitted to your certificate authority. (If you already have a new and non-expired PEM file with a valid certificate, you can skip to the next step.)

Generate the CSR file with the following command (by replacing the {subject-from-before} with your desired certificate subject):

```
openssl req -new -key tls-cert.privkey -nodes -subj "{subject-from-before}" -out new-tls-cert.csr
```

Extending our previous example, this command would be used:

```
openssl req -new -key tls-cert.privkey -nodes -subj "/DC=org/DC=incommon/C=US/ST=IL/L=Urbana/O=University of Illinois/OU=NCSA/CN=test-gffs-1.xsede.org" -out new-tls-cert.csr
```

This should produce a file called "new-tls-cert.csr" if successful. That file should be sent to your certificate authority along with any other required information. Upon successful completion of the certificate creation process, the certificate authority will arrange for you to receive a new certificate file that is based on the private key and subject information encoded in the CSR.

* It is important that the issued PEM file contain ONLY the end-entity x509 certificate that you have been issued, and that it not include any earlier links in the certificate chain. If requesting a certificate from XSEDE, this is the download option "as X509 Certificate only, Base64 encoded". If the file inadvertently contains more than one certificate entry, then it must be edited to contain just the "leaf" certificate (which is not a CA certificate).

**J.2.4.4. Packaging TLS keypair as a PFX file**

Assuming that one has a valid certificate issued by one's certificate authority and the private key that the certificate is based on, a PFX file can be created for the GFFS container to use for TLS connections. Run the following command with appropriate filenames plugged in for the private key and new certificate file:

```
openssl pkcs12 -export -out ./tls-cert.pfx -inkey ./tls-cert.privkey \
-in tls-cert.pem -name container -passout pass:mypassword
```

Make a note of the actual password used (chosen above as 'mypassword') as this password will need to be recorded in the container's configuration properties.

**J.2.4.5. Deploying a new PFX file as the container's TLS certificate**
This step assumes that you now have a valid, non-expired certificate and private key stored in a PFX (PKCS#12) format file that you wish to deploy as your container’s TLS certificate. GFFS clients will only connect to your container if this certificate is in the client trust store, as it is used as the container’s identity on the network for secure connections. Thus it is important for the certificate to be issued by a valid certificate authority for the grid.

Determine first where the TLS keypair file is located:

```
$ grid tell-config tls-keypair
Container TLS Keypair stored in: '/home/gffs1/root-state-keep/deployments/xsede_root/security/tls-cert.pfx'.
```

Then deploy the new PFX file to the appropriate location for your installation’s configuration:

```
cp tls-cert.pfx {reported-directory}
# for example:
cp tls-cert.pfx /home/gffs1/root-state-keep/deployments/xsede_root/security/tls-cert.pfx
```

Once the TLS keypair file has been copied into place, the password must be updated (if it has changed) as described in the next section.

**J.2.4.6. Modifying the configured password for the TLS keypair file**

After updating the PFX file to be used for TLS communication, the container needs to know the new password for that file. The password is stored in a different configuration file depending on the type of configuration model in use by the container. Unfortunately, the tell-config command does not give us enough information to go on to locate the appropriate passwords file, but the necessary steps to find the right file are provided below.

In all configuration scenarios, the configuration entries for the TLS keypair password are the same:

```
edu.virginia.vcgr.genii.container.security.ssl.key-store-password=X
edu.virginia.vcgr.genii.container.security.ssl.key-password=X
```

The password is stored in two separate entries; the “key-store-password” is the password on the PFX file, whereas the “key-password” is the password on the specific key within the PFX file. Usually these two entries will have the same password contents.

The heuristic below will locate the proper file in which to modify the two password fields:

1) If the configuration file $GENII_USER_DIR/installation.properties exists for your container, then examine that file. If this file contains the above two password fields, then it is the correct file to modify for your container. Change the passwords appropriately.

2) Otherwise, your installation’s passwords are stored in the deployment folder that is active for your container. Locate this folder with the following command:

```
$ grid tell-config active-deployment-dir
Active Deployment directory is '/home/gffs1/root-state-keep/deployments/xsede_root'
```
Change to the reported directory and edit the file “configuration/security.properties” to modify the two password fields.

J.2.4.7. Post certificate update container restart

After modifying the PFX keypair file used for TLS communication, the GFFS container should be restarted:

```
GFFSContainer restart
```
K. Appendix 2: Genesis II Deployment Arcana

This appendix is a very detailed compendium of knowledge that generally does not exist elsewhere. It is slowly being consumed by the main omnibus document.

Note that much of the information here is relevant to the XCG (Cross-Campus Grid) and possibly campus bridging, but it may not apply to the XSEDE grid.

K.1. Intended Audience

This deployment reference is intended for advanced users and system administrators who would like to understand the Genesis II software at a deeper level than most conventional grid users require.

This document presents information about the Genesis II platform's infrastructure of objects and services. There is a reasonable amount of complexity involved, and experience with XML, web services standards, and grid computing is recommended.

K.2. Genesis II Model Overview

K.2.1. Genesis II “Object” Model

Genesis II follows the Open Grid Services Architecture (OGSA) object model in which entities of interest are known as grid resources [18-20]. All grid resources have

- An **interface** defined by WSDL called a porttype [4]. WSDL is XML. The porttype defines a set of functions, their input parameter types, the response (if any), and the faults (if any) that may be thrown. Any given grid resource may implement many different porttypes, just as a Java class may implement many different interfaces.
- An **address** realized via WS-Addressing Endpoint References (EPRs) [5]. An EPR is an XML data structure that contains a wsa-address that is typically a http(s) URI, an optional opaque reference parameter element, and an optional metadata element. Clients are required by the WS-Addressing specification to include the wsa-address element and reference parameter element in the SOAP header when interacting with endpoints. GX EPRs may also be WS-Naming EPRs [6, 7]. WS-Naming is a profile on WS-Addressing to support EPR identities and rebinding of EPRs to support migration, replication, and fault-tolerance.
- An **identity** realized via WS-Naming EndPoint Identifier (EPI). Each distinct grid resource will have a globally unique EPI that uniquely identifies it. Replicas of the same grid resource will share the same EPI.
- Implementation specific **state** (may be empty). The state may reside in memory, in a database, on disk, or generated on the fly.
- Arbitrary XML metadata known as **resource properties** [18-20] realized via WSRF resource properties as profiled by OGSA-WSRF Basic Profile.
Inter-resource communication in GX (IPC if you will) is realized via remote procedure call (RPC) using SOAP 1.1 on http(s) or via asynchronous notifications using WS-Notification. Data integrity on the communication channel is via mutually authenticated SSL.

GX uses a factory pattern for grid resource creation. New grid resources are explicitly created using a grid resource factory or as a side-effect of an operation. Grid resource factories may be grid-resource-type specific – only returning grid resources that implement a particular set of interfaces, or they may be generic returning an EPR that points to the desired type (or throwing a fault).

Authentication in GX is as per the Web Services Interoperability Basic Security Profile (WSI BSP) [10, 11]. Identity credentials are carried in SOAP header as per WSI-BSP (Basic Security Profile) [18-20]. These tokens may include signed assertions by trusted resources – realized using WS-Trust Secure Token Services [16].

Authorization in GX is strictly up to the resource provider. It is expected that resource providers will use authentication contained in the SOAP header as per WSI-BSP, but there is no requirement to do so. The Genesis II implementation uses access control lists.

K.2.2. Grid Containers, Services, and Resources

Genesis II is implemented using Web services to perform the tasks of the various system components – such as components to implement file and directory resources, execution service resources, grid queue/meta-scheduler, user identity provider resources, etc. Each of these components implements a well-defined set of functionality with interfaces currently described by WSDL documents. Under the hood, Genesis II uses Web application servers to host these Web service components and each application server deploys the code for all Genesis II service types (which we usually name using one of the WSDL port-types they implement). We call a Genesis II Web application server a Genesis II grid container (grid container for short) as it contains Genesis II services and service instances.

Genesis II supports a number of resource types, which are usually identified by the name of their interface – or in Web services terms, by the name of their port-type. Here is a list of some of the more commonly used Genesis II resource types and a brief description:

- **RandomByteIOPortType**: random access files with an interface that follows the OGF RandomByteIO specification [17].
- **EnhancedRNSPortType**: hierarchical directories with an interface that follows the OGF Resource Naming Service v1.1 specification [18-20] with Genesis II-specific extensions.
- **GeniiBESPortType**: Genesis II’s version of the Basic Execution Service (BES) [21]. Follows the BES specification plus Genesis II-specific extensions.
- **BESActivityPortType**: jobs managed by a BES (the BES specification calls jobs contained by a BES “activities”). The interface is Genesis II specific since there is no standard interface for activities as yet.
- **QueuePortType**: Genesis II grid queue/meta scheduler for user jobs. The interface is Genesis II specific since there is no standard interface for grid queues as yet.
- **VCGRContainerPortType**: an interface to access functionality to manage a Genesis II grid container. This is a Genesis II-specific interface.
- **LightWeightExportPortType**: an interface for exporting a local file system directory into the Genesis II global namespace. File and directory resources are represented by OGSA-ByteIO and OGF RNS 1.1 resources respectively.

Most Genesis II services can create instances of their type – which we call Genesis II grid resources (resources for short). For example, the service RandomByteIOPortType can create grid resources that adhere to the standard OGSA ByteIO interface – i.e. resources that embody file functionality. Once created, each resource is a unique, individual entity with its own state, distinct from other entities - even those with the same interface. Therefore, multiple unique resources can be created from the same service on the same Genesis II grid container. For example, many RandomByteIO file resources can be created on the same grid container, each with different state. In this case, the different state includes the different file contents. Similarly, a single grid container can house a number of grid queues, a number of execution services, a number of RNS directories, etc.

The basic model for Genesis II is to deploy a Genesis II grid container wherever one is needed. For example, if you want to create a grid execution service to manage a particular physical compute resource, then you need to find an existing Genesis II grid container that can host it – such that it can do its job properly. If there is no suitable Genesis II grid container already deployed, you need to deploy and configure a new grid container on an appropriate machine.

### K.2.3. Global Namespace

One of the key components of a Genesis II system is that it can maintain a global directory/namespace to reference and organize grid resources of all types across the entire grid. The global namespace provides the glue to stitch together the resources of the grid and makes working within the grid and sharing resources across sites and between users relatively simple and intuitive. The directory structure of a Genesis II global namespace is implemented using a grid community specification named **Resource Naming Service (RNS)** while file-like services are implemented in accordance with **Random Byte IO** and **Streamable Byte IO** specification interfaces.

The global namespace relies on the fact that all grid resources within Genesis II are given an **endpoint reference (EPR)** when it is created. Genesis II EPRs follow the WS-Addressing and WS-Naming specifications and contain a combination of identity, contact, and metadata information for a resource. Basically, they act like a pointer to a grid resource and are registered with entries in the global namespace to provide a handle with which to identify and communicate with a grid resource.

#### K.2.3.1. Resource Forks

Genesis II extends the global directory/namespace model by incorporating **resource forks** in many Genesis II grid resource types. A resource fork looks and behaves (mostly) like a set of subdirectories and files contained within a grid resource. They are used in Genesis II to export a
view of interesting information about a grid resource and to sometimes allow interaction with the functionality or configuration of a resource through a file system interface. For example, Genesis II BES resources have a resource fork which includes a construction-properties file that shows the BES resource’s main configuration document and an activities directory that contains a subdirectory for each job it is managing. Each job directory in turn has a status file (job status information) and a working-dir directory that contains the contents of the job’s working directory. Resource forks provide a great deal of new functionality that can be accessed by a user using standard Genesis II directory browsing and file viewing/editing tools.

The entries within a resource fork use the same Byte IO and RNS interfaces as the rest of the global namespace and therefore tools that act on an RNS directory or a Byte IO file can act on resource forks as well.

K.2.4. Security Model

Security is an integral part of a Genesis II grid system. To support fine grained control over authorization, grid containers maintain an access control list for each grid resource it hosts. Genesis II divides the interface for each grid resource type into “read” operations, “write” operations and “execute” operations and access control is set separately for each of the three categories – similar to the standard UNIX file system model. Access control lists are maintained as a list of security tokens that are allowed to perform the category of operations on a particular grid resource. Grid containers are responsible for finding security tokens passed with requests, validating them and checking for authorization using the access control lists.

In order to perform an operation on some grid resource (e.g. read the contents of a directory or file or execute a job), a user (or other client) first must acquire security tokens (X.509 certificates, username/password tokens, etc.) that prove they are allowed to do the operation. To use the grid by issuing commands through Genesis II client software (e.g. the Genesis II grid tool or the Genesis II client-ui GUI tool), a user must get acquired security tokens placed into the proper location for the client software to find and use them. Genesis II tools will automatically grab security tokens placed into the proper location, will create a security context (which in turn is part of a larger Genesis II concept of a user context) and will properly format subsequent messages to send along this security information. It is this security context information that receiving grid containers use to determine authorization for an operation on a grid resource.

K.2.5. Understanding a Genesis II Grid Container

A Genesis II grid container is a Java Web application server. The grid container includes an embedded HTTPS server (jetty) which opens a particular TCP/IP address and port and listens for messages destined for grid resources it contains. Since Genesis II is web service based, message formats and delivery follow the relevant web service standards as much as possible. Messages are formatted as SOAP 1.1 messages, carry security information in SOAP headers as per WS-Security specifications, carry web service operation requests/responses in their SOAP bodies, and carry additional Genesis II specific information in appropriate extensibility areas of the message. Inside the grid container, message handlers break apart the message - finding security and other user
context information in the message headers, determining the destination grid resource for the web service operation, etc. The container checks the validity of any security tokens passed to it, determines whether the operation is allowed by the access control in force for the target grid resource and either returns an appropriate fault message or routes the message to the proper piece of code to handle the request (we call the code that handles a particular type of grid resource, a grid service). The container does all this by leveraging functionality provided by Apache Axis (v1.4).

K.2.5.1. **IP Address and Port**

Every grid container must be able to acquire an IP address and port that allows users (clients) and other grid containers within the grid to send messages to the grid container. Therefore care must be taken to select a machine & IP address/port combination that allows the grid container to send and receive its messages. Unless a grid is to be local, this usually means selecting a global IP address and making sure that firewalls allow incoming HTTPS/TCP traffic for the IP address/port selected and outgoing TCP/HTTPS traffic to all possible grid clients. For situations where a grid container will be behind a Network Address translator (NAT), you must ensure that appropriate port forwarding from an appropriate accessible IP address + port is done to the address on which the grid container is configured to listen.

**WARNING:** A grid container puts the IP address or DNS name (as per the container’s configuration) and port into the EPRs of grid resources it manages. Many of these EPRs are further used in RNS entries (potentially on any grid container) to link grid resources managed by the container into the global namespace. Therefore, it is very important that the IP address/DNS name and port do not change during the grid container’s lifetime. If a grid container is moved to a different machine it must retain the same IP address or DNS name as appropriate otherwise existing EPRs will be meaningless (both in RNS entries and in internal database state for grid resources). Changing the grid container’s configuration to use a new IP address/DNS name will not help existing EPRs – it will simply start using a different IP address/DNS name for listening for messages and for EPRs for newly created grid resources.

K.2.6. **Storage**

At a minimum, all Genesis II grid containers require a *grid container state directory* to store the state of the grid container and the grid resources it contains and a *Genesis II installation directory* to store the Genesis II code and configuration information that implements the grid container. Some types of resources may require additional storage areas which are specified in configuration files or in configuration options when certain types of grid resources are created. For example, execution services require a directory that can be accessed by the grid container and all machines that can ultimately run jobs for that execution service in order to set up working directories and stage data in and out.

K.2.6.1. *Genesis II Installation Directory*

The Genesis II installation directory contains the Genesis II code and grid container configuration directories/files. Because configuration information for a grid container is contained in this
directory, a separate installation directory is required for each grid container (currently there is no way to separate the code from the configuration information). Since multiple grid containers cannot share the code directory, we usually recommend placing this directory on a file system local to where the server will run – but that is not mandatory. For brevity, the remainder of the document will often denote the path to the Genesis II installation directory as INSTALL_DIR.

**K.2.6.2. Grid Container State Directory**

The grid container state directory stores information about the state of the grid container itself and that of the grid resources it contains. This directory is used to store most state information for the container, but system administrators should be aware that other storage areas may be required to create specific types of grid resources -- depending on the requirements of the specific type of resource created and its configuration. For example, execution services require a directory that can be accessed by the grid container and all machines that can ultimately run jobs for that execution service. Please read the sections on creating specific grid resource types later in this document. For brevity, the remainder of the document will often denote the path to the Genesis II installation directory as STATE_DIR.

**K.2.6.3. Grid Resource-Specific Storage**

The grid container stores the state of many grid resources completely within the grid container’s database. However, some grid resources require additional storage areas to perform their functions or require access to existing directories or files. Some of these are placed inside the grid container state directory, while others require user or system administrator configuration to tell the grid container the paths to the proper storage to use. Below is a list of grid resources that require additional storage areas (referenced by a description of what the grid resource does as well as its service name).

**Random access file (RandomByteIOPortType)**: Genesis II grid containers do not store the contents of a random byte IO file in the grid container database. Instead, the grid container maintains a separate directory (named rbyteio-data) within the grid container state directory to store file contents. The rbyteio-data directory contains two levels of subdirectories with names `dir.<#>/dir.<#>` and stores the contents of a random byte IO file in file one of these directories in a file named `rbyteio<big #>.dat`. The mapping from random byte IO resource to the path that holds its contents is stored in the grid container database. The reason for the 2 level structure and the use of numerous directories is to spread out the potentially large number of random byte IO files such that no directory has a large number of files – a known performance problem with many file systems.

**Streaming file/data (StreamableByteIOPortType)**: Genesis II grid containers do not store the contents of a streamable byte IO file in the grid container database. Instead, the grid container maintains a separate directory (named sbyteio-forks) within the grid container state directory to store streamable file contents.
Exported Directory (ExportedRootPortType and LightWeightExportPortType): Genesis II supports the notion of mapping a portion of a local file system into the global grid namespace. The idea is simple: a user chooses a rooted directory in their local file system and creates a grid resource (either of type ExportedRootPortType or LightWeightExportPortType) that acts as the grid proxy for accessing data in the target directory. This is referred to as exporting a directory into the grid. In order to do this, the user must find (or deploy) a Genesis II grid container that can access the target directory. In other words, the grid container must be on a machine that has the directory mounted and is running as a user account that has permission to do the desired operations on the target directory.

GeniiBESPortType: Basic Execution services are, not surprisingly, perhaps the most complex Genesis II grid resources to configure. First, there are several different flavors of Genesis II BES resources – each follows the same BES interface and functionality, but each executes jobs in a different execution environment and therefore requires different pieces of configuration information. All Genesis II BES resources also have some properties in common.

All support:

- Providing a unique working directory for each job;
- Providing an area for a “scratch” directory (as defined by JSDL). Data files may be staged into a scratch directory outside of a job’s unique working directory so that files can be reused across multiple jobs without having to stage the same file over and over again;
- Support stage-in of data files to the job’s current working directory and/or to the BES resource’s scratch directory;
- Support stage-out of data from job’s current working directory and/or to the BES resource’s scratch directory;

The bottom line is that thought should be given to the type of BES resource desired and the configuration it requires when choosing a machine to host the grid container in which it will reside. More detailed information on configuring all of the BES flavors is provided below.

K.2.6.4. Transport Layer Security

Genesis II grid containers (and client tools) are usually configured to use SSL/TLS to provide secure communication channels to/from clients or other grid containers. Using SSL/TLS provides several benefits. First, by using the server authentication functionality of SSL/TLS, Genesis II clients verify that the target grid container has a certificate signed by a trusted authority – therefore SSL/TLS in Genesis II provides a measure of server authentication. Second, SSL/TLS provides both message integrity and privacy.

In order to use SSL/TLS, a grid container must be configured to do so. This involves a number of configuration file settings and the provision of appropriate key and trust stores. In particular, a grid container must have an SSL trust store configured that will allow the grid container to trust connections made from clients. Clients that call the grid container must use SSL certificates that
match the trust chains in the grid container’s SSL trust store. Also, since the grid container sometimes acts as an SSL client to make outcalls, the grid container must also have an **SSL client key store** configured for outgoing calls to other grid containers.

Using SSL/TLS in Genesis II is optional, but is strongly encouraged and is usually configured by default in most Genesis II installers.

### K.2.6.5. Grid Container Administrator

Genesis II has a built-in notion of a **grid container admin user**. All container services and all grid resources created on the grid container allow the grid container admin user to perform any operation. Each grid container is configured to define which certificate identifies its admin user. Some Genesis II grid systems (such as the UVA Cross Campus Grid) install the same admin certificate for all grid containers so that there is a single grid-system-wide admin that can perform administrative tasks for the entire system. Other Genesis II grid systems may choose a more federated admin model.

### K.2.6.6. Grid Container Account

Each grid container runs under a particular user account. This account must be able to maintain the state of the grid container and perform the operations of the grid resources desired. Special care must be taken to ensure that the user account is appropriate when creating an execution service (i.e. a BES to run jobs) and when creating an export (proper permissions on target directory).

### K.2.6.7. Grid Resource Access Control

Grid containers are responsible for enforcing Genesis II RWX access control policies for each grid resource they host. Incoming requests contain security tokens which the grid container uses to determine if the request should be allowed to proceed based on the grid resource’s access control list. The grid container verifies each incoming security token – both to make sure that it is still valid (e.g. not expired, etc.) as well as trusted. A grid container’s trust of an incoming security token is determined by the grid container’s configuration. In particular the **grid container resource identity trust store** holds certificates for trusted certificate authorities. The grid container checks each incoming security token and only accepts those that have been signed by a trusted CA as identified by the container resource identity trust store.

### K.2.6.8. Trust

The setup of trust relationships for a grid container is a crucial component in the overall security model of Genesis II. Trust is needed for SSL/TLS connections to be established between clients and grid containers as well as to verify security tokens for access control. In many grid system deployments – such as UVA’s Cross Campus Grid (XCG) – trust is established by having a **grid-wide CA** that is trusted by the various components of the grid system. The grid-wide CA is then used to sign both SSL/TLS certificates and user security tokens for clients and grid containers and is also included in the SSL/TLS and resource identity trust stores for clients and grid containers. In this
way, the grid-wide CA provides the trust backbone for a particular grid system. Note that there is no requirement that the system use a grid-wide CA; this is simply a common and convenient way to manage trust within a grid system when appropriate.

To make acquiring signed certificates for grid containers easier in a Genesis II grid system, Genesis II supports a CertGeneratorPortType grid resource type which is designed to handle requests to sign new certificates with a particular CA certificate (subject to access control policy).

Genesis II installers for particular grid system deployments often manage setting up a reasonable default security configuration for a grid container. This includes, generating appropriate certificates, deploying appropriate trust stores, and setting proper security configuration information for the grid container.

K.2.6.9.  Grid Container VCGRContainerPortType Service

Each grid container has a service type called VCGRContainerPortType. This service represents the grid container as a whole and has an interface to support certain management functionality for the container. Part of its functionality is to provide a resource fork which exposes a rather expansive set of sub-directories containing status and configuration information about the grid container as well as links to the grid services it contains.

<table>
<thead>
<tr>
<th>Listing of a container and Example Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genesis II:$&gt; ls</td>
</tr>
<tr>
<td>BootstrapContainer:</td>
</tr>
<tr>
<td>resources</td>
</tr>
<tr>
<td>Services</td>
</tr>
<tr>
<td>container.log</td>
</tr>
<tr>
<td>filesystem-summary.txt</td>
</tr>
<tr>
<td>Example Grid Container RNS Entry :</td>
</tr>
<tr>
<td>VCGRContainerPortType</td>
</tr>
<tr>
<td>urn:ws-naming:epi:AB3458-9374-..</td>
</tr>
<tr>
<td>urn:ws-naming:epi:2FE878-3745-..</td>
</tr>
</tbody>
</table>

The above shows a portion of the resource fork that each Genesis II VCGRContainerPortType service exposes.

Resources contains a sub-directory for each grid service in the container (these are named after the Java class that implements them, which may be a bit confusing, but in practice it is pretty easy to match the names to the service). Within each service-specific sub-directory is an entry for each and every grid resource created for that service type on that container. For example, for the RandomByteIOPortType, there is an entry for every random byte IO grid resource on that container. The names of the grid resource entries are system generated (there is no official user-friendly name for any Genesis II grid resource) which is the EPI part of its EPR. The entries are real links to the actual grid resource, and therefore one can use the entry to talk to the grid resource, further browse into its resource fork, etc. and use it to make additional links to the resource in the global namespace. The summary entry contains a textual summary of the number of grid resources for each grid service type.
N.B. Grid services that do not have any existing grid resources are omitted from the resources directory for brevity.

Filesystem-summary.txt contains status information about file systems that are currently monitored by the container.

Services contains an entry for each grid service in the container. This is a link to the service itself and can be used to talk to the service.

Tip: The create-resource tool requires a path to the grid service on the grid container on which the resource is to be created. The RNS path <grid container entry>/Services/<service name> is the most convenient way to point to the desired service.

N.B.: Some Genesis II tools (e.g. the client-ui browser tool) get a bit confused when talking directly to a service. This is because the code that tries to identify the interfaces supported for a grid resource identifies the services as being of the same type as the resources they create. Unfortunately, this can cause errors when interacting with the service. For example, the random byte IO grid resources perform file operations: read, write, etc. The RandomByteIOPortType grid service does not perform file operations, but instead performs operations to create, destroy, and manage grid resources of type RandomByteIOPortType. Some tooling simply treats the service like instances of the service and errors ensue.

Container.log is an embodiment of the actual container.log file for the grid container -- reflected through its resource fork. This can be a really useful tool for debugging remotely -- one can access the log remotely using standard Genesis II tools.

Another benefit of having the VCGRCContainerPortType service is that it is a true Genesis II grid service and therefore it is a first class entity in Genesis II, including having an EPR and Genesis II ACL support. This means that this service can be linked into a grid’s global namespace as an RNS entry and all of the information it exports via its resource fork can be accessed using the global namespace and standard Genesis II client tools. It also means that access to the service can be controlled using standard Genesis II ACLs.

K.3. Pre-Installation

The majority of the work in installing a Genesis II grid container comes before running the Genesis II installer. A number of decisions need to be made in order to choose the proper machine, account, and storage areas to host the grid container.

- Decide what grid resources you expect to run on the container;
- Determine security requirements for grid container;
- Review the configuration requirements of the grid container for supporting the types of grid resources you expect to deploy on the container to determine the characteristics needed for the host machine;
- Based on (1-3) select the appropriate hardware and configuration for the grid container:
  - An appropriate machine
  - IP address, port number
Secure communication options (SSL)
Trust & key stores
User account
Storage paths – installation, state, and grid-resource type specific

K.4. Installation

The first step to install a grid container is to download the appropriate installation bundle. Most Genesis II installers are specific not only to a target machine architecture/operating system, but also to a specific grid deployment. This is done so that the installers can bundle not only generic Genesis II code, but also grid system specific configuration. In many cases, this scheme greatly reduces the configuration burden to deploy a new grid container and helps to avoid deployment mistakes and mismatches between different grid containers within the system.

Some of the things the grid-specific installers do:

- Turn on HTTPS/SSL communication by default;
- Deploy and configure certificates and trust stores that allow grid containers within one grid system to authenticate and/or trust each other;
- Downloads information about the root (i.e. "/") of the global namespace for the target grid system and store it in the local state directory. This allows clients and the container to communicate with the proper grid system without manual user intervention;
- Executes code to generate a new unique grid container certificate for the installation;
- Registers new grid container with target grid system’s global namespace (usually in a directory names /uninitialized-containers). This makes it easy to reference and communicate with the new grid container.

K.4.1. Planning for Installation Storage

During installation, the installer will place items in both the Genesis II installation directory and the grid container state directory.

K.4.1.1. Specify Installation Directory Path

The Genesis II installation directory is specified during the Genesis II installation process. Once specified, the installer places code and configuration templates into the installation directory. The installer also creates or modifies scripts and configuration properties in the directory to hard code proper paths for the installation. Therefore, moving the path after installation would require finding and modifying a number of files. Depending on what grid resources are created within a grid container it can sometimes be easier to reinstall a grid container rather than moving it.

K.4.1.2. Specify Grid Container State Directory Path

The grid container state directory can be specified in a number of ways. The grid container uses the following algorithm to find the path of the grid container state directory:

- If the GENII_USER_DIR variable is set in the environment, its value is used;
Otherwise, if the edu.virginia.vcgr.genii.container.user-dir property in the 
container.properties file in root Genesis II installation directory is present, then its value 
is used for the state directory path.

Otherwise, a default path is used as follows:

- For Windows: C:\Documents and Settings\<grid container user>\.genesisII-2.0
- For Linux: /home/ <grid container user>/.genesisII-2.0
- For Mac OS X: /Users/ <grid container user>/.genesisII-2.0

Note: If the directory does not exist, it will be created.

TIP: It is often easiest to use the default path if feasible. For Linux and Mac OS X installations, 
symbolic links can be used to point the default path to a different path. For example, if you desire to 
have the state directory be located at path /local/<my user>/GenesisII_state, you can set a symbolic 
link from /home/<my user>/.genesisII-2.0  →  /local/<my user>/GenesisII_state.

TIP: If you do not wish to use the default path and the symbolic link strategy will not work for some 
reason, the only other choice during installation is to use the GENII_USER_DIR environment 
variable (because the container.properties file does not yet exist). After installation, you can 
manually back-patch the container.properties file so that you do not need to continually keep the 
GENII_USER_DIR variable set.

WARNING: It is difficult/dangerous to change the path to the state directory once a grid container 
is operational. The biggest difficulty lies in certain grid services storing full path names as part of 
the state of the grid resources they manage. For example, the RandomByteIOPortType service 
stores the full path to the file it uses to store the file’s contents. This state is written to the grid 
container’s database and therefore it is impossible to change without understanding the container’s 
database schema and writing database code to change this state. It should be possible (though this 
has not been tested) to move the state directory or parts of it and use file system links to maintain 
the original paths.

K.4.2. Run Installer

The installers are graphical and by default a windowing system is expected to be operating when 
the installer is run. If a non-graphical installer is needed, run the installer with the –c option 
(console option). The installer can be executed using the normal methods on the host operating 
system. In other words, double-clicking the installer works on Windows XP and the graphical front- 
ends for Mac OS X and Linux as does opening a shell/command line prompt and executing the 
installer that way.

The installer prompts for all necessary input along the way. For a more detailed description of the 
installation process, please consult the “Getting Started with XSEDE” or “Getting Started with 
Genesis II” tutorial documents.

TIP: One issue to be aware of is that the installation procedure may silently fail if the 
information provided is inaccurate; so, be careful typing this information. If you find that
some configuration items were provided incorrectly, it is probably best to reinstall the software from scratch.

K.4.3. Start/Stop Grid Container

To manage starting, stopping, and checking the status of a grid container, the installer generates a script named GFFSContainer which is placed in the root installation directory. This script takes 1 argument - values start, stop, restart, or status – just like UNIX scripts to manage services in init.d. In fact, the script was designed to support the init.d interface on Unix-like systems so that grid containers could be automatically started upon reboot if desired.

Example: Start the grid container (on Linux):

```
cd <Genesis II installation root dir>
./GFFSContainer start
```

A container may take anywhere from 20 seconds to several minutes to completely start – depending on what is running on the container and how long it takes to run special container start up code -- such as code to do periodic sanity checks, cleanup old jobs, or download and implement automatic patches. If you wish to watch it proceed, you can watch the output in the container.log file in the Genesis II installation directory (using for example the tail command in Linux).

K.5. Post Installation: Configuring the Grid Container

K.5.1. Deployments and Deployment Inheritance

An important concept to understand regarding Genesis II grid container configuration is the notion of deployments. A deployment is a named related set of grid container configuration files. The configuration files for a particular deployment are stored in a sub directory with the same name as the deployment within the deployments directory of the Genesis II installation directory. The container.properties file contains the property edu.virginia.vcgr.genii.container.deployment-name which specifies the name of the deployment for the grid container and therefore names which deployment directory is to be used. When a grid container starts, it reads the container.properties file, determines the deployment to use, and then reads configuration information from the deployments/<deployment name> directory. A grid container configured to use the MyFavorite deployment will look for configuration files within the deployments/MyFavorite directory (and its sub-directories).

Currently, the deployments feature is not exploited to its fullest and the standard Genesis II installers deploy all grid containers configured to use the GeniiNetFullContainer deployment. Therefore, the deployments/GeniiNetFullContainer directory is where the grid container looks for most of its configuration information. However, it is still important for grid system administrators to understand the deployment concept because it determines where configuration files must be placed or modified in order to take effect with a grid container. In the future the deployments functionality may be used more extensively to create new pre-configured installer options for various situations.
To further complicate matters, deployments support an inheritance feature. Within a deployment directory, there is an optional configuration file named deployment-conf.xml. If present, this file contains an element `<deployment-conf>` that can point to a different deployment name upon which the current deployment is based. For example, the GeniiNetFullContainer deployment comes pre-configured with the deployment-conf.xml file containing the element `<deployment-conf based-on="default"/>`. This setup states that the GeniiNetFullContainer deployment is "based-on" the deployment named "default". Ordinarily, a grid container will simply look for a particular configuration file only within its deployment directory. If the file is found, it is used. If it is not found, no further searching is done by the grid container and it will take the appropriate action for that missing configuration file – failure if the file is mandatory, employing some default behavior if the file is optional. In the case where inheritance is used, the grid container will modify its search process. It will still search its deployment directory first. If the file is found there, then that file is used. If the file is not found in the deployment directory, then the grid container will search the "based-on" deployment's directory for the file and will use its information if found. In fact, the inheritance can conceptually continue further if the "based-on" deployment in turn is based-on another deployment (though this is not currently used in practice).

K.5.2. Grid Container Configuration Options

Genesis II has many configuration points; this can be a bit overwhelming if a system administrator had to do everything from scratch. Fortunately, Genesis II installers come with deployments that will set most of an installation's configuration files to acceptable values for a particular grid system. This section is intended to be both a reference section for various configuration points as well as a how-to guide for the average system administrator. The configuration topics are therefore broken into three categories. The first category -- Likely Configuration Topics -- describes configuration options that a system administrator can and very well may desire to configure for their installation. The second category -- Optional Configuration Topics -- describes options that a system administrator may want to configure in less common circumstances. The final category -- Unlikely Configuration Topics -- is included mostly for reference purposes and describes options that are very unlikely to need to be changed by a system administrator.

Likely Configuration Topics

K.5.2.1. Configure Scratch Space and the Download Manager

All Genesis II grid containers support the notion of a **SCRATCH file system** for execution services. This matches the concept in JSDL of a SCRATCH file system, which JSDL 1.0 defines as “Temporary space that persists some time after the job completes”. In Genesis II, a SCRATCH file system (sometimes referred to simply as SCRATCH space) is a useful way to try to avoid copying files commonly used among a number of jobs multiple times to the same destination execution service.

For example, imagine that a user has 1000 jobs to run which all execute the same binary and all use the same rather large read only database file as one of its inputs. Presumably, some input parameter or a different input file changes between each of the 1000 jobs. Let's further suppose that 500 of these jobs will ultimately run on BES1 and 500 will run on BES2. Copying the binary
and the large read only database to the unique working directory for each of the 1000 jobs is a huge waste of time, storage, and network bandwidth – and is unnecessary. Instead the user can define in their JSDL that the binary and the read only database file should be staged to SCRATCH space instead of to the job's working directory. If enough space is available in SCRATCH space, BES1 and BES2 should be able to copy the SCRATCH space files once each, rather than 500 times, and each job can share the one local copy. So, 1000 copies for each SCRATCH space file becomes 1 copy for each BES that runs the job.

Internally, Genesis II BES resources manage downloading to SCRATCH space using two grid container-wide internal components (called container services) – the download manager and the scratch file system manager. Note that both of these container services are grid container-wide services -- which means that all BES resources within a grid container share them and that they are configured at the grid container level, not for each BES resource.

The **scratch file system manager** controls where the root SCRATCH space is located for the entire grid container and which SCRATCH space subdirectory should be used for a given job’s SCRATCH space (jobs specify a key for their SCRATCH space file system, which is used to match SCRATCH space paths across jobs within a grid container). The SCRATCH file system manager also is responsible for managing clean up and storage usage of scratch space – which includes “locking” files used by active jobs to avoid cleaning up their required files.

The SCRATCH file system manager uses the **download manager** to actually perform any required file downloads. The download manager manages checking to see if a target file already exists and manages synchronizing multiple requests to download the same file between multiple jobs. The download manager has its own directory for temporarily storing partially downloaded files, before it moves them to their ultimate destination.

TIP: If you do not plan to start a Genesis II execution service (BES) on a grid container, there is no need to configure SRATCH space.

### K.5.2.2. Configure Scratch Space Service and Directory

The INSTALL_DIR/deployments/<deploy_name>/configuration/cservices/ScratchFSManager Container Service.xml file contains the configuration information for a grid container's scratch file system manager service. If this file is absent, the scratch file system manager service will be turned off for the grid container (and therefore SCRATCH space is inoperable). If the file exists, the value attribute for the <property> element with name="scratch-directory" defines the path for the scratch file system manager's root SCRATCH space directory. If the scratch-directory property is omitted, the scratch file system manager will use the default path “scratch-dir” within the grid container state directory.

**N.B.** The location of SCRATCH space is very important and must match how BES resources on the grid container will be configured. Since jobs run by a BES container need access to SCRATCH space, all machines where jobs can run must have access to SCRATCH space using the path defined by the ScratchFSManagerContainerService.xml file.
For example, a BES resource that feeds jobs to a cluster via a local queuing system, the SCRATCH directory must be mounted to each cluster compute node (as well as the grid container host machine) using a common mount path. Since SCRATCH space requires very similar properties to the root job working directory for BES resources, the SCRATCH space directory is often in the same local file system as the BES working directory – though there is no actual requirement for this.

Example:

```xml
<property name="scratch-directory" value="/home/gbg/shared-dir/scratch" />
```

K.5.2.2.1. Installer defaults

**XCG installer:** The INSTALL_DIR/deployments/default/configuration/cservices/ScratchFS ManagerContainer Service.xml exists, but does not specify a `scratch-directory property` element. Therefore, the scratch file system manager is turned on and uses the scratch-dir subdirectory within the grid container’s state directory by default.

K.5.2.3. Configure Download Manager Service and Directory

The INSTALL_DIR/deployments/<deploy_name>/configuration/cservices/DownloadManager ContainerService.xml file contains the configuration information for a grid container’s download manager service. If this file is absent, the download manager service will be turned off for the grid container (and therefore SCRATCH space is inoperable). If the file exists, the value attribute for the `<property>` element with name="download-tmpdir" defines the path for the download manager’s temporary storage directory. If the download-tmpdir property is omitted, the download manager will use the default path “download-tmpdir” within the grid container state directory.

Example:

```xml
<property name="download-tmpdir" value="/home/gbg/shared-dir/download-tmpdir" />
```

K.5.2.3.1. Installer defaults:

**XCG installer:** The INSTALL_DIR/deployments/default/configuration/cservices/DownloadManager ContainerService.xml exists, but does not specify a download-tmpdir property element. Therefore, the download manager is turned on and uses the “download-tmpdir” subdirectory within the grid container’s state directory by default.

K.5.2.4. Configure Local File System Monitoring

Genesis II grid containers can be configured to monitor the remaining storage space available for file systems locally accessible to the grid container. This monitoring is optional, but can be very helpful in avoiding or quickly remedying situations where file system storage is exhausted or becomes dangerously low. Note that currently the configuration of file system monitoring is completely independent of how various storage paths are defined in other configuration files. For example, there is no connection between configuring the path for scratch space or the download manager temporary directory and the monitoring service. At some point in the future, there is the
intention of combining configuration into a unified file system-centric model, but that is not the state of affairs at this time.

Monitoring is configured in the file INSTALL_DIR/<deploy_name>/configuration/filesystems.xml. You can monitor or “watch” any number of file systems accessible by the grid container. For each watched file system, you can define how often to check storage, specify what storage conditions are considered a out of bounds (based on percent or absolute space available), and specify what actions to take when a problem occurs or when status goes back to normal. Currently, actions include writing a log message to the grid container log or sending email to a specific set of addresses.

The format of the configuration file is fairly complex. In general, the first part of the file defines a number of <filesystem> elements that define which paths to monitor. Note that the term “file system” here is purely logical and scoped within the monitoring configuration – there is no explicit or implicit link to local real file systems or to other grid container paths. Each <filesystem> element defines a name for each file system to be monitored (used in later elements), defines the local path (to the grid container) of the file system and describes properties of the file system. The name and path definitions are mandatory, but the other elements are optional and are there to provide a hook for later unifying the concept of file systems across a grid container. The rest of the file is a series of <filesystem-watcher> elements. A <filesystem-watcher> element defines which file system is being watched (based on the name given in the previous <filesystem> element), how often to check conditions, and a <filter-expression> element which defines what is considered out of bounds for the file system. This expression can use comparison operators <less>, <less-equals>, <equals>, <not-equals>, <greater>,<greater-equal> to compare literal numeric values and/or the value of an of the six supported variables related to a file system’s storage space. The six currently supported variables are:

- spaceAvailable: absolute amount of currently available storage space in file system in number [+ unit] format;
- spaceUsed: absolute amount of currently used storage space in file system in number [+ unit] format;
- percentAvailable: amount of currently available storage space in file system as percent of total file system size;
- percentUsed: amount of currently used storage space in file system as percent of total file system size;
- filesystemSize: total amount of storage space in file system in number [+ unit] format;
- spaceUsable: absolute amount of currently usable storage space in file system in number [+ unit] format;

Units for storage amounts are TB, GB, MB, KB, and B.

More complex expressions can be created using <not>, <xor>, <or>, <true>, <false>, and <and> boolean logic operators.

Following the <filter-expression> are an arbitrary number of <watch-callback> elements – each defining an action to take when conditions become out of bounds or later recover. The <watch-
callback> elements define how many times the action should be taken once an out of bounds condition is detected (e.g. call-limit="1"), whether to perform any action if/when normal conditions return (register-anti-callback="true"), and what action to perform (as defined by a grid container Java class that performs the action, e.g. class="edu.virginia.vcgr.genii.client.filesystems.log.LogWatchHandler"). Each action has its own additional configuration information and format. Note that a single file watcher can take multiple actions of multiple types. Currently there are two actions possible.

K.5.2.4.1. Action: Write Log Message


Configuration: <log-watch-config> element contains <message> and <level> elements. <message> defines the message to be written to the grid container log file. The message may take two positional wild cards that will be macro substituted during output from values related to the current status of the file system. The first is referenced using %s and will macro substitute the name of the file system (from the <filesystem> element); the second is referenced using %.2f% and will substitute the percent that the file system is full. The <level> element defines at what level of severity the log message should be made (Trace, Warn, Fatal, Error, Info, Debug – the Log4j severity levels).

K.5.2.4.2. Action: Send Email

Class: edu.virginia.vcgr.genii.client.filesystems.email.EmailWatchHandler.

XML Schema:

```xml
<email-watch-config>: defines configuration for email action
<message>: body of the message to be sent when a problem condition is reported
<positive-message>: body of the message to be sent when a good condition is reported
<connection>: connection info for outgoing mail server.
isSSL attribute: SSL toggle for communication to outgoing mail server (true/false).
<smtp-server>: IP address or DNS name of mail server.
<port>: mail server port.
$username>: username for authenticating to mail server
$password>: password for user at mail server
<debug-on>: toggles debugging (true/false)
<addressing-info>: email address information – both sender and receiver(s).
<addressTo>: email address of a recipient. This element can be specified multiple times to send to multiple recipients.
<addressFrom>: email address of message sender.
<subject>: email subject line.
```
Example 1: Log message when absolute space below 10 GB on "local" file system
<filesystems xmlns="http://vcgr.cs.virginia.edu/filesystems">
  <filesystem name="local" path="/home/user/.genesisII-2.0">
    <filesystem-property>Local</filesystem-property>
    <filesystem-sandbox name="database" relative-path="derby-db"/>
  </filesystem>
  <filesystem-watcher check-period="1 minute" filesystem-name="local">
    <filter-expression>
      <less xmlns="http://vcgr.cs.virginia.edu/filesystems/filter-script">
        <variable name="spaceUsable"/>
        <literal value="10 GB"/>
      </less>
    </filter-expression>
    <watch-callback call-limit="1" register-anti-callback="true">
      class="edu.virginia.vcgr.genii.client.filesystems.log.LogWatchHandler">
        <log-watch-config xmlns="http://vcgr.cs.virginia.edu/filesystems/log-watch">
          <message>Filesystem "%s" is %.2f%% full</message>
          <level>Warn</level>
        </log-watch-config>
      </watch-callback>
    </filesystem-watcher>
  </filesystem>
</filesystems>

Example 2: Send email when absolute space below 10 GB or below 5% on "shared" file system
<filesystems xmlns="http://vcgr.cs.virginia.edu/filesystems">
  <filesystem name="shared" path="/home/user/shared-directory">
    <filesystem-property>Shared</filesystem-property>
    <filesystem-sandbox name="scratch" relative-path="scratch-dir"/>
  </filesystem>
  <filesystem-watcher check-period="1 minute" filesystem-name="shared">
    <filter-expression>
      <or xmlns="http://vcgr.cs.virginia.edu/filesystems/filter-script">
        <less>
          <variable name="spaceUsable"/>
          <literal value="10 GB"/>
        </less>
      </or>
    </filter-expression>
  </filesystem-watcher>
</filesystems>
Optional Configuration Topics

K.5.2.5. Configure HTTPS/SSL
The property `edu.virginia.vcgr.genii.container.listen-port.use-ssl` in the
`INSTALL_DIR/deployments/<deploy_name>/configuration/web-container.properties` defines
whether the grid container is to use HTTPS/SSL for communication. The value of true turns it on
and false turns it off.

The rest of the SSL configuration is done by properties in the `INSTALL_DIR/deployments/<deploy_name>/configuration/security.properties` file as follows:

### K.5.2.6. Trust Store for incoming SSL connections

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.trust-store-location:</code></td>
<td>name of trust store (in directory <code>INSTALL_DIR/deployments/&lt;deploy name&gt;/security/</code>) to use to verify trust of incoming SSL connections.</td>
</tr>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.trust-store-type:</code></td>
<td>type of trust store specified in <code>trust-store-location</code>. Types include PKCS12, JKS and WIN.</td>
</tr>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.trust-store-password:</code></td>
<td>password container should use to access SSL trust store.</td>
</tr>
</tbody>
</table>

### K.5.2.7. Key Store for outgoing SSL connections

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.key-store:</code></td>
<td>name of key store (in directory <code>INSTALL_DIR/deployments/&lt;deploy name&gt;/security/</code>) to use to obtain SSL certificate for outgoing grid container connections.</td>
</tr>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.key-store-type:</code></td>
<td>type of key store specified in <code>key-store</code>. Types include PKCS12, JKS and WIN.</td>
</tr>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.key-store-password:</code></td>
<td>password for SSL key store specified in <code>key-store property</code>.</td>
</tr>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.key-password:</code></td>
<td>password for certificate within key store specified in <code>key-store property</code>.</td>
</tr>
</tbody>
</table>

### K.5.2.7.1. Installer defaults for the XCG Installer

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.trust-store-location=trusted.pfx.</code></td>
<td>The XCG trusted.pfx trust store is included in the XCG installer. It contains the grid-wide container group CA certificate (public key part) which XCG grid containers are configured to use to generate their grid container certificates, which in turn are used by default by grid containers for outgoing SSL connections:</td>
</tr>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.trust-store-type=PKCS12;</code></td>
<td></td>
</tr>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.trust-store-password=***;</code></td>
<td></td>
</tr>
<tr>
<td><code>edu.virginia.vcgr.genii.client.security.ssl.key-store=container.pfx.</code></td>
<td>The container.pfx key store is generated by the XCG installer during first time setup of the grid container. This is the grid container certificate generated by calling the <code>/etc/ContainerGroup CertificateGenerator</code> resource to create/sign a unique certificate for the container. The certificate is</td>
</tr>
</tbody>
</table>
created with the grid-wide container group CA certificate in its certificate chain and so this will match the trust chain in the default SSL trust store (trusted.pfx) for XCG;

```
edu.virginia.vcgr.genii.client.security.ssl.key-store-type=PKCS12;
edu.virginia.vcgr.genii.client.security.ssl.key-store-password=***;
edu.virginia.vcgr.genii.client.security.ssl.key-store-password=***;
```

### K.5.2.8. Changing Grid Container Trust Store

**Outdated section**

Grid containers have two trust stores – one used for SSL connection verification (the grid container SSL trust store) and one used for everything else (simply the grid container trust store). The grid container trust store contains the certificates of CAs that the container trusts to mint certificates used for client authorization, for signing messages (signing done outside of SSL), etc. The INSTALL_DIR/deployments/<deploy_name>/configuration/security.properties file defines the trust store to use for a grid container.

```
edu.virginia.vcgr.genii.client.security.resource-identity.trust-store-location:
  file name of trust store within directory
  INSTALL_DIR/deployments/<deploy_name>/security.
edu.virginia.vcgr.genii.client.security.resource-identity.trust-store-type:
  type of trust store (e.g. PKCS12)
edu.virginia.vcgr.genii.client.security.resource-identity.trust-store-password:
  trust store password
```

It is possible to replace a grid container’s trust store. However, given the difficulty in coordinating managing trust relationships throughout a large grid system, changing a grid container’s trust store must be done very carefully.

To change a trust store, one simply needs to replace the existing trust store within the INSTALL_DIR/deployments/<deploy_name>/security/ directory. Alternatively, a new trust store can be placed in that directory and the three properties detailed above modified in the INSTALL_DIR/deployments/<deploy_name>/configuration/security.properties file.

### K.5.2.8.1. Installer defaults for the XCG Installer

Installs an XCG-wide trust store (trusted.pfx) that contains a special XCG CA certificate used to mint all certificates in the system and a number of well known CAs from industry (such as Verisign) and other deployed grid systems. The security.properties is set to point to the installed trusted.pfx file with the appropriate type (PKCS12) and password.

### Unlikely Configuration Topics

#### K.5.2.9. Specify Grid Container’s Deployment
The `edu.virginia.vcgr.genii.container.deployment-name` property in the `INSTALL_DIR/container.properties` file defines the deployment to use. The value of this property defines the name of deployment for the grid container.

The `<deployment-conf based-on="parent-deployment"/>` element in the `INSTALL_DIR/deployments/<deploy name>/deployment-conf.xml` file specifies whether a deployment is "based on" (inherits from) another deployment. The value of the "based-on" attribute defines the name of the parent deployment.

### K.6. Post Installation: Setting Grid Container Service Permissions

By default, only the grid container admin certificate and certificates listed in the default-owners directory (`deployments/<deploy_name>/security/default-owners`) at the time of first startup can access the grid services of a new grid container. If that is acceptable, then there is no requirement that additional steps be taken. However, that is not usually the case and some additional setup of permissions is required.

Of course, there is no set of rules for setting permissions for a grid container – it all depends on the grid container’s purpose and the overall security framework for a given grid system. Since this document is concerned with grid container configuration and not user-level grid resources, this section only covers permissions related to the grid services on the container. Permissions can be set through normal Genesis II ACLs tools using the links provided by the `VCGRContainerPortType's` resource–fork. For example, one can access the `RandomByteIOPortType` service via through the `Services/RandomByteIOPortType` relative path.

#### K.6.1. VCGRContainerPortType Service

**Read:** add users who should be able to view pieces of the grid container’s status and managed grid resources (does not give read permission into the grid resources themselves). Consider giving permission to all users, to other administrators in the grid system, or to groups of users who have some interest in the container

**Write/Execute:** Only give permission to those considered administrators of the grid container to perform management operations.

#### K.6.2. Other services

In general, it is tricky to set permissions on a service by service basis because some services depend on other services. For example, listing an RNS directory (part of the `EnhancedRNSPortType`) or listing jobs statuses (part of the `QueuePortType`) relies on the ability to create an iterator (`IteratorPortType`) for the results. Another example is that grid queues (`QueuePortType`) use subscriptions/notifications (`GeniiSubscriptionPortType, GeniiPublisherPortType`) to obtain information about job state changes from BESs.
Because of these dependencies, it is often easier to allow broad permissions to all services (for example via a grid-wide user group) and then to remove permissions to specific services that are desired to be turned off. Creating resources of its type is designated as an “execute” level operation for a grid service, so this boils down to setting/removing execute permissions.

**K.6.3. Tips**

In general, read permission is usually innocuous (there are no known serious information or security leaks identified at this time through read permissions) and it is often more hassle than it is worth to try to restrict read access to services.

In general, write permission should be restricted to only those considered to be administrators of the grid container.

- **To turn off creating RandomByteIOP Files:** remove execute permission on RandomByteIOPortType service.

**N.B.** It is NOT advisable to remove permission on StreamableByteIOPortType, because it is used by other services/resources (e.g. code that implements resource forks which export generates data that looks like a file)

- **To turn off creating users:** remove execute permission on X509AuthnPortType service.
- **To turn off creating BESes:** remove execute permission on GeniiBESPortType service.
- **To turn off creating exported file system directories:** remove execute permission on ExportedRootPortType and LightWeightExportPortType services.
- **To turn off creating grid queues:** remove execute permission on QueuePortType service.

**K.7. Creating and Configuring Genesis II BES Resources**

Genesis II Basic Execution Service grid resources can optionally be created on any grid container - in fact, a single grid container can host multiple BES grid resources. The job of a BES resource is to manage executing user jobs on some physical computing resources. The Basic Execution Service specification defines what all BES resources must do, i.e. what operations a BES must support, including their semantics and fault reporting obligations. For example, all BES resources must support createActivity\(^1\), terminateActivity, and getActivityStatuses operations to submit, delete, and check job status respectively. The BES specification also defines certain aspects of the computation model for jobs. For example, BES resources must supply a working directory for each job they accept, must follow a particular state transition model for processing jobs, may optionally provide scratch space for jobs, etc.

---

1 BES uses the term “activity” to describe an instance of a job managed by a BES resource.
In order to submit a job, it must be described using the Job Submission Description Language (JSDL) – another widely-used, though separate community standard. It is the JSDL that is submitted to a BES resource to request running a job (via createActivity).

JSDL describes a wide range of information about a job:

- what to execute, e.g. the executable and arguments for a POSIX-like application;
- properties required by the job, e.g. OS, architecture, memory, number of processors, wallclock time, etc.;
- directives to stage data to the job’s working directory or copy data from the job’s working directory

What the BES specification does not define is how each BES must carry out its operations. In Genesis II, all BES resources follow the BES specification for basic operations. Genesis II BES resources do contain extensions to the BES interface for its containers, but care has been taken to ensure that Genesis II software does not rely on these not-yet-standard extensions. This enables Genesis II software to interoperate with BES resources developed by other grid software products. Genesis II provides several BES flavors -- each designed to handle a particular type of local computation environment.

Fork/Exec flavor: The Genesis II fork/exec BES flavor executes jobs directly on the same machine as the grid container using a fork/exec mechanism or its equivalent on Windows.

Native Queue flavor: The Genesis II native queue BES flavor executes jobs by submitting jobs to a backend queuing system which runs the job on one of the compute resources that it encompasses. Genesis II currently supports Portable Batch System (PBS) and Sun Grid Engine (SGE) native queuing systems. Additional queuing systems can be supported either via Genesis II code changes or in some cases by writing scripts that translate PBS or SGE queue submission (qsub), queue delete (qdel), and queue status (qstat) commands to the target queuing system commands. In essence a new queuing system type can be supported by writing commands that make it behave like a supported queue type.

### K.7.1. BES Resource Attributes and Matching Parameters

All Genesis resources can export information about their state through attributes and there are well-known interfaces and tools for updating and retrieving them. Genesis II BES resources export a number of attributes specified by the BES standard as well as a number of additional Genesis II-specific attributes. Below is a list of BES-specific attributes exported by Genesis II BES resources and, where appropriate, the JSDL to which it matches. The BES Configuration section describes these attributes in more detail, including defaults and ways to set/override their values.

- **CPUCount**: number of total CPUs managed by a BES resource.
  - Matches JSDL NumberOfProcesses in SPMDApplication element.
- **OperatingSystem**: OS of compute nodes.
  - OperatingSystemName sub-element matches JSDL OperatingSystemName element.
- **CPUArchitecture**: machine architecture
o CPUArchitectureName sub-element matches JSDL CPUArchitectureName element.

- **IsAcceptingNewActivities**: is BES accepting new jobs currently?
  o BES rejects jobs if value is false.

- **VirtualMemory**: Virtual memory for entire BES resource.
  o Matches JSDL TotalVirtualMemory.

- **PhysicalMemory**: Physical memory for entire BES resource.
  o Matches JSDL TotalPhysicalMemory.

- **IndividualCPUSpeed**: CPU clock speed.
  o Matches JSDL IndividualCPUSpeed.

- **WallclockTimeLimit**: Max wallclock time for job execution.
  o Matches Genesis II JSDL extension WallclockTime.

- **TotalNumberOfActivities**: total activities currently managed by BES.
  o Maintained internally by BES resource.

Genesis II also supports **matching parameters**. Matching parameters are name-value pairs much like attributes, but they can also express whether the topic they embody is supported or required. Genesis II JSDL extensions can specify what matching parameter name/values a job requires of a BES resource and can similarly express what features the job can support. Similarly Genesis II BES resources can express what they support and what they require. In this way, the matching process is bi-directional.

Some examples:

- Job requires that a BES resource supports a particular application, say blastp, which is represented by the matching parameter name/value blastp=true:
  o JSDL defines matching parameter requires: blastp=true
  o BES resource defines matching parameter supports: blastp=true

- BES resource requires statically-linked executables (think Cray xTMM5), represented by the matching parameter name/value statically_linked=true:
  o JSDL defines matching parameter supports: statically_linked=true
  o BES resource defines matching parameter requires: statically_linked=true

**K.7.2. Creating a BES Resource**

Genesis II BES resources are created using the create-resource command (part of the Genesis II grid shell tool). The syntax of the command is as follows (some options omitted for brevity/clarity):

```
create-resource
    [--construction-properties=<construction-properties file>]
    <grid container service path>
    [<global namespace path for new BES resource>]
```

The only mandatory argument is the `<grid container service path>` which specifies not only what grid container should create and host the resource, but also what type of resource should be created (by specifying the grid container service that will create it). For BES resources, the proper
service is GeniiBESPortType, regardless of the flavor of BES. In other words, there is one BES grid service in Genesis II, which can be configured to be different flavors. Grid containers expose their contained services via the Services/ directory within their resource fork. In other words, if there is a grid container referenced by the global namespace at path /containers/Container1, then the grid container’s services have references in the directory /containers/Container1/Services and the BES grid service is at path /containers/Container1/Services/GeniiBESPortType.

The --construction-properties option allows the creator to pass in initial properties for the newly created resource. The format and semantics of these properties are specific to each type of resource and some types of services do not support any construction-properties. The BES grid service is one that accepts construction properties in the form of an XML document with the appropriate schema.

The third argument, <global namespace path for new BES resource> specifies where the newly created grid resource should be linked into the global namespace. While this argument is optional, it really is a good idea to use it every time. Failure to do so will result in a grid resource that may be difficult to reference and therefore communicate with.

K.7.3. Configuring Genesis II BES Resources

Configuration of a specific BES resource is primarily done through its BES construction-properties. Unlike the files used to configure a grid container, construction-properties are not stored in the local Genesis II installation or deployments directory – in fact they are not stored as an independent file in the file system at all. Rather, construction-properties are specified as an argument during the BES resource creation and are stored with the state of the BES resource in the grid container database. The contents of a BES resource’s construction properties may be viewed and edited via the Genesis II global namespace. Each BES resource exposes its construction-properties in its resource fork (named construction-properties) which supports the Genesis II random byte IO file interface. Editing the contents of the construction-properties byte IO file changes the configuration of the BES resource effective immediately.

Some grid container configuration heavily influences the behavior of BES resources it contains. For example, a grid container’s scratch space configuration determines where scratch space is rooted for all BES resources on the grid container.

A number of BES resource configuration options are common across currently available flavors, while others are specific to a particular flavor.

K.7.3.1. Cross BES Flavor Configuration Options

Root directory for job working directories. Defines the root directory where the BES resource should create subdirectories for individual jobs run on the BES. The directory must be accessible from the machine hosting the BES resource and have read/write/execute permissions for account running the grid container. The grid container configuration file INSTALL_DIR/deployments/<deploy_name>/configuration/server-config.xml contains a property element (within configuration/global-properties element) with name edu.virginia.vcgr.genii.container.bes.worker-dir whose value defines the grid
container’s default root job working directory. The `edu.virginia.vcgr.genii.container.bes.worker-dir.allow-override property (true/false)` defines whether this path can be overridden by individual BES resources. For most installations, the default is set to `GRID_CONTAINER_STATE_DIR/bes-activities`.

**Fork/Exec flavor:** The means for launching the job.

**Native queue flavor:** The path is overridden in the BES resource’s construction-properties. The `<nativeq>` element contains a shared-directory attribute which specifies the root directory where the root for job working directories is to reside. Below the root shared directory, the native queue BES resource creates a “bes” directory as the root for job working directories.

**Scratch space root directory.** Defines the root directory where the BES resource should manage its scratch space. The directory must be accessible from the machine hosting the BES resource and have read/write/execute permissions for account running the grid container. Note that the definition of the scratch space root directory is done at the level of the entire grid container and is therefore shared among all BES resources on the grid container. It is included in this section because it is an important consideration when creating a BES resource. The grid container configuration section “Configure Scratch Space Service and Directory” details how to change the scratch space directory for a grid container. Default is `GRID_CONTAINER_STATE_DIR/scratch-space`.

**User activity policy.** Genesis II grid containers can detect if a user is logged in or if the screen saver is active on its host machine. Each BES can have a policy as to what to do to existing jobs when each type of user activity is detected. When user activity is detected, it can be ignored (NOACTION, the default), or jobs can be suspended (SUSPEND), suspended then killed (SUSPENDORKILL), or simply killed (KILL).

N.B. The screen saver option is a possible future enhancement and is not currently functional.

N.B. While technically available to all Genesis II BES flavors, the user activity policy really only makes sense for the Fork/Exec flavor because the other flavors are usually executing jobs on machines other than the grid container host machine.

**Pre- and Post-Execution Delay.** Each Genesis II BES can enforce a time delay just before running job and just after detecting that a job finished. The purpose of this delay is to avoid any well-known timing problems surrounding the file systems used to stage data into or out of the job. For example, when using a native queue BES, job working directories and scratch space are on shared file systems with both the grid container and one or more other compute machines reading/writing data to/from this file system. Some file systems have well-known (or not so well-known) coherency windows where reads/writes/metadata will not be immediately reflected to all machines that can access the file system. So, it can happen that the BES writes out a staged-in file, starts the job, and the machine on which the job runs has out of date file system metadata and reports that the file does not exist. The same thing can happen in reverse when a job writes its final output to its working directory, ends, and the BES detects the job completion and starts to do stage out operations form the job’s working directory.

The default is no pre or post delay, but we often use 15 second delays to avoid fault errors. Pre- and post-execution delays are specified in a BES resource’s construction properties, in the `<pre-execution-delay>` and `<post-execution-delay>` elements respectively.

**Resource Properties.** By default, several properties about a Genesis II BES resource are automatically detected by the BES resource implementation. In some cases the auto-detection process is accurate and appropriate, in others it is not. All of these values can be overridden in a BES resource’s construction properties document, within in the `<resource-overrides>` element.

**CPU Count:** Total number of CPUs for all compute resources encompassed by a BES resource. By default a BES resource detects the number of CPUs on the grid container’s host machine. Override element: `<cpu-count>` element.
Physical Memory. Total aggregate physical memory across all compute resources encompassed by a BES resource. By default a BES resource detects the size of physical memory on the grid container’s host machine. Override element: <physical-memory>.

Virtual Memory. Total aggregate virtual memory across all compute resources encompassed by a BES resource. By default a BES resource detects the size of virtual memory on the grid container’s host machine. Override element: <virtual-memory>.

Operating System Name. Name of compute node operating system (defined by JSDL enumeration). By default a BES resource detects the OS name of the grid container’s host machine. Override element: <operating-system-name>.


CPU Architecture Name. Name of compute node CPU architecture (defined by JSDL enumeration). By default a BES resource detects the CPU architecture of the grid container’s host machine. Override element: <cpu-architecture-name>.

CPU Speed. CPU speed for compute resources encompassed by BES. By default a BES resource detects the CPU speed of the grid container’s host machine. Override element: <cpu-speed>.

Maximum Wallclock Time. A BES resource can be configured to advertise a maximum wallclock execution time allowed for a job. By default, there is no wallclock time limit, no BES resource wallclock time attribute is advertised, and the assumption is that wallclock execution time is unlimited. The default can be overridden a BES resource’s construction properties, in the <resource-overrides>/<wallclock-time-limit> element.

Note that this override does not instruct the BES resource to enforce such a limit – it merely instructs the BES resource to advertise that it has such a limit which presumably is enforced in some other way (such as through the native queue with which it interfaces).

K.7.3.2. Native Queue Flavor Configuration

Native queue BES resources are designed to execute and manage grid jobs via a queuing system that exists outside the Genesis II grid software. Native queue BES resources are created in the same way as other Genesis II BES resources – via the create-resource grid shell tool command, though their configuration can be a bit more complex than a fork/exec BES resource. Care must be taken to choose both a root job working directory and a scratch root directory that are accessible to both the grid container machine/account as well as all of the compute nodes on which jobs may be executed.

To configure a Genesis II BES resource to be a native queue flavor, you must add the <nativeq> element to the BES resource’s construction-properties document. This tells the Genesis II BES service (GeniiBESPortType), that it should behave like a native queue flavor instead of a fork/exec or other flavor. The <nativeq> element contains a number of attributes and sub-elements to define specific configure information.

The implementation of a Genesis II native queue BES resource relies on using native submit, status and delete commands for the target queuing system and must find these commands with the grid container’s path. For both PBS and SGE, this means that the native queue BES implementation will look for the qsub (submit), qstat (status) and qdel (delete) commands using the grid container’s path environment variable. If those programs are not in the grid container’s path or if different
program names/paths are to be used, the proper program path must be overridden in the BES resource’s construction properties.

Below are detailed descriptions of native queue specific configuration issue/options. Please refer to the Cross BES Flavor Configuration Options section above for common configuration options and the example construction-properties document in Figure X.

**Native queue provider.** Defines which native queuing system is being used. Currently, Genesis II supports two native queue providers – PBS and SGE. The provider attribute in the <nativeq> element specifies the type of native queuing system. Values are pbs and sge. This is mandatory.

**Native queue name.** Defines the name of the queue to which the BES should submit jobs. This name will be included in properly formatted native queue commands and/or submit scripts to start and manage jobs. This is defined in the queue-name attribute of the <nativeq> element in the BES’s construction-properties document.

**Submit program path override.** Defines the path to the submit program to be used by the native queue BES resource. The program specified must have the identical interface (both input and output) as the expected program for the type of native queue being used (i.e. the submit program must have the same interface as the PBS qsub command for “pbs” native queues). To specify a new submit command, add the <nativeq>/<pbs-configuration>/qsub element to BES resource’s construction properties document and set the path attribute to the full path of the executable to use. Make sure the file has execute permissions sets for the grid container user account.

**Delete program path override.** Defines the path to the delete program to be used by the native queue BES resource. The program specified must have the identical interface (both input and output) as the expected program for the type of native queue being used (i.e. the delete program must have the same interface as the PBS qdel command for “pbs” native queues). To specify a new delete command, add the <nativeq>/<pbs-configuration>/qdel element to BES resource’s construction properties document.

**Status program path override.** Defines the path to the status program to be used by the native queue BES resource. The program specified must have the identical interface (both input and output) as the expected program for the type of native queue being used (i.e. the submit program must have the same interface as the PBS qstat command for “pbs” native queues). To specify a new status command, add the <nativeq>/<pbs-configuration>/qstat element to BES resource’s construction properties document.

**Queue program binary directory override.** Overrides directory in which to find submit, delete and status programs. Set using the bin-directory attribute of the <nativeq>/<pbs-configuration> element. The individual queue program overrides take precedence over this override. Therefore, if both this override and the individual command paths are overridden, the individual path is used.

**Root directory for job working directories.** Defines root directory where the BES resource should create subdirectories for individual jobs run on the BES. See “Cross BES Flavor Configuration Options” section for details.
Scratch space root directory. Defines the root directory where the BES resource should manage its scratch space. See "Cross BES Flavor Configuration Options" section for details.

K.7.3.3. PBS Native Queue Example

Figure 2 shows a portion of the construction-properties document for an example native queue BES resource. The presence of the nativeq element (Lines 1-9) signifies that the BES resource is a native queue flavor. The shared-directory attribute (Line 1) sets the path (/home/gbg/shared directory) for the root of the directory to be used by the BES resource for all things (except scratch space) that need to be shared with executing jobs – including job working directories. The provider attribute (Line 1) sets the type of the native queue to pbs. Line 4 specifies that the BES resource is to submit jobs to the PBS queue named “centurion” (queue-name="centurion").

Lines 7-9 show the elements used to override the paths to the submit, status, and delete commands, respectively. In this case, they are not to be overridden and will use the default for the pbs provider (i.e. qsub, qstat, and qdel in the grid container’s path). If they were to be overridden, the path of the command would be in the path attribute (e.g. <qsub path="/path/to/myQSub" />).

Lines 10-11 set values for the pre- and post-execution delays to 15 seconds.

Lines 12-14 override default values for cpu-count and physical-memory. In this case the reason is that the grid container host machine is not of the same type as the compute nodes.

<table>
<thead>
<tr>
<th>Example of a construction-properties file</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.  &lt;ns3:nativeq shared-directory=&quot;/home/gbg/shared-directory&quot; provider=&quot;pbs&quot;&gt;</td>
</tr>
<tr>
<td>2.   &lt;ns5:pbs-configuration xmlns=&quot;&quot;</td>
</tr>
<tr>
<td>3.     xmlns:ns7=&quot;<a href="http://vcgr.cs.virginia.edu/native-queue">http://vcgr.cs.virginia.edu/native-queue</a>&quot;</td>
</tr>
<tr>
<td>4.     queue-name=&quot;centurion&quot;&gt;</td>
</tr>
<tr>
<td>5.   <a href="">ns7:qsub/</a></td>
</tr>
<tr>
<td>6.   <a href="">ns7:qstat/</a></td>
</tr>
<tr>
<td>7.   <a href="">ns7:qdel/</a></td>
</tr>
<tr>
<td>8.   &lt;/ns5:pbs-configuration&gt;</td>
</tr>
<tr>
<td>9.   &lt;/ns3:nativeq&gt;</td>
</tr>
<tr>
<td>10.  <a href="">ns3:post-execution-delay</a>15.000000 Seconds&lt;/ns3:post-execution-delay&gt;</td>
</tr>
<tr>
<td>11.  <a href="">ns3:pre-execution-delay</a>15.000000 Seconds&lt;/ns3:pre-execution-delay&gt;</td>
</tr>
<tr>
<td>12.  <a href="">ns3:resource-overrides</a></td>
</tr>
<tr>
<td>13.   <a href="">ns3:cpu-count</a>2&lt;/ns3:cpu-count&gt;</td>
</tr>
<tr>
<td>14.   <a href="">ns3:physical-memory</a>2060000000.000000 B&lt;/ns3:physical-memory&gt;</td>
</tr>
<tr>
<td>15.   &lt;/ns3:resource-overrides&gt;</td>
</tr>
</tbody>
</table>

K.7.4. Genesis II BES Resource Security

K.7.4.1. BES Resource Access Control List
When a Genesis II BES resource is created using the create-resource command (or any other type of Genesis II resource for that matter), it acquires an initial access control set based on the credentials passed in during the create-resource request. Each credential passed in is added to each of the Read, Write, and Execute (RWX) permissions for the new BES resource. For example, say that I login to my Genesis II user resource (jfk3w) and acquire both a user X.509 credential (jfk3w) and a group credential (uva-group). My create-resource call will actually pass three credentials (the third is a credential made for client tools to establish SSH communication, which we call a “Client Cert” – but that’s a story for another time…). All three of these credentials will be added to the R,W, and X access control lists for the new BES resource. In addition, the service that creates the BES resource (the GeniiBESPortType service on the host grid container) adds its credential to the call chain and gets included in the access control lists. This default permission scheme gets the job done to bootstrap things, but is far from perfect.

Client Cert really isn’t needed since it is short lived and really isn’t what should identify who has permission. So, if you wish, you can remove it from the ACLs of the new resource to clean up the ACLs a bit – though it is unlikely to do any real harm to leave it in. After creating the BES you will need to decide what exact permissions you wish to set. In particular, you need to be careful not to allow write permission, and perhaps execute as well, to too wide of a group of users. Once decided, either the client-ui GUI tool (Security tab of new BES resource) or the grid shell Genesis II chmod command can be used to set ACLs to the desired values. In particular, you need to be careful not to allow write permission to too wide of a group of users.

Genesis II BES resources add another layer of access control on top of the normal Genesis resource ACLs. Genesis II BES resources restrict which activities can be manipulated or viewed by a particular caller. When an activity is created, a Genesis II BES resource remembers most of the credentials used to start the activity – only excluding those credentials with the word “group” in it (this is bit of a hack and a better system is in development for identifying which credentials should have permission for newly created resources). In the end, only callers who present a credential that matches one of the original creator’s credentials (plus the admin credential defined for the grid container) can perform read or execute operations on a particular activity. In this way, a user cannot manipulate (or see) another user’s activity.

<table>
<thead>
<tr>
<th>Definition of R, W, X for Genesis II BES Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read:</strong></td>
</tr>
<tr>
<td>• getActivityStatuses: retrieve status information for a list of activities</td>
</tr>
<tr>
<td>• getActivityDocuments: retrieve submission details for list of activities</td>
</tr>
<tr>
<td><strong>Write</strong></td>
</tr>
<tr>
<td>• addMatchingParameters: add matching parameter definition to BES resource</td>
</tr>
<tr>
<td>• stopAcceptingNewActivities: management function to instruct BES resource to stop accepting jobs until further notice.</td>
</tr>
<tr>
<td><strong>Execute:</strong></td>
</tr>
<tr>
<td>• createActivity: submit job/activity</td>
</tr>
</tbody>
</table>
K.7.4.2. Genesis II BES Activity Security

Currently, Genesis II BES resources run all jobs using the account of the grid container (i.e., the user account under which the grid container’s application server runs). The good news is that because of this, Genesis II grid containers do not require any special root or administrator permissions for its account – thereby removing a significant hurdle at some sites. On the other hand, this has significant security implications both for the Genesis II grid container and for user jobs. Jobs running under the grid container account means that a job can access anything on the compute machine that the grid container account can access as well as anything that other jobs concurrently running can access. If a job can run on the same machine as the grid container (e.g., using a fork/exec BES or via a native queue that includes the grid container’s host machine), then a job can access/manipulate/destroy the grid container code, configuration and state. Whenever possible, it is desirable to avoid letting the Genesis II code, configuration, and state be visible to jobs to remove this threat. For example, one can deploy Genesis II code, configuration and state on a file system only visible to the grid container’s host machine and not allow jobs to run on that machine – such as through a native queuing system that does not include the grid container’s host machine as a compute target. Unfortunately, potential malicious interference between user jobs is currently unavoidable other than to only allow one job to run at a time on a container. Improved BES implementations that reduce or eliminate these problems are under design/development.

K.8. Creating and Configuring Genesis II Queue Resources

Genesis II Queue grid resources can be created on any grid container. A single grid container can host multiple queue resources. A queue stores jobs until they can be executed. A queue resource monitors one or more BES resources. When a queue finds an available BES that has the necessary properties (operating system, memory, etc.) to run some job in the queue, it submits the job to the BES.

Genesis II Queue resources are created using the create-resource command. The syntax of this command was described in the section on creating BES resources. The queue service is QueuePortType.

Each queue resource exposes its list of monitored BES resources through its resource fork named resources. For example, if /queues/queue1 is a queue resource, then you can see its list of BES resources by running:

```
ls /queues/queue1/resources
```

Then, if /bes-containers/BootstrapContainer is a BES resource, then you can add this BES to the queue by running:

```
lx /bes-containers/BootstrapContainer /queues/queue1/resources/bes1
```
In a queue, you can configure the number of "slots" of each BES resource. The number of slots is the maximum number of jobs that the queue will submit to the BES to run simultaneously. See the qconfigure command.

K.8.1. Creating a Hierarchy of Genesis II Queues

A Genesis II queue can submit jobs to another Genesis II queue (which can, in turn, submit those jobs to still more Genesis II queues). To set this up, the "parent" queue must recognize the "child" queue as a BES resource. A BES resource must have certain properties (operating system, memory, etc.), so these properties must be defined for the child queue.

To create a Genesis II queue with BES properties, write the property values in an XML file with the appropriate structure, and give the properties file to the create-resource command. For example, the Genesis II source tree contains a file named queue-example.xml. You can create a queue:

```
create-resource --construction-properties=file:queue-example.xml
            /containers/BootstrapContainer/Services/QueuePortType
            /queues/besQueue
```

Then, add BES resources to /queues/besQueue/resources.

Finally, add the child queue as a resource of the parent queue:

```
chmod /queues/besQueue +r -everyone
ln /queues/besQueue /queues/queue1/resources/besQueue
```

The chmod command is necessary to allow the parent queue (which runs by default with no privileges) to read the child queue’s properties.

You can use the qconfigure command to assign a number of slots to besQueue.


The root installation contains a mix of executable code (scripts, executables, etc.), supporting libraries/jar files/external software, configuration files and directories, license information, necessary certificates, and log files. The structure of a typical installation directory for a newly installed grid container is shown below (note that details are omitted for brevity) and brief descriptions of some of the more important components follows. More detailed descriptions are contained in the appropriate appendices.

```
Genesis II Installation Directory

$ ls $GENII_INSTALL_DIR
grid* jre/ uninstall*
    client-ui* container.properties grid-update* lib/ updates/
    client-ui.app/ deployments/ jar-desc.xml License.txt webapps/
    client-ui.desktop* ext/ jni-lib/ RELEASE
```

K.9.1. Configuration Directories/Files
The configuration for Genesis II grid containers is spread out among a number of files and subdirectories inside the installation directory.

container.properties contains a property to specify which deployment is to be used for the grid container. This is set by the installer during a full container installation to GeniiNetFullContainer. A system administrator should rarely if ever need to modify this file.

deployments contains most grid container configuration files. There is a sub-directory for each named deployment. The standard installation comes with 2 deployments – GeniiNetFullContainer and default.

Within a deployment directory there is an optional deployment-conf.xml file which contains directives about the deployment itself – such as whether the deployment inherits (is “based-on”) another deployment. There are also four main subdirectories within a deployment directory -- configuration, security, services, and secure-runnable.

The configuration directory contains a number of files and subdirectories with configuration information – usually in either a Java properties format or an XML format. There are a number of files in this directory (or subdirectories) that administrators may need/desire to change.

The security subdirectory contains security data for the grid container and grid tools -- such as necessary certificates and trust stores. For grid systems that have an installer designed to properly configure trust relationships for their grid containers, administrators will likely not need to modify anything in the security subdirectory. However, for administrators creating their own grid system with their own trust relationships this is where the proper certificates and trust stores need to be placed and pointed to by the proper configuration files.

Under the security directory is a default-owners subdirectory which contains a number of certificates. These certificates are used to set default access control for a grid resource during its creation only in the situation where no user security credentials are present. This is a rather arcane piece of the Genesis II grid container and mostly used by the grid container during certain container bootstrapping operations. Admins will likely never have to concern themselves with the contents of the default-owners directory.

The services directory contains configuration information for how individual grid container services are to perform their operations within the grid container. Installers/system administrators will rarely if ever need to examine or modify configuration in this subdirectory.

The secure-runnable subdirectory is where the built-in Genesis II patching system will place secure runnables. The patching system supports the ability to execute arbitrary Java code during a patch, but does so only for code that is properly signed by a trusted source. A trusted, signed Java jar file with a runnable Main program is called a secure runnable in Genesis II. The deployments/ <deploy_name>/security/secrun.cer file contains the trusted certificate for secure runnables.

The patching system places secure runnables into the secure-runnable subdirectory when the patching system determines that a patch should be done and each is run when a grid container or
grid client tools next runs (done by the ApplicationLauncher). For a grid container, patches are downloaded each time the grid container is restarted; for grid client tools, this occurs once per week. Once successfully run, a secure runnable is deleted (they are considered “run once” programs).

### K.9.2. Executable Code

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid</td>
<td>Grid is the main client command-line tool for Genesis II. It is a script that runs a java program that works somewhat like a UNIX shell. If no arguments are specified, it opens an interactive “grid shell” program that allows users to run Genesis II commands interactively from a command prompt. Alternatively, if a Genesis II grid command is specified as arguments to the grid script, it will execute the Genesis II grid command, display the results and exit. Virtually all Genesis II functionality can be performed by some command issued via the grid shell.</td>
</tr>
<tr>
<td>client-ui</td>
<td>A GUI interface for browsing the grid namespace and for performing a number of grid operations, such as looking at file contents, submitting jobs, editing BES configuration and much more. In order to run, you must have a properly configured windowing/graphical environment running on your machine. For LINUX and Mac OS X, Genesis II GUI tools (including client-ui) require that X Windows is properly running and configured.</td>
</tr>
<tr>
<td>GFFSCContainer</td>
<td>Executable script to start and stop a grid container. This script is generated by the Genesis II installer and contains paths hard-coded for the installation. The script is designed to work with UNIX-like init.d type commands – i.e. it supports start, stop, restart, and status command line arguments. The start and restart commands start a grid container and run it in a nohup/background mode so that the script can exit and its command shell go away without terminating the grid container.</td>
</tr>
</tbody>
</table>

The installer sets up everything needed for the ApplicationLauncher in the ext directory. This includes the ApplicationLauncher code (app-manager.jar), and files to configure the launcher to patch and start Genesis II grid containers and client tools (e.g. genii-container-application.properties and genii-client-application.properties).

**N.B.** A system administrator should rarely if ever need to modify these files except if the installation directory is to be relocated and paths need to be changed in configuration information.

### K.9.3. Log Files

<table>
<thead>
<tr>
<th>Log File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container.log</td>
<td>Contains log information output by grid container. By default, the log will grow to 100MB maximum, be saved to container.log.1 and a fresh log opened. Therefore, the maximum log size is currently 200MB. This will be a system administrator’s best friend in debugging a grid container’s behavior.</td>
</tr>
<tr>
<td>JavaServiceWrapper/wrapper/bin/wrapper.log</td>
<td>Contains log information for the JavaServiceWrapper. Log statements generated by the service wrapper itself (during startup/shutdown) as well as messages output to standard output or error by programs it starts</td>
</tr>
</tbody>
</table>
K.9.4.  Supporting Libraries, Jar Files and Other Software

Genesis II grid containers and client software rely on a wide range of software, developed both internally and externally to the Genesis II project. Several sub-directories within the Genesis II installation directory contain various support software components.

JavaServiceWrapper  The JavaServiceWrapper directory holds software that helps manage running the Java-based Genesis II grid container as a persistent service. The JavaServiceWrapper software wraps the grid container, monitors it, and in some cases it can restart a failed grid container. On Windows systems, it also wraps the grid container in such a way that it can be run as a Windows service – which enables Windows to automatically start grid containers during a Windows OS startup. For system administrators, the most important item is that the JavaServiceWrapper/wrapper/bin/wrapper.log file contains debugging information about grid container starts and stops and sometimes other valuable debugging information.

jre  This directory contains the Java runtime Environment used by Genesis II. The Genesis II interactive installer deploys its own complete Java runtime to execute Genesis II tools and the grid container. The installer accomplishes this by hard coding paths in relevant scripts to use the Genesis II version of the JRE for Genesis II commands/scripts. Because of this, there is no requirement to install Java when running the interactive installer. However, RPM and Debian packages do require a Java JRE to run. Section H.1 provides more details about Java in circumstances where it must be installed.

lib  The lib directory stores the Genesis II jar files and a few related Java properties files. System administrators generally should not need to modify anything in this directory. The built-in Genesis II patching system will modify the contents of this directory to apply Genesis II software updates.

ext  The ext directory contains "external" software required by Genesis II grid containers and client programs -- i.e. software developed outside of the Genesis II project. System administrators generally should not need to modify anything in this directory. The built-in Genesis II patching system will modify the contents of this directory to apply Genesis II software updates.


Example Grid State Directory

$  ls $GENII_USER_DIR
bes-activities/  derby-db/  sbyteio-forks/  user-config.xml
container-id.dat  download-tmp/  scratch-space/  user-context.xml
container-id.dat.lock  rbyteio-data/  timing.txt  user-transient.dat
It is rare that a system administrator will need to be concerned about the actual contents of the grid container state directory, but it is good to know what is there in case of a problem or the need to back up state, etc. Please note that the grid container state directory cannot be shared between multiple grid containers – a unique state directory is required for each GII grid container.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container-id.dat</td>
<td>As the name implies, this contains a unique id for the grid container.</td>
</tr>
<tr>
<td>derby-db</td>
<td>This directory is where the files for the grid container database reside. Genesis II uses an embedded database named Derby maintained by Apache. Obviously, it is a very bad idea to modify anything in this directory. On the other hand, making a copy of this directory (when the grid container is shut down) is how a system administrator can make a backup of the grid container database state.</td>
</tr>
<tr>
<td>rbyteio-data</td>
<td>This directory is where the contents of random byte IO file grid resources created on the grid container are stored. There are a number of sub-directories and sub-sub-directories with server-generated names to keep the number of files stored in a single directory to a reasonable number (it hurts performance on many file systems to have too many entries in a single directory). The grid container keeps a mapping in its database to the proper path for each random byte io grid resource it contains.</td>
</tr>
<tr>
<td>sbyteio-forks</td>
<td>This directory contains stream contents for streamable byte io grid resources created on the grid container.</td>
</tr>
<tr>
<td>user-config.xml</td>
<td>The Genesis II code directory can store multiple sets of configuration files each with a different purpose. Each of these sets is called a deployment configuration set and the user-config.xml file specifies which set to use for the grid container. The current installer comes with only a single deployment configuration set named GeniiNetFullContainer. In the future, this feature may be used to create other installers for specific purposes.</td>
</tr>
<tr>
<td>user-context.xml</td>
<td>Stores information to be used for outcalls from grid tools. This usually stores information related to a user’s session – such as credentials and other contextual information related to the client. However, the grid container uses this area as well.</td>
</tr>
<tr>
<td>user-transient.dat</td>
<td>Stores items that should not be publicly visible on the grid, such as full certificates with private keys.</td>
</tr>
</tbody>
</table>

### K.10.1. Grid Resource-Specific Storage

The grid container stores the state of many grid resources completely within the grid container's database. However, some grid resources require additional storage areas to perform their functions. Some of these are placed inside the grid container state directory, while others require user or system administrator configuration to tell the grid container the paths to the proper storage to use. Below is a list of grid resources that require additional storage areas (referenced by a description of what the grid resource does as well as its service name).
Random access file (RandomByteIOPortType): Genesis II grid containers do not store the contents of a random byte IO file in the grid container database. Instead, the grid container maintains a separate directory (named rbyteio-data) within the grid container state directory to store file contents. The rbyteio-data directory contains two levels of sub-directories with names dir.<#>/dir.<#> and stores the contents of a random byte IO file in file one of these directories in a file named rbyteio<big #>.dat. The mapping from random byte IO resource to the path that holds its contents is stored in the grid container database. The reason for the 2 level structure and the use of numerous directories is to spread out the potentially large number of random byte IO files such that no directory has a large number of files – a known performance problem with many file systems.

Streaming file/data (StreamableByteIOPortType): Genesis II grid containers do not store the contents of a streamable byte IO file in the grid container database. Instead, the grid container maintains a separate directory (named sbyteio-forks) within the grid container state directory to store streamable file contents.

Exported Directory (ExportedRootPortType and LightWeightExportPortType): Genesis II supports the notion of mapping a portion of a local file system into the global grid namespace. The idea is simple: a user chooses a rooted directory in their local file system and creates a grid resource (either of type ExportedRootPortType or LightWeightExportPortType) that acts as the grid proxy for accessing data in the target directory. This is referred to as exporting a directory into the grid. In order to do this, the user must find (or deploy) a Genesis II grid container that can access the target directory. In other words, the grid container must be on a machine that has the directory mounted and is running as a user account that has permission to do the desired operations on the target directory.

GeniiBESPortType: Basic Execution services are, not surprisingly, perhaps the most complex Genesis II grid resources to configure. First, there are several different flavors of Genesis II BES resources – each follows the same BES interface and functionality, but each executes jobs in a different execution environment and therefore requires different pieces of configuration information. All Genesis II BES resources also have some properties in common.

All support:

- Providing a unique working directory for each job;
- Providing an area for a “scratch” directory (as defined by JSDL). Data files may be staged into a scratch directory outside of a job’s unique working directory so that files can be reused across multiple jobs without having to stage the same file over and over again;
- Support stage-in of data files to the job’s current working directory and/or to the BES resource’s scratch directory;
- Support stage-out of data from job’s current working directory and/or to the BES resource’s scratch directory.

The bottom line is that thought should be given to the type of BES resource desired and the configuration it requires when choosing a machine to host the grid container in which it will reside. More detailed information on configuring all of the Genesis II BES flavors is in .
K.11. Cross-Campus Grid (XCG) Global Namespace

The XCG grid maintains a particular structure for its global namespace. Using a consistent structure makes it easy for users and scripts alike to know where to find certain resources within the grid. Certainly, an individual grid system can develop its own directory structure and conventions if the administrators choose to do so.

### XCG Directories

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/accounting</td>
<td>Information to support gathering of accounting information. In particular, the XCG stores a single entry (CS-MySQL) whose EPR is a URL pointing to the database used to store all XCG accounting data (which is an external DB).</td>
</tr>
<tr>
<td>/bes-containers</td>
<td>Contains links to all of the major BES resources in the XCG system. The directory is organized with sub-directories for each major organization/institution/project and with further sub-directories for relevant sub-divisions. The sub-directory structure maintained in the /bes-containers directory mirrors the sub-directory structure in the /containers RNS directory as much as possible. The leaves are links to the BES resources themselves. Naming for BES resources is usually chosen to include the type of queuing system used, the queue or cluster being submitted to (if a NativeQ flavor BES) and the DNS name of the machine hosting the grid container for the BES.</td>
</tr>
<tr>
<td>/containers</td>
<td>Contains links to all of the grid containers in the XCG system. The directory is organized with sub-directories for each major organization/institution/project and with further sub-directories for relevant sub-divisions. The sub-directory structure maintained in the /containers directory mirrors the sub-directory structure in the /bes-containers RNS directory as much as possible. The leaves are links to the VCGRContainerPortType service for the grid container. Naming for BES resources is usually chosen to be the DNS name of the machine hosting the grid container.</td>
</tr>
<tr>
<td>/etc</td>
<td>Used like /etc in Unix-like OSs. Currently only contains the XCG grid-wide container certificate generator (ContainerGroupCertGenerator).</td>
</tr>
<tr>
<td>/groups</td>
<td>Contains all general purpose groups maintained by the XCG grid admin. Users may create their own groups elsewhere. In XCG, the most important group is the uva-idp-group which contains all users created by the XCG admin (after a modest vetting process). The uva-idp-group is used to set appropriate permissions for most typical grid container services and many of the standard global namespace directories.</td>
</tr>
<tr>
<td>/home</td>
<td>Home directories for all Genesis II users created by the XCG admin. These are created when the user is created for use by the individual users. Each user has a separate subdirectory named with their username (e.g. /home/drake for user /users/drake). By default a user’s home directory is set with RWX permissions for the user and admin and R permissions only for the uva-idp-group.</td>
</tr>
<tr>
<td>/queues</td>
<td>Contains grid queues. In XCG, the main queue is named grid-queue.</td>
</tr>
</tbody>
</table>
/uninitialized-containers holds handles to new grid containers registered by the XCG installers. A grid admin usually removes entries from here once the container has been linked into a more appropriate location in the global namespace and other configuration tasks have been completed. There is a sub-directory named failed that is sometimes used by admin scripts to keep links to containers which encountered errors during configuration.

/users Contains an entry for each Genesis II user resource (IDP resource) created by the grid XCG admin. The user resources are linked here by username during user creation. As part of the interface for Genesis II user resources, groups can be linked to a user resource. This informs the login tool that it should attempt to also acquire these group credentials when logging in a particular user. By default only the individual user and admin is given R, W, or X permissions to the user resource.

K.12. Security in the Grid

K.12.1. Supported Security Tokens

Genesis II supports a number of security token formats, mechanisms to acquire them and/or set them into the proper location for Genesis II software to use, and mechanisms to specify them for use in grid resource ACLs. Currently, Genesis II supports username/password tokens (following the WS-Security specification), X.509 certificates, and X.509 based SAML-like Genesis II security tokens - called GAML tokens. In addition, Genesis II supports a special “everyone” token for access control lists which can be used to allow access to all requests -- including those that present no security tokens.

K.12.2. Genesis II User/Group Resources and GAML Tokens

Genesis II supports mechanisms to create Genesis II user and group identities and to acquire a Genesis II GAML security token based on these identities via username/password authentication. GAML security tokens, in turn, can be used on access control lists for any Genesis II grid resource.

Genesis II users and groups are created by a grid administrator (someone who has proper authority) by creating a new grid resource of type X509AuthnPortType, which requires a username and a password to be specified. The new X509AuthnPortType resource acts as a Secure Token Service (STS) following the OASIS WS-Trust specification. Its purpose is to generate GAML security tokens identifying the user or group – but only if the caller can authenticate using the proper username/password. The Genesis II grid shell login command provides the mechanism for a client to acquire (and store in their security context) a GAML security token from their Genesis II X509AuthnPortType resource by passing in the proper password.

Example:

```
login grid:/users/drake --username=drake --password=123
chmod /home/stan/myResource +r grid:/users/drake --password=123
```

K.12.2.1. Username/password
Genesis II supports username/password security tokens as per the WS-Security specifications. Username/passwords security tokens can be set for grid resource ACLs via either the client-ui for a resource (in the Security tab) or via the Genesis II chmod command (\texttt{--username=<username> --password=<password>}). The Genesis II grid shell command \texttt{passwordLogin} allows a user to create a username/password token and set it into their security context for use in subsequent Genesis II tools.

Example:

\begin{verbatim}
passwordLogin --username=fred --password=123
chmod /home/stan/myResource +r --username=fred --password=123
\end{verbatim}

\section*{K.12.2.2. \textit{X.509 Certificate}}

Genesis II can support X.509 security tokens minted outside of Genesis II – both as entries in a grid resource’s ACL and as security tokens passed in messages for authorization. The grid shell \texttt{keystoreLogin} command allows a user to add an X.509 certificate into a client’s security context from a key store. The user is prompted for password information as necessary to authenticate to the key store to acquire the X.509 certificate. To set grid resource ACLs using an X.509 certificate, the Genesis II \texttt{chmod} command accepts a path (local or RNS path) to a certificate file, the contents of which are extracted and used to set/unset the desired access control.

Example:

\begin{verbatim}
keystoreLogin local:/home/fred/certs/fred.pfx -password=keystorepassword
chmod /home/stan/myResource +r local:/home/fred/certs/fred.cer
\end{verbatim}

\section*{L. Appendix 3: XScript Language Reference}

\subsection*{L.1. Introduction – What is XScript?}

The XScript scripting language is an XML-based scripting language developed by the Global Bio Grid research group at the University of Virginia for use with the Genesis II grid software. Originally the language was designed to support only minimal capabilities – enough to get the project started until something better could be developed – but has grown into a more sophisticated and fully featured language in its own right.

Today, the XScript language supports many of the language features that one would expect from a programming language including loops, conditionals, and exceptions. While the language is still not officially supported, the need for documentation remains. To that end, this document serves as the language reference for the XScript 1.0 scripting language.

\subsection*{L.2. Namespaces}
The XScript language supports two namespaces for its XML elements. The first one is used for language elements and is abbreviated as gsh. The second indicates that the element is for a Genesis II grid command and is abbreviated geniix. We will use the first of these, gsh, as the default namespace for all XML in this document and thus assume that the root element of all XScript scripts looks like the one shown in the figure below.

```xml
<xml version="1.0" encoding="utf-8" ?>
<gsh:script
 xmlns:gsh="http://vcgr.cs.virginia.edu/genii/xsh/script"
 xmlns:geniix="http://vcgr.cs.virginia.edu/genii/xsh/grid"
 xmlns="http://vcgr.cs.virginia.edu/genii/xsh/script">
...
</gsh:script>
```

**L.3. Running XScript Scripts**

Before we delve into the language itself, a word about running XScript scripts is in order. Genesis II supports multiple scripting languages through the use of the Java Scripting API. In order to differentiate between the various scripting languages, Genesis II uses filename extensions to determine the correct language to use when running scripts. Thus, to run a JavaScript script, one would indicate a file whose filename ended in the .js extension. Similarly, to run an XScript script file, the filename must end with the .xml filename extension.

**L.4. XScript Variables/Macros**

Every attribute value and text content node of an XScript script can include a reference to a variable. If included, the value of this variable will be inserted at run time as a macro replacement. Further, variables are scoped by their statement level. This makes it possible to write scripts that contain multiple variables of the same name without additional variable definitions interfering with outer definitions.

Variables in XScript documents are indicated by surrounding the variable name with ${ and }. Thus, to indicate the value of the NAME variable, one would use the string ${NAME} anywhere that text was expected (such as for an attribute value or as the text content of an appropriate XScript statement).

Arrays are also supported in the XScript language, though at the time of the writing of this document, only for accessing passed in as parameters to either the script itself, or to functions. The length of an array in XScript is indicated with the ${ARRAY_VARIABLE} expression syntax, while the elements inside of the array are indicated with the ${ARRAY_VARIABLE[INDEX]} syntax. Thus, to echo all elements of the ARGUMENTS array, one might use the following XScript code.
Arguments passed in to the script as well as those passed into function are contained in the ARGV array variable (for command line arguments passed into the script, the first element is the name of the script file itself).

L.5. XScript High-level Description

In XScript, every XML element (other than the root document element) represents a single language statement. These statements may or may not themselves contain other statements depending on the element type in question. For the most part, those statements which can support inner statements are the language feature elements such as conditionals and loops, while those that cannot generally represent simple statement types like echoes, grid commands, and sleep statements. Further, in XScript, every XML elements falls into one of two categories. The first category represents the language features and these elements are always in the gsh namespace (which, recall, is the default namespace). The second category is the grid commands category. These elements are always in the geniix namespace and the name of the element must directly match a Genesis II grid command.

L.5.1. Grid Command Elements

The simplest form of statement in an XScript script is a grid command. Grid commands are identified by belonging to the geniix namespace. Any time an XML elements exists in this namespace, the XScript engine attempts to find a grid command with the same name as the element's local name. If it finds such a command, the statement is assumed to represent that command, otherwise an exception is thrown. Parameters (commandline arguments to the grid command) are indicated with XScript param elements described later in this document. Below we show example grid commands in the XScript language.

...<geniix:ls/>
<geniix:cp>
  <param>--local-src</param>
  <param>/etc/passwd</param>
  <param>/home/passwd</param>
</geniix:cp>
...

L.5.2. XScript Language Elements
The remainder of this document will present XScript language elements. For each element, a short description will be given followed by a table of all applicable element attributes (if any exist). Following this, a more detailed description will be given as well as an indication of whether or not nested statements are permitted.

**L.5.2.1. echo**

The echo statement is used to echo output to the console.

**L.5.2.1.1. Attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>yes</td>
<td>The text message to display to the console</td>
</tr>
</tbody>
</table>

The echo statement indicates that text should be printed to standard out of the scripting engine. This statement cannot contain other statements.

**L.5.2.2. define**

Defines a new variable and associated value.

**L.5.2.2.1. Attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>The variable to define.</td>
</tr>
<tr>
<td>source</td>
<td>yes</td>
<td>The source or value of the variable (usually text with macros)</td>
</tr>
<tr>
<td>pattern</td>
<td>no</td>
<td>A regular expression to do a search and replace on the source text with (the use of this attribute requires the inclusion of the replacement attribute).</td>
</tr>
<tr>
<td>replacement</td>
<td>yes (required only if the pattern attribute is given)</td>
<td>A string value to replace all occurrences of the pattern regular expression with</td>
</tr>
</tbody>
</table>
The define statement is used to define variables (or assign values to them). This statement type cannot include any inner statements. Note that it is not required that a variable be defined before it is used.

L.5.2.3. **default**

The default statement is included as a short-cut for defining variables that are currently undefined.

L.5.2.3.1. **Attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>The variable to define.</td>
</tr>
<tr>
<td>value</td>
<td>yes</td>
<td>A default value to assign to the variable if the variable is currently un-defined.</td>
</tr>
</tbody>
</table>

The default statement is similar to the `define` statement in that it assigns a value to a property, but it differs in that the value is *only* assigned if the property was un-defined before the statement executed. The default statement cannot contain any sub-statements.

L.5.2.4. **sleep**

The sleep statement causes the XScript script to pause for a certain period of time.

L.5.2.4.1. **Attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>yes</td>
<td>An integral value indicating a period of time to sleep.</td>
</tr>
<tr>
<td>units</td>
<td>no</td>
<td>A string giving a unit type associated with the <code>value</code> attribute. The default is MILLISECONDS.</td>
</tr>
</tbody>
</table>

The sleep statement causes the running script to sleep for a period of time indicated by the value attribute and the units attribute. The units attribute must contain a string which matches one of the enum values supported by the `java.util.concurrent.TimeUnit` enumeration. Valid values include DAYS, HOURS, MICROSECONDS, MILLISECONDS, MINUTES, NANOSECONDS, and SECONDS. This statement cannot have any sub-statements.
L.5.2.5. exit

The exit statement causes the running script to exit at that moment with the exit code given.

L.5.2.5.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitcode</td>
<td>yes</td>
<td>The integer exit code with which to exit the running script.</td>
</tr>
</tbody>
</table>

The exit statement causes the running script to exit with no further action. This statement cannot have any sub-statements.

L.5.2.6. param

Param statements are used to indicate a parameter to either a grid command or to another XSCript function. These statements have no attributes and the value of the parameter is indicated as the text content of that XML element (thus, param statements can have only text as their inner content, not other XScript statements).

L.5.2.7. condition

Defines a variable as a boolean value based off of conditional statements included as sub-statements of the condition.

L.5.2.7.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>yes</td>
<td>The variable to assign the boolean result to</td>
</tr>
</tbody>
</table>

Condition statements define a variable as having a boolean value. The value assigned to the variable is determined by the conditional expression contained within the condition statement. Only one conditional expression is permitted inside of a condition statement (though some conditional expressions such as or, and, and not may themselves contain sub-statements).

Condition statements must contain a sub-statement which results in a boolean value. These sub statements include:

- and
- or
• xor
• not
• equals
• istrue
• isfalse
• isset
• matches
• compare

L.5.2.8. and

A boolean expression that combines all boolean sub-expressions using a logical and operation. The and statement has no attributes and can contain any number of conditional sub-expressions.

L.5.2.9. or

A boolean expression that combines all boolean sub-expressions using a logical or operation. The or statement has no attributes and can contain any number of conditional sub-expressions.

L.5.2.10. xor

A boolean expression that combines all boolean sub-expressions using a logical xor operation. The xor statement has no attributes and can contain any number of conditional sub-expressions.

L.5.2.11. not

A boolean expression that negates the value of a contained conditional sub-expression. The not statement has no attributes and can contain exactly one conditional sub-expression.

L.5.2.12. equals

A boolean expression which tests whether or not two strings are equal (this statement will work for integer numbers, but only in so much as it does a string comparison of them).

L.5.2.12.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg1</td>
<td>yes</td>
<td>The first argument to compare</td>
</tr>
<tr>
<td>arg2</td>
<td>yes</td>
<td>The second argument to compare</td>
</tr>
<tr>
<td>casesensitive</td>
<td>no</td>
<td>Indicates whether or not the string comparison should be case sensitive. This attribute must have a</td>
</tr>
</tbody>
</table>
value of either true or false. The default value is true.

The equals statement tests for string equality of two arguments. This statement cannot contain any sub-statements.

L.5.2.13.  istrue

Tests whether or not a variable contains the boolean value true.

L.5.2.13.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>yes</td>
<td>The name of the variable to test</td>
</tr>
</tbody>
</table>

This statement is used to test whether or not a variable represents a boolean true value. These variables are usually the ones assigned to by condition statements. This statement cannot contain any sub-statements.

L.5.2.14.  isfalse

Tests whether or not a variable contains the boolean value false.

L.5.2.14.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>yes</td>
<td>The name of the variable to test</td>
</tr>
</tbody>
</table>

This statement is used to test whether or not a variable represents a boolean false value. These variables are usually the ones assigned to by condition statements. This statement cannot contain any sub-statements.

L.5.2.15.  isset

Tests whether or not a given variable has been set yet.

L.5.2.15.1. Attributes
### Property

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>yes</td>
<td>The name of the variable to test</td>
</tr>
</tbody>
</table>

If the indicated variable has been set, then this expression returns true. Otherwise, the result is false. This statement will look for any variable of the given name, thus, scoped variables above the scope of this statement can satisfy this expression. This statement cannot contain any sub-statements.

**L.5.2.16. matches**

Tests whether or not a given string matches an indicated regular expression.

**L.5.2.16.1. Attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>yes</td>
<td>The string to test</td>
</tr>
<tr>
<td>pattern</td>
<td>yes</td>
<td>The regular expression to test the string against.</td>
</tr>
</tbody>
</table>

Does a regular expression test against the given string. This statement cannot contain any sub-statements.

**L.5.2.17. compare**

The compare expression is used to compare two values against one another.

**L.5.2.17.1. Attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numeric</td>
<td>no</td>
<td>A boolean attribute (<strong>true</strong> or <strong>false</strong>) that indicates whether numeric, or textual comparison is indicated. The default value is <strong>false</strong>.</td>
</tr>
<tr>
<td>arg1</td>
<td>yes</td>
<td>The first argument to compare</td>
</tr>
<tr>
<td>arg2</td>
<td>yes</td>
<td>The second argument to compare</td>
</tr>
</tbody>
</table>
An enumerated attribute which contains one of the values \( \text{lt, le, eq, ge, or gt} \).

The comparison statement produces a boolean value by testing two arguments against one another. If the numeric attribute is true (the default is false), then the comparison will be a numeric comparison. Otherwise, the canonical alphabetization comparison is used. The comparison to perform is determined by the \textit{comparison} attribute which has one of the following values (and associated meanings).

\[ \text{lt} : \text{true if arg1 is less than arg2} \]
\[ \text{le} : \text{true if arg1 is less than or equal to arg2} \]
\[ \text{eq} : \text{true if arg1 is equal to arg2} \]
\[ \text{ge} : \text{true if arg1 is greater than or equal to arg2} \]
\[ \text{gt} : \text{true if arg1 is greater than arg2} \]

This statement cannot contain any sub-statements.

\textit{L.5.2.18. switch}

The switch statement is used to select a block of code to run from amongst a number of choices.

\textit{L.5.2.18.1. Attributes}

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>yes</td>
<td>The text to use in the switch comparison</td>
</tr>
</tbody>
</table>

The XScript switch statement works very similar to the way that switch works in bash. THe value attribute gives a string value which is to be compared against a number of regular expressions. The first one that it finds that matches is run. Only two XScript statements are permitted as children of the switch statement -- \textit{case} and \textit{default}. 0 or more \textit{case} statements are permitted, but only 0 or 1 \textit{default} statements can be included. If no \textit{case} statements match, then the \textit{default} (if it exists) is run.

\textit{L.5.2.19. case}

Case statements occur inside of \textit{switch} statements and indicate a possible match for the \textit{switch} value.

\textit{L.5.2.19.1. Attributes}

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
</table>
Case statements contain other statements that are run when the case pattern matches the outer `switch` value. Case statement can container 0 or more other XScript statements.

**L.5.2.20. default**

Default statements occur inside of `switch` statements and are run when no `case` statement matches the `switch`s value attribute. Default statements contain no attributes and can container 0 or more other XScript statements.

**L.5.2.21. if**

The if statement is used to execute code conditionally based off of a boolean property value.

**L.5.2.21.1. Attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>yes</td>
<td>Gives the name of an attribute (not the value) that is to be tested for a boolean value. This is often a property set by the <code>condition</code> statement.</td>
</tr>
</tbody>
</table>

The if statement contains either one or two sub-statements that indicate which script code to run based off of the truth of the test property. If only one sub-statement is included, it must be the `then` statement element. If two are included, one must be `then` and the other must be the `else` sub-statement. If the test property is true, then the `then` statement block is executed. If the test property is not true, and if an `else` sub-statement exists, then the `else` block of script is executed.

**L.5.2.22. then**

This statement is included inside of an `if` statement and the block of code is run if and only if the `test` value from the `if` statement tested true. The then statement can contain any number of sub-statements.

**L.5.2.23. else**

This statement is included inside of an `if` statement and the block of code is run if and only if the `test` value from the `if` statement tested false. The else statement can contain any number of sub-statements.
The for statement permits a simple kind of iteration structure where one can iterate over a count of integers.

### L.5.2.24.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param-name</td>
<td>yes</td>
<td>Gives the name of a variable that will contain the count value.</td>
</tr>
<tr>
<td>initial-value</td>
<td>no</td>
<td>Gives an initial integral value for the count to start with. The default value is 0.</td>
</tr>
<tr>
<td>increment-value</td>
<td>no</td>
<td>Gives an integral value by which the count will be increased every time through the loop. The default is 1.</td>
</tr>
<tr>
<td>inclusive-limit</td>
<td>one of inclusive-limit and exclusive-limit is required</td>
<td>Gives a limit (inclusive) up to which the count proceeds before the loop exits.</td>
</tr>
<tr>
<td>exclusive-limit</td>
<td>one of inclusive-limit and exclusive-limit is required</td>
<td>Gives a limit (exclusive) up to which the count proceeds before the loop exits.</td>
</tr>
</tbody>
</table>

The for statement is a simple looping statement which executes all contained sub-statements one time each for each progression of a count variable. This statement can contain any number of sub-statements.

### L.5.2.25. foreach

A specialization of the for loop, the foreach statement loops over a set of values.

### L.5.2.25.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
</table>
The foreach statement permits the script to iterate over a set of entries. The entries are given either by lines in a text file (using the `source-file` attribute) or over entries in a local (`source-dir`) or a grid (`source-rns`) directory. If the `filter` attribute is given, then only entries or lines which match that regular expression are used. The foreach statement can contain any number of XScript sub-statements.

### L.5.2.26. `throw`

Causes the XScript engine to throw a Java exception.

#### L.5.2.26.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>yes</td>
<td>The Java exception class to throw an instance of.</td>
</tr>
<tr>
<td>message</td>
<td>yes</td>
<td>A string message to construct the Java exception class with.</td>
</tr>
</tbody>
</table>

Throws a new exception (which has to be a valid Java exception). This statement cannot contain any other sub-statements.
L.5.2.27. try

Being a try/catch/finally block for catching exceptions thrown from sub-statements. This statement can only contain sub-statements of the block, catch, or finally type. The try statement must contain exactly one block statement, any number of catch statements, and a single optional finally statement.

When encountered, the try statement causes the block statement to be executed. If an exception is thrown which matches any of the catch statements, then the first matching catch statement is executed. Finally, if a finally statement is included, then that statement is executed after the block and any catch statements no matter what.

L.5.2.28. block

This statement is included inside of try statements and can contain any number of sub-statements. This statement is always executed inside of a try statement.

L.5.2.29. catch

The catch statement is used to catch exceptions thrown from a block of script code.

L.5.2.29.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>yes</td>
<td>The Java exception class to catch.</td>
</tr>
<tr>
<td>property</td>
<td>no</td>
<td>A property name to hold the exception (if caught).</td>
</tr>
<tr>
<td>message</td>
<td>no</td>
<td>A property name that will hold the exception's text message when caught.</td>
</tr>
</tbody>
</table>

When an exception is caught, the property and message attributes indicate variable names to use to store the exception and the message from the exception. The former is of limited use at this point as the language doesn't deal well with complex types, but the message can be used for printing out text. The catch statement can contain any number of sub-statements.

L.5.2.30. finally

A finally statement is guaranteed to be run at the end of every try statement no matter what happens inside the try's block or catch statements. It can contain any number of sub-statements.

L.5.2.31. function
A function statement defines a new function that can be called from other parts of the script.

L.5.2.31. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>The name to give the function.</td>
</tr>
</tbody>
</table>

Functions can contain any number of sub-statements and are called using the call statement. Inside of a function, the ARGV array variable is defined to match the parameters that were used to call that function.

L.5.2.32. return

Return statements allow for early returns from a defined function. Thus, they can only be used inside of function statements.

L.5.2.32.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>One of property, value, or included sub-statements is required</td>
<td>Gives the name of a property whose value is to be returned.</td>
</tr>
<tr>
<td>value</td>
<td>One of property, value, or included sub-statements is required</td>
<td>Gives a string value to return as the result.</td>
</tr>
</tbody>
</table>

Return statements return from a function with a given value. The value is either taken from the property attribute, the value attribute, or it can be set by executing any number of contained sub-statements (the value of a set of sub-statements is the value of the last statement to execute).

L.5.2.33. call

Calls a defined function by name.

L.5.2.33.1. Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Required?</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>function</td>
<td>yes</td>
<td>Indicates the name of the function to call.</td>
</tr>
<tr>
<td>property</td>
<td>no</td>
<td>Indicates a property or variable that will hold any results returned from the function.</td>
</tr>
</tbody>
</table>

A call statement is used to call a defined function by name. This statement can only have `param` statements contained within as sub-statements. If the property attribute is included, then it indicates the name of a property or value that is to hold the result returned from the function after the call.

L.5.2.34.  

```
parallel
```

The parallel statement marks a block of code that MAY contain `parallel-job` statements to execute in parallel threads.

L.5.2.34.1. Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread-pool-size</td>
<td>no (default is 8)</td>
<td>An integer indicate how many simultaneous threads can be used to execute parallel jobs.</td>
</tr>
</tbody>
</table>

A parallel statement is used to mark the beginning and end of a section of code that might be run in parallel. Parallel statements can contain any number of sub-statements. When a parallel statement start executing, a thread pool is created of the indicates size. Every time a `parallel-job` statement is encountered, its statements are automatically queued into the thread pool and may execute in parallel. When the parallel statement finishes, it blocks until all running or queued parallel jobs finish.

L.5.2.35.  

```
parallel-job
```

A parallel job statement indicates a section of script code which can execute in parallel. These statements do not have to be contained directly under `parallel` statements, but they have to exist somewhere within a `parallel` statement at some level. Parallel-job statements are automatically enqueued into the nearest (on the call stack) `parallel` statement block. Parallel-job statements can contain any number of sub-statements.
M. GFFS Exports Explicated

GFFS Exports Explicated

Everything you might ever want to know about GFFS exports

M.1. What is an Export?

A GFFS export is a mechanism to make data in local file systems accessible, subject to access control, to remote users and clients. GFFS exports are realized via a GFFS container (service) that supports standard protocols for file and directory access. The GFFS container receives requests from external clients, verifies that the external client has permission to perform the desired action, and if so, performs the action on the exported files or directories. Thus the GFFS container can be thought of as a proxy server that provides a secure alternative access mechanism to the directory structure being exported. This is shown in Figure 48. Remote clients interact securely with the GFFS container using Web Services calls. Authorization is based on owner-definable Access Control Lists (ACLs).

![Figure 48 Remote Client Interaction with Container](image)

Read, write, create, and delete operations performed by remote clients on the GFFS container are immediately executed against the local file system and are therefore immediately visible to local users of the file systems. Similarly, read, write, create, and delete operations performed by local users on the local file system are immediately visible to remote clients.

The GFFS container executes as some identity in the local administrative domain. For example, an end-user might run a GFFS container on their desktop as their local identity, or a department administrator may run a container as a newly created, non-privileged, gffs-user. In either case the GFFS container accesses the local file system as that user and can only access files and directories which that user may access. The GFFS container does not run as root (i.e., as a privileged user), and thus is subject to the normal file access controls of the operating system.

2 "Immediately" here means at the next access by the client. Clients may use caching strategies though.
Therefore, to export a local file system directory tree the GFFS container must both be able to traverse the path to the exported directory tree and have permission to read, write, create, and delete files and directories in the directory tree.

**M.2. Types of Exports**

As of XSEDE SDI-175 (GFFS version 2.7.560), there are three mechanisms used by the GFFS container to access local file systems. They are known as *ACL Exports, ACLAndChown Exports, and ProxyIO Exports*. The ACL export type was the only type of export available until SDI-175. To the end user, the details of the container’s configuration only matter with respect to whether the user will need to take action to make the exported data visible to the GFFS container.

**M.2.1. ACL Exports**

If a path to be exported is already visible to the container, then no special configuration is needed. More specifically, an exported directory must be accessible by the user account that the container runs under. GFFS containers run as a non-privileged user, so cannot take advantage of the root account’s full view of the filesystem.

When the exported paths are not already visible to the container account, then Linux Extended ACLs must be used to make the export directory structure visible to the container account. Through extended ACLs, the export owner can grant rights to the grid container’s Unix user on specific directories, which enables the container to export those directories. The extended ACLs feature is only available for certain types of filesystems (mainly on Unix). Section M.3.3 below covers how a user can set extended ACLs to allow the container account to see the exported path.

**M.2.2. ACLAndChown Exports**

The ACLAndChown export type is identical to the ACL type of export, except that new files are “chowned” to be owned by the Unix user who is determined to be the export’s owner. The basic idea is simple; file ownership is reassigned when files are created on the export. This permits accounting mechanisms that are based on file ownership to properly assess how much space the user has allocated to their files. If the “chown” was not done, all the space allocations would be billed to the gffs container user rather than to the specific user that owns the files.

The ACLAndChown export mode requires configuration by an administrator, because the “sudoers” file must be adjusted to allow container to run “chown”. The system’s “grid-mapfile” must also be edited to map the user’s “preferred identity” to a real OS user. Setting up *sudo* capabilities for the container is explained below in section M.4.7 and *grid-mapfile* editing is explained in section M.4.6.

**M.2.3. ProxyIO Exports**

The ProxyIO type of export is quite different from the two ACL export types, and relies on a separate co-process that runs under the export owner user’s actual Unix identity in order to obtain access to the exported files. The container launches the IO co-process if no existing co-process is
found, and ensures that it is talking to the right co-process via a shared secret “nonce”. Each export will have at most one ProxyIO co-process active at a time, and the co-process will be active as long as the export is in use. After a certain amount of inactivity, the co-process dies (and it is restarted as needed).

The ProxyIO co-process ensures that all activity on an export is done as the export owner’s corresponding Unix user. No other Unix user identity or group identity matters when accessing the export; it is exactly as if the user herself were performing all the file operations taken by the co-process. Within the grid, all authorization decisions for the export are made using grid ACLs, and the export owner’s real Unix account is invisible to other grid users.

Configuration on the GFFS container side is more involved than for ACL exports, but the end user who owns the export does not have to do anything extra to make their data visible. When configured properly for ProxyIO exports, any path that the user can normally access can be exported to the grid without further action on the user’s part.

The ProxyIO export mode requires configuration by an administrator, because the "sudoers" file must be adjusted to allow container to launch the ProxyIO co-process. The system’s “grid-mapfile” must also be edited to map the user’s “preferred identity” to a real OS user. Setting up sudo capabilities for the container is explained below in section M.4.7 and grid-mapfile editing is explained in section M.4.6.

Table 1 GFFS Container Export Configuration Options

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Typical use case</th>
<th>Advantages and disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL - single user</td>
<td>A user wants to export data on a desktop computer or laptop, or a small server. All files to be exported are owned by the user that will be executing the container.</td>
<td><strong>Advantage:</strong> Very simple to set up. Requires no management of local access control environment. <strong>Disadvantage:</strong> one container is needed for each user.</td>
</tr>
<tr>
<td>ACL - multi user</td>
<td>Department or center file system that supports Linux extended ACLs.</td>
<td><strong>Advantage:</strong> requires little work for the system administrator. Users have explicit control over which directories it is even possible to export. <strong>Disadvantages:</strong> Users must explicitly execute extended ACL commands to make directory structures accessible to GFFS container.</td>
</tr>
</tbody>
</table>
### ACLAndChown

Department or center file system that supports extended ACLs AND organization wants to enforce file system quota’s on a user basis.

**Advantage:** requires a bit more work for the system administrator. Users have explicit control over which directories it is even possible to export. Native quota systems work properly.

**Disadvantages:** Users must explicitly execute commands to make directory structures accessible to GFFS container. Administrators must configure `sudoers`.

### ProxyIO

Multiuser environment with no extended access control lists in the underlying file systems, or where admin wants to control all exports.

**Advantage:** requires more work by the system administrator. Extended access control lists not required. Very easy for end users.

**Disadvantages:** Administrators must configure `sudoers` and grid-mapfile configuration files. More processes running at any given time.

---

## M.3. Users View - Exporting File Systems Directory Trees to the GFFS

### M.3.1. Creating an Export in the GFFS

Creating an export establishes a mapping between two paths; the local path on the host where the GFFS container is running is mapped to the target directory where the export is to appear in the GFFS directory structure. This is similar to the concept of a *mount point* in Unix. The user creating the export also needs “read” and “execute” permission on the container’s Export Port Type (if permission is not set up yet, how to do so is documented below for container owners).

The basics of creating an export are the same regardless of the type of the export configured on a container. Given that the user has permissions to create exports on the container, has permissions in the GFFS at the grid path in question, and has permissions on the operating system path specified, then this command creates the export:

**General syntax**

```bash
grid export --create {containerPath} local:{/path/on/host/to/dirToExport} \ 
grid:{/path/inGrid/to/exportedDir}
```

The export command requires that the local directory “dirToExport” be visible on the local file system of the container specified by “{containerPath}”. The command also requires that the grid user executing it has permissions to create the grid directory “exportedDir” in the specified parent directory.
For example user “xd-fred” on the host Mason at Indiana might export a directory "myData" off of their home directory using the command:

```
grid export --create \\resources/xsede.org/mason.iu.xsede.org/containers/mason-gffs \\
local:/home/xd-fred/myData  grid:/home/xsede.org/fred/mason-data
```

The method for granting visibility of the exported data to the container depends on the type of export in use on the container. For “ACL” and “ACLAndChown” type exports, Section M.3.3 describes how to use extended ACLs for fine-grained control of file and directory visibility. For “ProxyIO” type exports, the user herself does not need to make any accessibility changes, but the container owner has a fair amount of configuration to enable the exports (see section M.4 for information on container-side configuration).

Once the export is created successfully, the contents of the local dirToExport are made visible inside the grid path at the exportedDir path. Any files that are written on the user’s computer are made available seamlessly within the GFFS grid, and vice-versa, in that changes made to the files and folders within the grid’s RNS space are reflected in the user’s local filesystem. This technique is very useful because it makes data accessible within the GFFS while keeping the files’ actual source on the user’s computer, which avoids any extra copying to keep the data in sync, as well as freeing the user from needing storage on the container itself.

Cautionary Note: A user should never export the top-level of their home folder into the GFFS grid. This is because the top of the home folder stores many confidential directories and files (such as the user’s ssh and gnupg keys). The only time a top-level of home might be valid to export is when the user does not intend to give any other user access, since allowing other users to view the export at all would compromise those sensitive files.

### Setting Permissions on the Export

Now that the user has created an export of some portion of their local filesystem into the grid, the user may want to share that data with other users or groups in the grid. This is accomplished using the grid “chmod” command, as shown below:

```
# general syntax:
grid chmod {asset to share} {permissions, i.e. r, w, and x} {user or group}  
# allow arnoldg to read files in my exported folder.  
grid chmod /home/xsede.org/fred/mason-data +rx /users/xsede.org/arnoldg  
# allow ssmallen to read and write files in my exported folder.  
grid chmod /home/xsede.org/fred/mason-data +rw /users/xsede.org/ssmallen  
# allow the gffs-admins group to read my exported data.  
grid chmod /home/xsede.org/fred/mason-data +rx /groups/xsede.org/gffs-admins
```

The chmod command should always include “x” (execute permission), since users trying to access the exported folder need to be able to traverse into that folder and sub-folders.

### M.3.2. LightWeight vs. HeavyWeight Exports

In general, users tend to prefer “lightweight” exports. These are implemented by the LightWeightExportPortType in the GFFS container’s services. Lightweight exports are usually
preferable to “heavyweight” exports (implemented instead by the ExportedRootPortType), because lightweight exports provide a simplified permission model and are significantly faster in operation. The permissions on a lightweight export are set at the top of the export, and affect all files and directories underneath. In contrast, a heavyweight export has ACL records for every file and directory stored in the GFFS grid, which provides finer-grained security at the expense of speed and storage overhead (kilobytes of access control information per file and directory). The new ProxyIO type of export is only available for lightweight exports at this time.

To create a heavyweight export with the export tool, one can use the “--heavy” flag to indicate this. The earlier example command can instead use a heavyweight export like so:

```
grid export --create --heavy /
/resources/xsede.org/mason.iu.xsede.org/containers/mason-gffs \
local:/home/xd-fred/data grid:/home/xsede.org/fred/mason-data
```

The resulting “mason-data” folder in the grid operates like the lightweight export, but any subdirectory or file under it can be given specific permissions for a grid user without affecting any other exported files and directories. Contrarily, setting permissions on assets under the lightweight export affects all assets that are available on that export.

### M.3.3. Setting Extended ACLs for ACL and ACLAndChown Exports

An ACL type of export works fine when the grid container’s user account (i.e. the Unix user running the container) already has access to the files in question. But if the files are owned by a different account, then some additional steps may be necessary to make the files visible to the grid container. The approach used for ACL and ACLAndChown exports depends on the “Extended Attributes” (or Extended ACLS) feature of modern Unix and Linux operating systems to grant the container’s user account the necessary permissions on the files and directories that a user wants to export. Extended ACLs allow the granted file access to be very precise and limited, rather than a blunderbuss-like approach such as “chmod -R 777 /my/path”. Extended ACLs are not available on all platforms or filesystems.

It is important to be aware that it is generally the user that owns the data who must actually take action to make the files available. The unprivileged user account that runs the GFFS container normally cannot help with this. But it is not difficult to change the ACLs on a folder, and should be possible for most users to accomplish. A script for editing the extended ACL information is provided as part of the XSEDE tools and tests suite (see section I.1 for detailed information about the XSEDE tools and tests) called “set_acls.sh” (look in the “library” folder under the tools). If Extended ACLs are supported on the OS platform and the affected file system, then this script can grant the grid container account access to the specified directories in the local file system, which then allows the container to export them. The script is run as the user who owns the files, i.e. as the local Unix account of the person who wants to export a hierarchy into the grid. It is worth repeating that the user must set extended ACLs before the container will be able to export the directory in ACL or ACLAndChown modes.
For example, I am logged in as user “fred” in the local Linux system where I would like to allow the grid container to export some of my files. The grid account is running as user “gffs”. My folder to export is located in “/home/fred/new-source”. Using the set_acls script, the folder can be made ready for export with these two commands:

```
bash $XSEDE_TEST_ROOT/library/set_acls.sh gffs /home/fred rx
bash $XSEDE_TEST_ROOT/library/set_acls.sh -R gffs /home/fred/new-source rx
```

The first command allows my home folder to be accessed by the gffs account, but only that specific folder since the recursive flag is omitted. Other content in my home directory still has the pre-existing access level granted by normal Unix file permissions.

The second command recursively enables the gffs account to see all content under the “new-source” folder, but not to modify the contents. After the extended ACL has been established on “new-source”, the grid administrator can create an export for that folder (see prior section) on the grid container running as the “gffs” user.

The example command below opens a “scratch” folder that I can allow other users to write into:

```
bash $XSEDE_TEST_ROOT/library/set_acls.sh -R gffs /home/fred/scratch rwx
```

This command provides the container with the ability to write files into the user’s local folder when the exported files are modified.

Once an export is created by the grid administrator, I can grant another grid user the ability to read or write in the exported path within the grid. With full read-write permission given to the GFFS container user via extended attributes, that user will be able to store, modify and delete files in that local folder from within the grid. Other grid users have no such rights until granted, and thus will not even have visibility of an export until the owner grants them read access to the folder in the grid.

### M.3.4. Preferred Identity Management

The “preferred identity” for a user is a feature that tracks the user’s primary identity for use in decision making about file ownership and exports. The user’s preferred identity is generally established automatically. When the grid user logs in to an XSEDE identity, the preferred identity is set based on their MyProxy credentials. Similarly for a username+password login to a non-XSEDE identity, the preferred identity is set based on the STS that authenticates the user.

The preferred identity can be displayed using the “identity” tool. For example:

```
grid identity --show
```

This will display the current preferred identity that has been set, if one exists. There will be no preferred identity if one is not logged in. An example listing from the identity tool for my XSEDE login is below:

```
$ grid identity --show
```
The current preferred identity is: /C=US/O=National Center for Supercomputing Applications/CN=Chris Koeritz

The real point of the preferred identity is to mark one particular identity as the applicable one for services that need to know one and only one identity to associate with a particular request. The GFFS credential wallet is designed to hold onto many different identities as needed for authentication, but in a case where the wallet has more than one USER credential type, there is no obvious way to know which user is desired. Thus the preferred identity makes the choice of user obvious. Most users will never need to set or modify their preferred identity, but power users require a way to specify which of their credentials is associated with a request.

For exports in particular, the preferred identity provides the definitive owner of an export when it is about to be created. The preferred identity is used to look up the appropriate DN in the grid-mapfile in order to determine the real operating system user for a grid identity.

To get the built-in documentation for the preferred identity tool, use the “man” command:

```
grid man identity
```

There are quite a few options available, and we will cover the most important ones in this section.

To show all the possible preferred identities that one could choose from, run:

```
grid identity --listAll
```

When I have logged into my XSEDE portal account “koeritz” first, and then also logged into another user credential (for my “fred” user on a different grid), then the listing of all possible identities may show a list like this:

```
User and Connection Identities in Wallet:
"/C=US/O=National Center for Supercomputing Applications/CN=Chris Koeritz"
"/C=US/ST=Virginia/L=Charlottesville/O=Testing Grid/OU=Genesis II/SERIALNUMBER=urn:ws-naming:epi:E6357794-8AA7-BF1B-9D6F-892612EDA516/O=TERAGRID.ORG/CN=koeritz/CN=KerbAuthnPortType"
"/C=US/ST=Virginia/L=Charlottesville/O=Testing Grid/OU=Genesis II/SERIALNUMBER=urn:ws-naming:epi:12D42885-7D5C-7DED-5C1F-F280FD253183/CN=fred/CN=X509AuthnPortType"
```

The “listAll” command displays all of the current identities in the proper format, which is “one-line openssl RDN format”. This is compatible with how identities are listed in the grid-mapfile. For most users, this will only show their TLS session credential (a MyProxy X509 certificate for XSEDE users) and their grid identity (based on a Kerberos STS for XSEDE users). For users who have authenticated to multiple STS identities, this will show all of those other identities as well.

To pick a different user listing as the preferred identity, one can either specify the exact DN string that is preferred or one can use a pattern to find a matching DN in the existing credentials. My current preferred identity is based on my XSEDE portal account (“koeritz”), but I may need to perform some actions with a different credential based on my “fred” user. I can specify a different preferred identity by using the full DN string:
This is quite a verbose and unwieldy command, and the DN string must match the exact form shown in the listAll command. An easier way to pick the preferred identity is to use a pattern to specify the DN. (At this time, this is a simple string match rather than a regular expression match):

```bash
$ grid identity --set --pattern="CN=fred"
```

The preferred identity has been set.

The current preferred identity is: /C=US/ST=Virginia/L=Charlottesville/O=Testing Grid/OU=Genesis II/SERIALNUMBER=urn:ws-naming:epi:12D42885-7D5C-7DED-5C1F-F280FD253183/CN=fred/CN=X509AuthnPortType

To remove my preferred identity, I can use the "reset" flag:

```bash
$ grid identity --reset
```

The preferred identity has been reset.

The preferred identity is used in conjunction with the grid-mapfile in order to map a GFFS grid user to an Operating System user. Thus the DN strings of the preferred identity must be found in the grid-mapfile or no mapping can occur, and this will cause export creation in ProxyIO mode to fail. A missing DN string for a user will also cause ACLAndChown exports to fail to "chown" the file to the proper Unix user. Exports created in these modes further require that the grid-mapfile continues to list the appropriate DN string for the user, or the ability to map to the proper user will stop and the export will begin throwing faults during operation.

### M.4. System Administrator Considerations

As described earlier there are three different GFSS container configuration options, **ACL Exports**, **ACLAndChown Exports**, and **ProxyIO Exports**. The mechanism to be used is determined via configuration files. The default behavior is ACL Exports.

#### M.4.1. Establishing the Container's Export Configuration

The first step in setting up a container for exports is to make a copy of the default `export.properties` file that is shipped with the installation. This should be copied into the container state directory in order to define an overridden set of properties for exports. For example:

```bash
cp $GENII_INSTALL_DIR/lib/export.properties $GENII_USER_DIR
```

All configuration topics below will assume that modifications occur on the file `$GENII_USER_DIR/export.properties`.

#### M.4.2. ACL Export Mode Configuration
To set the container to use ACL export mode, perform the following steps:

a. Edit the “$GENII_USER_DIR/export.properties” file and change the “export.Mechanism” entry to “ACL”.

The container side configuration for an ACL type export is minimal. The export owner, rather than the container administrator, should change extended attributes (covered in section M.3.3) on the appropriate directory to allow the container user to “see” it (that is, the container’s Unix account needs at least “rx” access, and it needs “w” access if the export is to allow any writes). If necessary, a system administrator can help the user do this, but the GFFS container account’s user generally does not have the system privileges to help out.

The main administrative duty here (and one that is shared with all types of exports) is to ensure that users have permission on the appropriate export port type in the grid. This is only required if users are allowed to create exports on their own; if only the administrator will create exports, then just the grid administrator (or container owner) needs rights on the appropriate export port type. Creating exports for other users is discussed further in Section M.4.8.

M.4.3.  ACLAndChown Export Mode Configuration

To set the container to use ACLAndChown export mode, perform the following steps:

a. Edit the “$GENII_USER_DIR/export.properties” file and change the “export.Mechanism” entry to “ACLAndChown”.

ACLAndChown exports share the requirement of ACL exports that the export owner must provide visibility of the exported directory to the container’s Unix account (covered in section M.3.3).

b. Additionally, ACLAndChown exports require that the grid-mapfile be set up with each Unix user who is allowed to create exports. Configuring the grid-mapfile is covered in Section M.4.6. The grid-mapfile tells the container which Unix user should own any new files created on the export. Note: for XSEDE Service Providers, this is the same grid-mapfile that you are already using.

c. Another specific requirement for ACLAndChown is that the gffschown script is configured in the export.properties file. Generally the script that ships with the Genesis II install is adequate; if no additional actions are needed in the gffschown script at the time file ownership is changed, then the default script should be fine.

d. A final requirement for ACLAndChown exports is that the gffschown script be added to the “/etc/sudoers” file to allow the GFFS container user to execute it as any user. The sudoers configuration in general is covered in Section M.4.7.

M.4.4.  ProxyIO Export Mode Configuration

To set the container to use ProxyIO export mode, perform the following steps:

a. Edit the “$GENII_USER_DIR/export.properties” file and change the “export.Mechanism” entry to “ProxyIO”.

The ProxyIO export takes most of the burden off of the end users for exports, since they no longer need to change any extended attributes on their files to make them visible to the container.
However, the administrator must perform a few configuration tasks in order to use ProxyIO exports.

b. One important consideration for the ProxyIO export mode is whether users have visibility to the Genesis II installation directory. If the install is system-wide and installed by root, then all users should be able to see the folder. However, if the container installation is located in a personal folder, then all relevant users (who might want an export in the GFFS) must be granted visibility to the install folder to permit the ProxyIO co-process to be launched. For example, the container user could execute “chmod 755 $HOME/GenesisII” to make the personal installation folder visible to all users.

c. ProxyIO exports use the grid-mapfile similarly to ACLAndChown exports; the export owner DN is sought out in the map file, and converted into the corresponding Unix user if possible. The grid-mapfile configuration is covered in Section M.4.6.

Once the proper Unix user is known via the grid-mapfile, the GFFS Container either launches a new ProxyIO server co-process as the export’s owning user, or it connects to an existing ProxyIO server if already available for that user. ProxyIO servers time out eventually if they are not accessed within a certain time period, so these co-processes do not survive long as zombie processes if the container is restarted. The ProxyIO server only listens to connections on localhost, to avoid providing any external attack surface. The ProxyIO server also only serves requests to clients who possess the “secret nonce” that it was given by the container when the co-process was started up.

d. The ProxyIO server requires setting up a sudoers configuration, which is covered in Section M.4.7.

e. One additional consideration for ProxyIO exports is where to allow users to create exports in the file system. The default configuration shipped with the GFFS install actually initially disables any path from being exported. It may be appropriate for some sites to modify this to allow any path in the operating system to be exported, when users already are restricted from viewing any paths that would be inappropriate to export onto the grid. Finer grained path permissions are also possible and are probably the most generally applicable, since sensitive OS paths can easily be disallowed. The paths allowed for export creation are configured with the gffs.exports file (documented in section M.4.9.2).

### M.4.5. Enabling Export Creation for Grid Users

A grid user that wishes to create an exported directory in the grid must have “read” and “execute” permission on the container’s appropriate Export Port Type. The port type will either be LightweightExportPortType for lightweight exports or ExportedRootPortType for heavyweight exports. If a user (or a group) has not been granted access to the export port type, then the following command can be executed by an administrator (or container owner) to grant permission to create exports:

```
# make container visible to users (do for both lightweight and heavyweight exports)
grid chmod {containerPath} +r {user or group identity}

# command to enable use of lightweight exports:
grid chmod {containerPath}/Services/LightWeightExportPortType +rx {user or group identity}

# command to enable use of heavyweight exports:
grid chmod {containerPath}/Services/ExportedRootPortType +rx {user or group identity}
```
This is a specific example that allows the user “koeritz” to create lightweight exports on the XSEDE root container (gffs-users can already see the container, thus we omit the first chmod):

```bash
grid chmod /resources/xsede.org/containers/gffs-1.xsede.org/Services/LightWeightExportPortType \\
   +rx /users/xsede.org/koeritz
```

Here is another specific example that allows export creation on a personal container by members of gffs-admins:

```bash
grid chmod /home/xsede.org/fred/my-container +r /groups/xsede.org/gffs-admins
grid chmod /home/xsede.org/fred/my-container/Services/LightWeightExportPortType +rx \\
   /groups/xsede.org/gffs-admins
```

After these “chmod” commands are executed, the specified group or user is enabled to create exports that will be hosted on the container specified.

### M.4.6. Configuration of the Grid-Mapfile

The “grid-mapfile” is an important link between identities in the GFFS grid and user accounts in the operating system. This file is a standard security configuration file for the grid community, and is more fully documented at [https://dev.globus.org/wiki/Gridmap](https://dev.globus.org/wiki/Gridmap). The file simply lists the X500 DN (in OpenSSL one-line RDN format) and the Unix user account. It is standard for XSEDE (and others) to store this file in “/etc/grid-security/grid-mapfile”. Here is a short example grid-mapfile that contains my “koeritz” account (and some other bogus DNs):

```
"/C=US/O=National Center for Supercomputing Applications/CN=Calinda Moshen" calmo
"/C=US/O=National Center for Supercomputing Applications/CN=Chris Koeritz" drake
"/C=US/O=National Center for Supercomputing Applications/CN=Sandy Filsteen" sandyf
```

This file specifies that when I perform activities as my XSEDE MyProxy identity of “koeritz”, then the associated Unix user account is “drake”. Thus when I create an export with the container in ProxyIO export mode, it will start a co-process as the “drake” account and serve files to the GFFS as that Unix identity. Similarly, in ACLAndChown mode, any new files created by the “koeritz” user will be set to be owned by “drake” in the underlying file system.

To configure more identities in the grid-mapfile, simply add more pairs of “DN string” + “OS User name” as new lines in the file.

### M.4.7. Configuration of the “sudoers” File for Sudo Access

Both the ACLAndChown and ProxyIO types of exports require modification of the /etc/sudoers file to permit the container user account to execute with the proper permissions. The required configuration changes can be made with the “visudo” tool, for example:

```
sudo visudo
```
Two different types of sudo permission lines will be required, based on the two types of exports that need sudo-level access (ACLAndChown and ProxyIO).

**M.4.7.1. ACLAndChown Exports: Allowing the Container to Sudo Gffschown**

The simpler configuration is for the ACLAndChown type of export, which just needs a line similar to the following added to /etc/sudoers:

```
gffsuser ALL=NOPASSWD: /opt/genesis2-xsede/gffschown
```

This configuration line grants the “gffsuser” access to run the “gffschown” script from the default installation location for the XSEDE grid. If your container runs as a different user, substitute that for “gffsuser”. Similarly, if you are running a different version of Genesis II on a different path, then the path to “gffschown” should be modified.

If one wishes to run a different version of gffschown, perhaps with expanded logging or security checks, the path to this other version could be placed in /etc/sudoers instead of using the default installation path for gffschown.

**M.4.7.2. ProxyIO Exports: Allowing the Container to Sudo Proxyio.launcher**

The sudo configuration for ProxyIO type exports is more complex, but still not unfathomable. Feedback from system administrators has indicated a desire to control the set of users that can start a ProxyIO server for serving files to the GFFS container. This is implemented by adding a line for each user who is allowed to have a GFFS export. For example:

```
gffsuser ALL=(drake) NOPASSWD: /opt/genesis2-xsede/proxyio.launcher
```

This grants the container Unix user account “gffsuser” the ability to run the ProxyIO server as the Unix user called “drake”. The launcher script is provided by the installation and generally does not need to be modified, although an alternative launcher script can be substituted.

This line will need to be added for each user that can launch the ProxyIO server. This configuration may be too unwieldy for some system administrators who would prefer to trust that the GFFS will only launch the ProxyIO server appropriately for user export actions. If the admin would rather add a single configuration line instead, this line grants the container user account the ability to run ProxyIO servers as any user:

```
gffsuser ALL=NOPASSWD: /opt/genesis2-xsede/proxyio.launcher
```

These configuration changes to the sudoers file allow the GFFS container to run (via sudo) the gffschown and proxyio.launcher scripts as needed. It is important to realize that the sudoers modifications must be done before new exports are created on the GFFS container in ACLAndChown or ProxyIO mode. Otherwise, failures will occur at export creation time, or new files on the export may not be created with the proper user.

**M.4.7.3. Special Sudo Considerations**
It is important to note that some platforms also require any user who will be allowed to “sudo” to be a member of the “sudo” group. This can be done by editing “/etc/group” and adding the user at the end of the line with the entry for the “sudo” group.

Further, on some platforms, the sudo configuration may need to be adjusted to allow the sudo command to be executed from Java by the GFFS container without a tty interface. If this problem is occurring, then the container will be logging failures to communicate with the ProxyIO process that will resemble this line:

```java
java.io.IOException: got erroneous response from proxyio startup, said 'sudo: sorry, you must have a tty to run sudo' instead of 'ready'
```

The solution to allow sudo to run without a tty interface is to comment out the following two lines in the /etc/sudoers file (by running “visudo” to edit that file appropriately):

```ini
[Defaults requiretty]  #Defaults requiretty
[Defaults !visiblepwd] #Defaults !visiblepwd
```

After this, the ProxyIO export mode should start working properly, and the container should stop logging errors about it.

One can determine if the ProxyIO co-process is running for a particular user with this command:

```bash
ps auwax | grep -i java | grep -i $USER | grep -i 'proxy.*io'
```

If this prints out any output, then the co-process is running for the current value of $USER. If you want to check whether the ProxyIO co-process is running as a different user than your current Unix login, replace $USER with that user’s name when running the command.

### M.4.8. Creating Exports for Other Grid Users

When exports are created, the preferred identity is automatically used to determine the owner of the export. In most cases this is sufficient, but in some cases, the grid administrator may not want individual users to create their own exports. This is enabled by a special flag on the export tool called “--force”. If the force flag is seen, then an administrative user can create exports for any DN string, even if they do not own that credential themselves.

Forcing the ownership of an export can only be done by users who have WRITE permission on the relevant export port type. (In general, possessing WRITE permission on a GFFS resource makes one an administrator of it.) When an attempt to force an export owner is made during export creation, the container tests whether the user is an administrator on the requested port type, and fails the request if the user has no admin access.

Since the exports force flag requires administrative rights, the grid is protected from rogue users who would make bogus exports as other users. But by providing this forcing feature, legitimate administrators can support their users by creating exports for them.
For example, assume one is logged in as an administrator of the lightweight port type on a container and one wants to make an export for “fred”. The administrator does not own any credential referencing fred, and so normally would not be allowed to create the export as fred. But by using the force flag, the admin can issue this command to export fred’s files:

```
# real command should all be on one line...
grid export --create --force "
--creator="/C=US/ST=Virginia/L=Charlottesville/O=Testing Grid/OU=Genesis II/SERIALNUMBER=urn:ws-naming:epi:12D42885-7D5C-7DED-5C1F-F280FD25183/CN=fred/CN=X509AuthnPortType" 
/resources/xsede.org/mason.iu.xsede.org/containers/mason-gffs 
local:/home/xd-fred/data grid:/home/xsede.org/fred/mason-data
```

The DN string provided must be an exact match with a DN listed in the grid-mapfile for the “forced” type of export creation to succeed.

### M.4.9. Configuration Files and Scripts

#### M.4.9.1. `export.properties` Configuration File

The file `export.properties` is used to define a particular GFFS container’s export behavior. The definitions supported in this file are shown below.

```
# The first value listed is the default value for the setting.
# Any paths shown refer to the default XSEDE installation paths.
export.Mechanism = ACL | ACLandChown | ProxyIO | Archive
export.AllowOverride = false | true
byteio.InUserHome = false | true
byteio.Storage = OwnedByGFFS | ChownToUser
export.GffsChownApp = /opt/genesis2-xsede/gffschown
export.ProxyIOApp = /opt/genesis2-xsede/proxyio.launcher
export.GridMapFile = /etc/grid-security/grid-mapfile
```

The `export.properties` file will be located via a similar mechanism to SDIACT-147 for container configuration files (see the activity page for more details at [https://software.xsede.org/jira/view/sdiact-147](https://software.xsede.org/jira/view/sdiact-147)). First `export.properties` is sought in the container state directory ($GENII_USER_DIR). If it is not located there, then it will be sought in the installation directory ($GENII_INSTALL_DIR).

In the following, the Unix user that is running the container will be referred to as “GffsUser”.

The `export.Mechanism` defines the overall method for implementing exports. It supports several values:

- **ACL** – Assume that `GffsUser` has sufficient permissions and can manage all exported files as the `GffsUser`. In other words, don’t mess with permissions. This is the current mechanism, which assumes that each user (or the system administrator) has set up any necessary extended access controls so that the `GffsUser` can read/write/etc. where it needs to. Created files will be owned by the GffsUser.
- **ACLandChown** – Similar to ACL mode, but requires that the `GffsUser` has `chown` permissions via `sudo` so that the GFFS container can `chown` created files as the required user. This will be done using the script `gffschown` (described in section M.4.9.3) which is root-owned and
which will be configured to only allow the GFFS container to chown from GffsUser to other users, rather than allowing any arbitrary user to chown. The chowned file must retain its original group and be RWX for that group, so that GffsUser can still manipulate it.

- **ProxyIO** – The gffs container fork/execs co-processes as individual users. Those processes handle all IO operations on behalf of the gffs container. The GffsUser must have sudo access to run the proxy IO process as any user that will be allowed to export files. The mapping from credentials to local user id will use the grid-mapfile file, and will be cached in memory. If the grid map changes on disk, then any existing cached entries will be discarded.

- **Archive** – Rather than using a file system, the export will interact with an archival storage system. *Note that this option is reserved for future implementation and does not exist at this time.*

**export.AllowOverride** – When a new export is created on the container the mechanism specified will be used unless over-ridden in the construction properties of the request. If this property is set to true then construction properties are allowed to override the container-specified mechanism. Currently, setting this flag to true will have no effect.

**byteio.Storage** – This property controls whether ByteIO files in the grid are kept in the ownership of the Unix user that owns them. This can support quotas based on the true owner of ByteIO files. The property supports two values:

- **OwnedByGFFS** – All ByteIO files are owned by the GFFS container’s Unix user. This is the current implementation and will be the default choice if the option is not provided.
- **ChownToUser** – Any newly created ByteIO file is stored in a subdirectory named after the grid user, as now, but in addition we will chown the file to make it owned by the Unix user associated with that grid user. Similarly to ACLandChown, this will use the script gffschown (discussed in section M.4.9.3).

**export.GffsChownApp** – This parameter defines the location of the GffsChown script. This is used for the ACLandChown mechanism as well as when the byteio.Storage option is set to ChownToUser. The details of the script will be discussed below in section M.4.9.3.

**export.ProxyIOApp** – This parameter specifies where the launcher script is located for the ProxyIO (a.k.a. sudo-based) export mechanism. This script is described further in section M.4.9.4.

**export.GridMapFile** – Defines the location of the grid-mapfile that maps DNs from user credentials to Unix users.

**M.4.9.2. gffs.exports Configuration File**

The GFFS container will read the gffs.exports file and use it to constrain the creation of exported directories. The file is consulted only for the ProxyIO export mechanism. The approach described below borrows concepts and format from /etc/exports.

*Important: The gffs.exports file concerns the creation of exports. It is checked only when creating exports, not when using existing exports.*
The `gffs.exports` file will first be sought in the container state directory (`$GENII_USER_DIR`) and then in the GFFS installation’s "lib" directory (`$GENII_INSTALL_DIR/lib`) as per the Unified Configuration model implemented in `SDIACT-147`.

If one wishes to use an overridden `gffs.exports` file, the first step should be copying the default version into the state directory for editing, e.g.:

```
    cp $GENII_INSTALL_DIR/lib/gffs.exports $GENII_USER_DIR
```

Each line in the `gffs.exports` file is either a comment or a record that specifies information about a particular file system path. Blank lines are ignored, comments can be made by starting a line with the hash mark (`#`), and long lines can be wrapped with a backslash (`\`). Each exported file system path should be on its own individual line.

**Important:** If there is no `gffs.exports` file found in the locations described above, then the GFFS will not allow a user to create any exports. The installation program for the GFFS provides a basic `gffs.exports` file which can be modified as desired, but it ships with all paths disabled for creating exports (for all users). This ensures that the admin has a chance to consider which paths are valid for users to export, before users can start creating exports.

The default `gffs.exports` file is shown below. It does not allow any user to create an export on any path.

```
# Path UserID [Options]
# Default exports file disallows exports on any path.
/ * n
```

Options: `r` = read, `w` = write, `n` = none

Before any exports can be created, the admin must explicitly allow paths by editing the `gffs.exports` file, such as by allowing exports to be created on the common home directories path:

```
/ * n
/home * rw
```

The following is a more full-fledged example of a `gffs.exports` file:

```
# Path UserID [Options]
/ * n
/users/tom/data1 tom.smith rw
/usr * r
/etc * n
/users/admin * n
/users/joe/public * r
/localtmp/sandbox * rw
```

Options: `r` = read, `w` = write, `n` = none

The options are indicated by a single character, where “`r`” indicates read access, “`w`” indicates write access, and “`n`” indicates no access. The “`n`” option cannot be combined with “`r`” or “`w`”. If no option is specified for a path, then this path is interpreted as having no permissions (“`n`”) for export creation.
Paths specified in the `gffs.exports` file are understood to also apply to all sub-paths. This can be overridden by defining permissions on a more specific path. For example, if there are restrictions on both “/etc” and “/etc/samba”, then the “/etc/samba” restriction will be used for an export on a path called “/etc/samba/secrets”. This configuration is shown below:

```
/etc * n
/etc/samba * rw
```

Further, the most specific user permission is used to determine access, when both a wildcard and a particular user are specified in the file. For example, the /etc folder may be specified as unavailable for any user to create exports on, except for the user “admin3” who is given read access, as shown below:

```
/etc * n
/etc admin3 r
```

The restrictions in gffs.exports just indicate that the GFFS container will not block an attempt to creation an export on a particular path. The underlying operating system may still prevent the action due to user-level access restrictions, which would need to be addressed by the system admin.

### M.4.9.3. `gffschown` – bash script

**Usage:** `gffschown UserId filepath`

A shell script is provided to implement the ACL and Chown export mechanism. In an RPM installation, this file will be owned by root and stored by default in `/opt/genesis2-xsede/gffschown` (for a GFFS installation using the XSEDE grid). Sudo access to execute this file as `root` must be established for the GffsUser. The default script provided by the Genesis II team will simply be a call through to the native `chown`. However, system administrators may choose to customize the script, for example, to put limits on the user ids that can be `chowned` to or to add criteria that must be checked (e.g. the script could disallow privileged IDs, or could only `chown` files that have been created in the last 30 seconds, or could only operate on empty files, etc). Note that the GFFS will only call this script immediately after creating a new file. This should under normal circumstances happen in less than a second, and certainly in less than 30 seconds.

Below is an example of a very simple gffschown script:

```
#!/bin/bash
# example usage: gffschown UserId /path/to/file/or/directory
# ensure that the container can still access the file as needed.
/bin/chmod g+rw "$2"
# actually perform the chown operation.
/bin/chown "$1" "$2"
```

See section M.4.7 for information on configuring the `/etc/sudoers` file to allow the GFFS container user to launch this script.

### M.4.9.4. `proxyio.launcher` – bash script
This script will be used by the GFFS container to start the I/O co-process running as the appropriate Unix user for a GFFS export. In ProxyIO export mode, each export has at most one co-process for each export that runs as the export owner’s corresponding Unix account. The co-process is not launched (by the proxyio.launcher script) until the export is actually used, and the co-process will continue to run until the export is unused for a certain timeout period. By default the script is located in “/opt/genesis2-xsede/proxyio.launcher” for an RPM-based installation based on the XSEDE grid. This script will be started by the container using sudo. The script merely launches a helper co-process provided with the GFFS installation (where the co-process is also implemented in Java). To prevent the GFFS container from calling arbitrary Java programs as arbitrary user identities, the container is restricted to calling this particular shell script, which can be tightly controlled by root and modified as necessary for any local administrative requirements.
N. Central Administrator’s Guide for the XSEDE Production Grid

This guide documents the steps that were taken to establish the XSEDE production GFFS grid and provides important maintenance steps for the running grid.

This guide integrates software and documentation delivered by “SDIACT-149: Incremental Genesis II Fixes and Enhancements”. This SD&I Activity builds on “SDIACT-123 - Integrated EMS and GFFS Increment 3 Update” and “SDIACT-126 - GFFS namespace design, policies, and support tools”. The guide is structured in three major sections: creating an XSEDE GFFS grid with 4 central containers, updating an XSEDE GFFS grid from the SDIACT-126 release of Genesis II to the SDIACT-149 release, and administrating the XSEDE GFFS grid using the latest tools.

N.1. Installation Support

Email help@xsede.org to report problems and/or ask questions.

N.2. Deployment Perspective

The central container setup process creates a system of 4 containers. The root container provides namespace services to users of GFFS and the root replica container serves as its backup. The primary STS container authenticates XSEDE users and the secondary STS container serves as its backup. These containers are depicted below.

N.3. Creating an XSEDE GFFS Grid of Four Central Containers

N.3.1. Deployment Prerequisites

N.3.1.1. Recommended Training Plan

XSEDE Operations administrators would benefit from reading the following documents before proceeding with any installation or administration steps described here:

SDIACT-126 GFFS Namespace Design:

This document, the XSEDE GFFS Central Administrator Guide.

Genesis Omnibus Reference sections C, D.4, D.8, F.1, F.2, G.1.10, G.2.[1-4].

N.3.1.2. Environment Requirements

Four (4) servers: 2 minimal configuration standalone locked-down servers for STS services, and 2 more robustly configured servers for Namespace services (see the SDIACT-126 GFFS Namespace Design document).
These are the official production grid DNS names for containers; test system host names will of course vary:

- **Root container**: gffs-1.xsede.org
- **Root replica container**: gffs-2.xsede.org
- **STS container**: sts-1.xsede.org
- **STS replica container**: sts-2.xsede.org

The 4 servers must have publicly accessible IP addresses.

Suggested server specifications for Namespace services root and root replica:

- **8 CPU cores of at least 2 Ghz each**.
- **16 gigabytes RAM (or 2 gigabytes of memory per CPU)**.
- **Root container should have 1 TB free on hard disk (to allow users to store files in home directories)**.
- **Root replica container should have 50 gigabytes free on hard disk (more space should be made available if admins plan to replicate any user home directories; currently this is not planned)**.

Example: XCG root container is on an 8 CPU host with 16 gigabytes of memory, and the CPUs are 2.4 Ghz each. Hyperthreading is enabled, yielding 16 apparent CPUs.

Suggested server specifications for STS services primary and replica:

- **4 CPU cores of at least 2 Ghz each**.
- **8 gigabytes RAM (or 2 gigabytes of memory per CPU)**.
- **50 gigabytes free on hard disk**.

Administrative Client: The configuration process also requires an administrative client installation. The host for this does not need to be very powerful, but it should have at least 2 gigabytes of RAM and 2 gigabytes free on the hard disk. The administrative client install does not need to be on a separate host, but it is required that it is a separate user account from any of the container installs.

XSEDE Certificates: Each host should have the official XSEDE certificates package installed. This can be acquired from http://software.xsede.org/security/xsede-certs.tar.gz and the certificates should be installed in the official location of /etc/grid-security/certificates.

**N.3.1.3. Host Certificates and Kerberos Keytabs**

Each of the four servers requires two X.509 host certificates in PFX format; one certificate is for TLS communication and one is for Administrative login on the container. These certificates must be signed by an XSEDE approved CA for the production system. This URL describes XSEDE security in general: https://www.xsede.org/security and this page describes the certificate service: https://www.xsede.org/security/certificates/. Contact help@xsede.org if you need help acquiring host certificates.
XSEDE will provide certificate files in PEM format (*.cer) based on your private key files (*.key), and the following steps will generate the necessary PFX files within a folder structure used by the remainder of the deployment process. Note that this process is geared for scriptability and testing; in the production grid, security concerns may require an entirely different process for managing admin certificates and passwords. The commands below may still be helpful in generating PFX files.

The steps below recommend a set of certificate Distinguished Names (DNs) for the production system; if a test system is being deployed, the real DNS hostnames of the container hosts should be used in the certificate DNs below instead. Regardless of the file names originally issued for the keys and certificates, please store them using the names below (i.e. tls-cert.* and admin.*).

Important: When downloading the certificate files after they are generated, use the choice for “as X509 Certificate only, Base64 encoded”, which will provide a PEM format file with a single certificate. Otherwise, the PEM files will need to be edited to remove intermediate certificates.

1. Create a folder hierarchy for each of the certificates to reside in to avoid name conflicts:

```bash
mkdir $HOME/gffs-certs; \
pushd $HOME/gffs-certs; \
mkdir gffs-1 gffs-2 sts-1 sts-2; \ 
popd
```

2. Primary root container (gffs-1.xsede.org):
   a. File names: tls-cert.key and tls-cert.cer; DN: /C=US/O=National Center for Supercomputing Applications/OU=Services/CN=gffs-1.xsede.org
   b. File names: admin.key and admin.cer; DN: /C=US/O=National Center for Supercomputing Applications/OU=Services/CN=gffs-1-admin.xsede.org
   c. Store all four files in: $HOME/gffs-certs/gffs-1
   d. For use in generating the final admin.pfx file, create a file with a single line containing the desired administrative password in: $HOME/gffs-certs/gffs-1/admin.pass

3. Replica root container (gffs-2.xsede.org):
   b. File names: admin.key and admin.cer; DN: /C=US/O=National Center for Supercomputing Applications/OU=Services/CN=gffs-2-admin.xsede.org
   c. Store all four files in: $HOME/gffs-certs/gffs-2
   d. For use in generating the final admin.pfx file, create a file with a single line containing the desired administrative password in: $HOME/gffs-certs/gffs-2/admin.pass

4. Primary STS container (sts-1.xsede.org):
   a. File names: tls-cert.key and tls-cert.cer; DN: /C=US/O=National Center for Supercomputing Applications/OU=Services/CN=sts-1.xsede.org
   b. File names: admin.key and admin.cer; DN: /C=US/O=National Center for Supercomputing Applications/OU=Services/CN=sts-1-admin.xsede.org
c. Store all four files in: $HOME/gffs-certs/sts-1

d. For use in generating the final admin.pfx file, create a file with a single line containing the desired administrative password in: $HOME/gffs-certs/sts-1/admin.pass

5. Replica STS container (sts-2.xsede.org):
   b. File names: admin.key and admin.cer; DN: /C=US/O=National Center for Supercomputing Applications/OU=Services/CN=sts-2-admin.xsede.org
   c. Store all four files in: $HOME/gffs-certs/sts-2
   d. For use in generating the final admin.pfx file, create a file with a single line containing the desired administrative password in: $HOME/gffs-certs/sts-2/admin.pass

5. After acquiring the official TLS certificates, each of the TLS and administrative certificates should be turned into a PFX file that is appropriate for usage by the GFFS.
   a. Given the directory structure and file names defined above, this step will generate the necessary files:

```bash
for i in gffs-1 gffs-2 sts-1 sts-2; do \
  pushd $HOME/gffs-certs/$i; \
  openssl pkcs12 -export -out ./tls-cert.pfx -inkey ./tls-cert.key -in tls-cert.cer -passout pass:container -name container; \
  openssl pkcs12 -export -out ./admin.pfx -inkey ./admin.key -in admin.cer -passout file:admin.pass -name admin; \
  popd; \
done
```

b. If the admin.pass file was not created due to security concerns, the "-passout file:admin.pass" can be omitted on the admin.pfx line to cause a manual password entry.

6. The two STS servers require Kerberos keytabs and service principals, which can be requested from help@xsede.org. Read the Kerberos Authorization section (Omnibus G.1.10) for more detailed information. We recommend requesting the following Kerberos principal names for the STS containers:
   a. File: sts-1.keytab; Principal: gffs-sts/STS-1.XSEDE.ORG@TERAGRID.ORG
   b. Store sts-1.keytab in: $HOME/gffs-certs/sts-1
   c. File: sts-2.keytab; Principal: gffs-sts/STS-2.XSEDE.ORG@TERAGRID.ORG
   d. Store sts-2.keytab in: $HOME/gffs-certs/sts-2

7. After receiving all 8 certificates and 2 keytab files and running the prior steps, create a package of these for use during the container deployment process below:

```bash
pushd $HOME ; tar -czf gffs-certs.tar.gz gffs-certs; popd
```

8. Keep the gffs-certs folder and archive file safe from exposure, as these keys are crucial security assets for the grid.

N.3.1.4. Establishing Environment Variables
In these instructions, the following variables are needed on each installation host:

- **GENII_INSTALL_DIR** should point to the Genesis II software installation directory.
- **GENII_USER_DIR** should point to the Genesis II state directory if not using the default location. The default location is $HOME/genesisII-2.0.
- **XSEDE_TEST_ROOT** should point to the top of the xsede_tools folder. This variable can be set automatically using the "prepare_tools.sh" script (examples are provided for each installation section below).

Additional variables may be needed for certain processes, such as when using the deployment generator, and they will be specified at the appropriate time.

Because environment variables disappear if the current bash shell is exited from, it is recommended to initialize them from the shell startup script (e.g. $HOME/.bashrc) in each account that will run Genesis II applications. (If the .bashrc approach is used, remember to log out and log back in again for the variables to take effect.) Example sets of the required variables will be provided below during the set up of each Genesis II installation.

### N.3.1.5. Verifying GFFS Port Is Open

This test should be executed for each container installation below.

GFFS Containers require a port number that is available and open in any firewall. Before installation, the intended port number (by default 18443) can be tested to see if it is in use with this command:

```
netstat -an | grep :18443
```

If there is a port conflict, then one can use a different port number for the container in question. The official XSEDE central container hosts should be dedicated to the GFFS to eliminate potential port conflicts.

### N.3.2. Time and Effort Estimates

Once appropriate hardware and operating systems are running for the 4 servers and all host certificates and keytabs have been acquired, we estimate the following effort:

Software deployment and configuration of the four central containers should take 2 full business days if already familiar with GFFS (i.e., expert) and 6 full business days if new to GFFS (i.e., novice).

Configuring automated namespace account maintenance procedures should take 3 hours. Running the full script initially can take more than 24 hours.

Ongoing daily maintenance will vary.

### N.3.3. Deploying the XSEDE GFFS Central Containers
The following deployment steps should be performed by an XSEDE Operations GFFS grid administrator. In these steps, we will reference the official host names for the XSEDE grid containers. If you are setting up a test system, then change any external host names as needed but all in-grid names should remain the same, based on the roles for the containers. For example, the secondary STS container should always reside in /resources/xsede.org/containers/sts-2.xsede.org, regardless of its real host name.

N.3.3.1. Step 1. Root Container Deployment

Official hostname “gffs-1.xsede.org” in production grid.

1. Transfer the gffs-certs.tar.gz file created in the prerequisite section to this host by a secure means, such as scp or sftp. Extract the file to “$HOME/gffs-certs”.
2. Download the source code from http://software.xsede.org/development/genesis2/sdiact-149/production_grid/sdiact149-installers/ The source code is in the genesis2*.tar.gz file. Extract the source code to a directory (for example $HOME/genesis-trunk).
3. Set the required environment variables appropriately for this host (see Establishing Environment Variables above). Here is an example set of variables for the gffs-1 host based on source code:

   # Set a specific state directory for the root container.
   export GENII_USER_DIR=$HOME/gffs-root-state
   # Point at Genesis II source code.
   export GENII_INSTALL_DIR=$HOME/genesis2-trunk
   # Load the basic tools and testing variables (only interactively).
   if [ "$TERM" != "dumb" -a -z "$PBS_ENVIRONMENT" ]; then
       source $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh \ 
       $GENII_INSTALL_DIR/xsede_tools/prepare_tools.sh
   fi

4. Configure Java according to the Omnibus H.1. Build the Genesis II source code according to Omnibus H.3.
5. Verify that the GFFS port is open (see Verifying GFFS Port Is Open above).
6. Use the GFFS deployment generator to bootstrap the root container described in the Omnibus (F.2, F.2.1, F.2.2), but with the following modifications:
   a. Do not download the source code newly, as it has just been built in prior steps.
   b. In step F.2.2.1 where the trust store is created, copy the gffs-1 tls-cert.pfx and admin.pfx files into the override_keys directory:

   cp $HOME/gffs-certs/gffs-1/tls-cert.pfx \ 
   $HOME/gffs-certs/gffs-1/admin.pfx \ 
   $XSEDE_TEST_ROOT/tools/deployment_generator/override_keys

c. Prepare the trusted-certificates folder with the UVa XCG CA key:

   pushd $XSEDE_TEST_ROOT/tools/deployment_generator; \

---

Genesis II Omnibus Reference Manual  Page 276
d. Prepare the myproxy certificates folder, using the official XSEDE certificates:

```
cp known-grids/uva_xcg_certificate_authority.cer
   trusted-certificates; \\
popd
```

e. Prepare the grid-certificates folders using the official XSEDE certificates:

```
cp /etc/grid-security/certificates/*.*.0 \\
$XSEDE_TEST_ROOT/tools/deployment_generator/myproxy-certs
```

f. The passwords.txt file should be configured with ‘container’ as the TLS identity password and with the contents of $HOME/gffs-certs/gffs-1/admin.pass as the administrative password.

7. Log in via administrative “keystore” login using the root container admin.pfx:

```
$GENII_INSTALL_DIR/grid keystoreLogin local:$HOME/gffs-certs/gffs-1/admin.pfx \\
  --password=$(cat $HOME/gffs-certs/gffs-1/admin.pass)
```

8. Add one user account for the administrator who will temporarily “own” the central containers. This account is a simple X.509-based identity that will reside on the bootstrap container rather than the STS container:
   a. Create a user “BootstrapUser” with the following commands, changing the password ("chosenPassword") as needed:

```
bash $XSEDE_TEST_ROOT/library/create-user-and-group.sh \\
   /users/xsede.org/BootstrapUser chosenPassword \\
   /groups/xsede.org/gffs-admins /home/xsede.org
```

   b. The user “BootstrapUser” is only needed during the installation of the four central containers and it will be superseded by the container administrator certificates, as described in later steps.

9. Enable service permissions for groups in the XSEDE namespace on the root container:

```
$GENII_INSTALL_DIR/grid script \\
   local:$XSEDE_TEST_ROOT/tools/xsede_admin/configure_central_container_permissions.xml \\
   /resources/xsede.org/containers/gffs-1.xsede.org
```

10. Backup the root container (according to the Container Backup and Restore section below).

**N.3.3.2. Step 2. Obtain Installer Package (Pre-Replication Version)**
1. Deliver deployment package built in prior step 1 (Omnibus F.2.2.6) to the UVa Genesis II developers (xcghelp@cs.virginia.edu) for bundling into installer. This package does not contain any private key files.
2. UVa staff will bind the configuration into Install4j and RPM install packages.
3. The container hosts will use the Genesis II RPM package rather than source-code. The root container is updated to use the RPM in Step 4 below.

N.3.3.3. Step 3. Administrative Client Deployment

To set up replication cleanly and to perform other administrative tasks while building the grid, we deploy the following client with admin privileges as follows:

1. On an account that is separate from any that will host the central containers, deploy the Genesis II client using the RPM package provided by UVa Genesis II developers (Omnibus D.7). 
2. Transfer the gffs-certs.tar.gz file created in the prerequisites to this host by a secure means, such as scp or sftp. Extract the file to “$HOME/gffs-certs”. (This step will not apply when using a previously created administrative client during the original grid deployment.)
3. Create a copy of the xsede_tools.cfg file for use on the admin client:

   ```
cp $GENII_INSTALL_DIR/xsede_tools/examples/xsede_tools.cfg $HOME/xsede_tools.cfg
   
   # Set state dir variable, but use default location.
   export GENII_USER_DIR=$HOME/.genesisII-2.0
   # Point at the tools configuration.
   export XSEDE_TOOLS_CONFIG_FILE=$HOME/xsede_tools.cfg
   # Below sets both GENII_INSTALL_DIR and XSEDE_TEST_ROOT:
   source /opt/genesis2-xsede/set_gffs_vars
   
   5. Run the following command to login with admin privileges to all central containers. This step is called “acquiring all administrative certificates” or “logging into all administrative keystores”.
      a. This login step is needed for most major operations on the four central containers:

   ```
$GENII_INSTALL_DIR/grid logout --all;
for i in gffs-1 gffs-2 sts-1 sts-2; do 
   $GENII_INSTALL_DIR/grid keystoreLogin local:$HOME/gffs-certs/$i/admin.pfx 
   --password=$(cat $HOME/gffs-certs/$i/admin.pass); 
   done

   b. Or, one can log into admin.pfx files individually if needed (repeat for each pfx file):

   ```
$GENII_INSTALL_DIR/grid keystoreLogin local:admin.pfx
```
6. Verify that all four admin credentials are listed (and that no other user credentials are listed) when running the whoami command below:

```
$GENII_INSTALL_DIR/grid whoami
```

**N.3.3.4. Step 4. Update Root Container to use GFFS RPM Package**

1. After the installer packages have been delivered, backup the container state by following the Backup/Restore section below and install the GFFS RPM package (Omnibus D.7).
2. After logging in to the appropriate account that runs the container (do not do this as root), perform the conversion from a source-based container to the Unified Configuration format (Omnibus D.8.7).
3. Update the script that sets GENII_INSTALL_DIR and XSEDE_TEST_ROOT to point at the new folder locations (e.g. modify $HOME/.bashrc). Since the RPM is in use now, this can be done by using the script set_gffs_vars to set both variables. Here is an example of the variables needed for the gffs-1 host based on running from the RPM package:

```
# Set a specific state directory for the root container.
export GENII_USER_DIR=$HOME/gffs-root-state
# Below sets both GENII_INSTALL_DIR and XSEDE_TEST_ROOT:
source /opt/genesis2-xsede/set_gffs_vars
```

4. Be sure to invoke this to set the variables in the current shell environment.
5. Move the source code and xsede tests folders to new locations; these should no longer be needed for running the root container. The source code folder should be kept for reference rather than discarded.
6. Backup the root container (according to the Container Backup and Restore section below).
7. Start up the root container and verify successful container startup (see “Container log files” section below):

```
$GENII_INSTALL_DIR/GFFSContainer start
```

**N.3.3.5. Step 5. Root Replica Container Deployment**


1. Transfer the gffs-certs.tar.gz file created in the prerequisites to this host by a secure means, such as scp or sftp. Extract the file to “$HOME/gffs-certs”.
2. Install the RPM package from Step 2 above (Omnibus D.7).
3. Set the required environment variables appropriately for this host (see Establishing Environment Variables above). Example variables for the gffs-2 host:

```
# Set a specific state directory for the primary sts container.
export GENII_USER_DIR=$HOME/gffs-state
# Below sets both GENII_INSTALL_DIR and XSEDE_TEST_ROOT:
source /opt/genesis2-xsede/set_gffs_vars
```

4. Verify that the GFFS port is open (see Verifying GFFS Port Is Open above).
5. Configure the root replica (according to Omnibus D.8.1) with the following command. (Test systems should have a different hostname than gffs-2.xsede.org):

```
bash $GENII_INSTALL_DIR/scripts/configure_container.sh gffs-2.xsede.org \
18443 BootstrapUser $HOME/gffs-certs/gffs-2/tls-cert.pfx container
```

6. Deploy the root replica container's administrative certificate (see Omnibus F.2.3 for more details):

```
cp $HOME/gffs-certs/gffs-2/admin.cer $GENII_USER_DIR/certs
rm $GENII_USER_DIR/certs/owner.cer $GENII_USER_DIR/certs/default-owners/owner.cer
```

7. Start the container and verify successful container startup (see “Container log files” section below):

```
$GENII_INSTALL_DIR/GFFSTContainer start
```

8. Configure a new resolver and replicate the top-level directories in the grid as documented in Omnibus G.2.5.5 using the Admin Client Deployment created in Step 3.

9. Update permissions on root replica with the following command using the Admin Client Deployment created in Step 3:

```
a. $GENII_INSTALL_DIR/grid script
local:$XSEDE_TEST_ROOT/tools/xsede_admin/configure_central_container_permissions.xml /
1 /resources/xsede.org/containers/gffs-2.xsede.org;
bash $XSEDE_TEST_ROOT/tools/xsede_admin/configure_resolvers.sh
```

10. Email the replicated-context.xml file to the Genesis II team at xchg.help@cs.virginia.edu so that the installers can be updated with the new replica information.

11. Backup both the root and the root replica containers (according to the Container Backup and Restore section below).

12. After receiving the updated installer packages from the Genesis II team, install the new RPM on the root replica host and on the administrative client host (RPM install documented in Omnibus D.7). This will ensure that the local grid client on the root replica host and on the administrative client will be replication-aware. Reconnect to the replicated context with the following command:

```
$GENII_INSTALL_DIR/grid connect "local:/opt/genesis2-xsede/deployments/current_grid/context.xml" "current_grid"
```

N.3.3.6. Step 6. Primary and Failover STS Container Deployments

Official hostnames sts-1.xsede.org and sts-2.xsede.org in production grid.

N.3.3.6.1. STS Host Installation

These steps must use the new installer created at the end of Step 5 above.
1. Transfer the gffs-cert.tar.gz file created in the prerequisites to this host by a secure means, such as scp or sftp. Extract the file to "$HOME/gffs-cert".
2. Install each STS container using the updated (replication aware) RPM package (Omnibus D.7) from prior step 5.
3. Set the required environment variables appropriately for this host (see Establishing Environment Variables above). Example variables for the sts-1 and sts-2 hosts:

```
# Set a specific state directory for the primary sts container.
export GENII_USER_DIR=$HOME/gffs-state
# Below sets both GENII_INSTALL_DIR and XSEDE_TEST_ROOT:
source /opt/genesis2-xsedexset/put_gffs_vars
```

4. Verify that the GFFS port is open on each STS container (see Verifying GFFS Port Is Open above).
5. Configure the STS container (according to Omnibus D.8.1) with the following commands (the first parameter, for hostname, should be adjusted if this is a test system):

sts-1:

```
bash $GENII_INSTALL_DIR/scripts/configure_container.sh sts-1.xsede.org \ 
18443 BootstrapUser $HOME/gffs-certsts-1/tls-cert.pfx container
```

sts-2:

```
bash $GENII_INSTALL_DIR/scripts/configure_container.sh sts-2.xsede.org \ 
18443 BootstrapUser $HOME/gffs-certsts-2/tls-cert.pfx container
```

6. Deploy the root replica container's administrative certificate (see Omnibus F.2.3 for more details):

sts-1:

```
cp $HOME/gffs-certsts-1/admin.cer $GENII_USER_DIR/certs
rm $GENII_USER_DIR/certs/owner.cer $GENII_USER_DIR/certs/default-owners/owner.cer
```

sts-2:

```
cp $HOME/gffs-certsts-2/admin.cer $GENII_USER_DIR/certs
rm $GENII_USER_DIR/certs/owner.cer $GENII_USER_DIR/certs/default-owners/owner.cer
```

7. Configure Kerberos authorization as described in Omnibus G.1.10 using the keytab files created in the Deployment Prerequisites section. Be sure to use sts-1.keytab on the STS-1 host and sts-2.keytab on the STS-2 host. For example, on STS-1 copy the keytab:

```
cp $HOME/gffs-certsts-1/sts-1.keytab $GENII_USER_DIR/certs
```

And then edit $GENII_USER_DIR/installation.properties to add two lines (use the real principal name that was requested in the second line):

```
gffs.sts.kerberos.keytab.TERAGRID.ORG=sts-1.keytab
```
8. Start both STS containers and verify successful container startup (see “Container log files” section below):

```
$GENII_INSTALL_DIR/GFFSContainer start
```

### N.3.3.6.2. Configuring STS Hosts

The following steps should be done after both STS containers have been installed:

1. Follow plan for migrating users and groups to STS container (Omnibus G.2.5.7) using the Admin Client Deployment created in Step 3. Be sure to run the tests described here, using your own XSEDE portal account as the user name.
2. Perform the steps below in the section called “Add a pattern-based ACL for MyProxy users”. This enables users to gain membership in the gffs-users group when logging in with xsedeLogin, which relies on the XSEDE MyProxy server.
3. Add permissions to the gffs-admins and gffs-amie group for both STS machines using the Admin Client Deployment created in Step 3.

```bash
for i in sts-1 sts-2; do \
   $GENII_INSTALL_DIR/grid script \n   local:$XSEDE_TEST_ROOT/tools/xsede_admin/configure_central_container_permissions.xm1 \n   /resources/xsede.org/containers/$i.xsede.org; \
done; 
bash $XSEDE_TEST_ROOT/tools/xsede_admin/configure_resolvers.sh
```

4. Backup the root, the root replica, and both STS containers (according to the Container Backup and Restore section below).

### N.3.3.6.3. Step 7. Finalizing the Setup

At this point, all four central containers have been configured.

1. The BootstrapUser grid user can be deleted now (Omnibus G.1.5).
2. It is very important to keep a copy of the gffs-certs.tar.gz for use by grid administrators! But the copies of gffs-certs.tar.gz and the gffs-certs folders can be deleted from the central containers.
3. The administrative client host can be decommissioned if it will not be used by grid administrators.
4. To establish the XSEDE user accounts, proceed to the administrative activity “Creating XSEDE User Accounts” below.

### N.4. Updating the XSEDE GFFS Production Grid to SDIACT-149

The Activity 126 version of the Genesis II software has been deployed as the XSEDE production grid. The following four sections update the central containers to the latest Activity 149 version of Genesis II using RPM install packages. The first section updates the containers to use RPMs, the second re-enables the storage of files on the root container for users’ home directories, the third
section provides pattern-based login to the gffs-users group for MyProxy users, and the fourth section configures the automatic certificate update process for GFFS clients and containers.

Cautionary note for SDIACT-126 to SDIACT-149 upgrade only: If there are any queued jobs active in the system, they should be allowed to run to completion or aborted before performing the upgrade process. This is due to serialization changes in the UNICORE-7 SecurityLibrary. Upgrades from SDIACT-149 to later versions are not affected. If necessary, the jobs can also be stopped and removed after the upgrade, but any upgraded jobs are left in a non-functional state due to the serialization change.

N.4.1. Update Installations to RPM Packages

1. Backup all container accounts that will be affected by the upgrade. This includes the 4 central containers, and any other containers that are an essential part of the grid (such as a queue container).

2. Install the activity 149 RPM Package on each central container and on the admin client. The production grid deliverables are located here:

   http://software.xsede.org/development/genesis2/sdiact-149/production_grid/sdiact149-installers/

   The testing grid deliverables are located here:


   It may be necessary to install missing dependencies for the RPM, such as expect. This can be done with "yum install expect" (where "expect" should be replaced with whatever package names are reported as missing). This step will be done as root or sudo.

3. On the root container, perform the container conversion step documented in Step 4 of the central container configuration steps above. This step will be done as the container-owning user and not as root.

4. On the root replica and both STS containers, log in as the container owning user and perform the container conversion process documented in the Omnibus D.8.2 using the following approach: when prompted by the script, do not copy the deployments folder during the conversion process. These containers do not have a specialized deployment (and thus are not configured like the root container). An example command line for conversion of these containers will resemble this (without the stop or depcopy flags):

   ```bash
   $GENII_INSTALL_DIR/scripts/convert_container.sh {genesis2-trunk}
   ```

5. On any existing administrative client installations, switch the GENII_INSTALL_DIR variable from pointing at the interactive installer (usually $HOME/GenesisII) to point at the RPM package in /opt/genesis2-xseed instead. The old installer directory can be moved out of the way or removed afterwards.

   a. If a completely new administrative client installation is needed, then it should be configured using the process described in Step 3 in the initial deployment section above.

6. On all container and client hosts in the system, update the environment variables to point at the new installation location and test suite, as documented in Step 4.3 of the root container conversion steps above.
7. Configure all of the central containers and the administrative client to use the official XSEDE certificates directory according to Omnibus F.2.4.

N.4.2. Enable Users to Store Files on Root and Root Replica Containers

1. On both the root container and root replica container, execute the following command:

   ```bash
   chmod -R u+rw $GENII_USER_DIR/rbyteio-data
   ```

2. This enables file storage for Random ByteIO files in the grid, allowing users to store files in their home directories.

N.4.3. Add a Pattern-based ACL for MyProxy users

From the administrative client for the grid (created during step 3 of Central Container setup), perform the following steps:

1. Log in with all administrative certificates using each of the 4 central container administrative keypairs. For example:

   ```bash
   $GENII_INSTALL_DIR/grid keystoreLogin local:$HOME/gffs-1-admin.pfx
   ```

   ...for each admin.pfx file available.

2. Configure the gffs-users group to allow user membership based on the XSEDE MyProxy servers. (The first line is for the older teragrid server, which will eventually be replaced. The second line is for the newer xsede.org MyProxy server):

   ```bash
   $GENII_INSTALL_DIR/grid chmod /groups/xsede.org/gffs-users "--pattern='O=National Center for Supercomputing Applications' +rx" local:$GENII_INSTALL_DIR/deployments/current_grid/security/myproxy-certs/f2e89fe3.0
   $GENII_INSTALL_DIR/grid chmod /groups/xsede.org/gffs-users "--pattern='O=National Center for Supercomputing Applications' +rx" local:$GENII_INSTALL_DIR/deployments/current_grid/security/myproxy-certs/c36f6349.0
   ```

3. Test the new configuration (still logged in with administrative certificates):
   a. Edit the gffs-users ACL to remove Read and Execute permissions for your own portal account:

      ```bash
      $GENII_INSTALL_DIR/grid chmod /groups/xsede.org/gffs-users 0 /users/xsede.org/MyPortalAccount
      ```

   b. Ensure that you have a link under your user account for gffs-users:

      ```bash
      $GENII_INSTALL_DIR/grid ls /users/xsede.org/MyPortalAccount
      ```

      Above should show gffs-users under the account, but if not, add the link:

      ```bash
      $GENII_INSTALL_DIR/grid ln /groups/xsede.org/gffs-users
      ```
c. Log out of all credentials:

```bash
$GENII_INSTALL_DIR/grid logout --all
```

d. Log in as your XSEDE portal account:

```bash
$GENII_INSTALL_DIR/grid xsedeLogin --username=MyPortalAccount
```

e. Show credentials after logging in:

```bash
$GENII_INSTALL_DIR/grid whoami
```

f. If the login succeeded and you are in the gffs-users group, then the MyProxy pattern-based ACL configuration is working.


Follow the process described in Omnibus F.2.6.1 to create a cron job for the certificate package upload on any appropriate client or container host that has the official XSEDE certificates installed. As long as `fetch-crl` is run regularly on this host, then the cron job will ensure that the CRL files for the grid will be automatically updated from the official certificates.

**N.5. XSEDE GFFS Central Container Administrative Procedures**

**N.5.1. Container Startup/Shutdown**

**N.5.1.1. Root container still based on source:**

Start container:

```bash
nohup bash $GENII_INSTALL_DIR/runContainer.sh &>/dev/null &
```

Stop container:

```bash
bash $XSEDE_TEST_ROOT/library/zap_genesis_javas.sh
```

List running containers:

```bash
bash $XSEDE_TEST_ROOT/library/list_genesis_javas.sh
```

**N.5.1.2. Containers still based on SDIACT-126 interactive installer:**

Default installation location: `$HOME/GenesisII`

Start container:

```bash
$GENII_INSTALL_DIR/XCGContainer start
```

Stop container:

```bash
$GENII_INSTALL_DIR/XCGContainer stop
```
Get container status:

\$GENII\_INSTALL\_DIR/XCGContainer status

N.5.1.3. Containers based on SDIACT-149 RPM and interactive installs:

Default installation location: /opt/genesis2-xsede

Start container:

\$GENII\_INSTALL\_DIR/GFFSContainer start

Stop container:

\$GENII\_INSTALL\_DIR/GFFSContainer stop

Get container status:

\$GENII\_INSTALL\_DIR/GFFSContainer status

N.5.2. Revert/Undo/Rollback

Before any major change is made to the deployed grid, backups of all four central containers should be made according to the Backup / Restore section below. This allows an unsuccessful modification to be rolled back if necessary.

If a serious problem has occurred during a particular container installation, the rollback procedure is to delete the state directory of that container (in $GENII\_USER\_DIR), to stop all other containers, and to restore clean versions of all containers that were set up prior to the problematic container. After that, the other containers can be restarted, and the current container install can proceed again.

N.5.3. Container Backup and Restore

Backing up a container’s state is documented in Omnibus G.2.3.

There is also a backup script in the XSEDE Tool suite (it produces the backup file in the current working directory):

bash $XSEDE\_TEST\_ROOT/library/backup_container_state.sh

Restoring a container’s state is documented in Omnibus G.2.4.

There is also a restore script in the XSEDE Tool suite (this wipes out the existing contents in $GENII\_USER\_DIR):

bash $XSEDE\_TEST\_ROOT/library/restore_container_state.sh {backupFile}

It is highly recommended that the four central servers be backed up on a regular basis.

The root and root replica containers should be backed up at non-overlapping times to maintain grid availability.
The STS and STS replica containers should also be backed up at non-overlapping times.

While any of the four central containers is shut down, administrators should not add or remove accounts, as replication is not fully functional if either the primary or the replica location is unavailable.

**N.5.4. Creating XSEDE User Accounts**

After the four central containers are deployed and operational, the following procedure will authorize XSEDE allocated users to use GFFS on sts-1.xsede.org.

Prerequisites:

The machine designated as the primary STS container will need the Perl prerequisites described in:

```
$XSEDE_TEST_ROOT/tools/xsede_admin/README-manage_gffs_namespace_accounts.txt
```

The `manage_gffs_namespace_accounts` script automates the process of adding XSEDE users to GFFS and can be configured to remove users from the GFFS that are no longer valid, though it doesn’t remove them by default (see the SDIACT-126 GFFS Namespace design document). This script performs the following tasks automatically:

- Retrieves all XSEDE allocated usernames from XDCDB.
- Creates missing `/users/xsede.org/{xsede_username}` STS resource entries, `/home/xsede.org/{xsede_username}` as needed and adds users to gffs-user group.
- Can move (unlink and relink) inactive `/users/xsede.org/{xsede_username}` STS resource entries to `/users/deleted.xsede.org/{xsede_username}` and `/home/xsede.org/{xsede_username}` directories to `/home/deleted.xsede.org/{xsede_username}` and removes user from groups

This is disabled by default and should remain so until the policy decision is made to revoke inactive user access to GFFS.

Can be run on the command line with `--testing` and `--debug` flags (which creates only 10 new accounts starting with ADMIN_ACCOUNTS first).

The following steps should be run from the Admin Client setup, logged in with all administrative PFX keystores, to add a user for executing the `manage_gffs_namespace_accounts` script on sts-1.xsede.org:

1. Create a new user XsedeAccountAdmin and add it to the gffs-amie group (replacing chosenPassword below with your choice):

```
$GENII_INSTALL_DIR/grid script local:$XSEDE_TEST_ROOT/library/create_one_user.xml
/resources/xsede.org/containers/sts-1.xsede.org XsedeAccountAdmin
/users/xsede.org/XsedeAccountAdmin chosenPassword gffs-amie /groups/xsede.org/gffs-amie /home/xsede.org /users/xsede.org
```

2. Add XsedeAccountAdmin to the gffs-users group:
To install `manage_gffs_namespace_accounts`, login into the primary STS container and run the following steps:

3. Ensure the Perl prerequisites described above are installed.

4. Login as the XsedeAccountAdmin by executing the following command and entering the password when prompted. This requests that the login duration be set for 2 days to allow sufficient time for the account creation script to run:

   ```bash
   $GENII_INSTALL_DIR/grid login --username=XsedeAccountAdmin --validDuration=2d
   ```

5. Set three variables either in the environment or at the top of the `manage_gffs_namespace_accounts` script: `$DBUSER`, `$DBPASSWD`, and `$GENII_INSTALL_DIR`. The values `$DBUSER` and `$DBPASSWD` provide limited access to the XDCDB and should be provided by SD&I during handoff to Operations. Afterwards, please contact help@xsede.org if you need help acquiring the values.

6. Test the script by running the below command which will create 10 user accounts (by default):

   ```bash
   $XSEDE_TEST_ROOT/tools/xsede_admin/manage_gffs_namespace_accounts --debug --testing
   ```

7. Verify new user directories appear under `/users/xsede.org/` and `/home/xsede.org/`

8. Re-run script to create the remainder of the user accounts as follows:

   ```bash
   $XSEDE_TEST_ROOT/tools/xsede_admin/manage_gffs_namespace_accounts
   ```

9. The first time this script is executed, it will create 9000+ accounts for current XSEDE users which will take longer than 24 hours to run. The first few lines of the script output should print the total number of accounts the script will create, e.g.,

   ```
   Allocated user count from database = 9018
   ```

To verify all user accounts have been created upon script completion, type:

```bash
$GENII_INSTALL_DIR/grid ls /users/xsede.org | wc -l
```

10. The script should be installed in a cron job to keep the user accounts up to date. It is recommended that the cron job run on the primary or secondary STS container. The cron job can leverage the container’s admin.pfx file, which will give it permission to create STS entries. Below is a sample crontab entry that can be modified in accordance with Operations’ security policies:

   ```
   DBUSER={user}
   DBPASSWD={passwd}
   GENII_INSTALL_DIR=/opt/genesis2-xsede
   # below should all be on one line.
   0 0 * * * ( $GENII_INSTALL_DIR/grid login --username=XsedeAccountAdmin --password=chosenPassword --validDuration=1d;
   ```
Note: Due to the large potential size of the "gffs-users" group, membership is granted using a pattern-based ACL to all MyProxy-authenticated users in the grid. Thus users are not given direct permissions on the "gffs-users" ACLs. The "link-xsede-user.xml" script (in $XSEDE_TEST_ROOT/tools/xsede_admin) handles this as a special case, and that script is used by the manage_gffs_namespace_accounts script.

N.5.5. Container log files

By default, the log file for a container can be found in $HOME/.GenesisII/container.log.

When starting up a container, look for the phrase “Done restarting all BES Managers” in the log file to make sure the container started up successfully.

N.5.6. User and group management

The following actions require membership in the gffs-amie group unless otherwise noted.

N.5.6.1. Manually create a new XSEDE GFFS user:

```
$GENII_INSTALL_DIR/grid script \n local:$XSEDE_TEST_ROOT/tools/xsede_admin/create-xsede-user.xml {XSEDE-User-Name}
```

Note: this is the same command used internally by manage_gffs_namespace_accounts above.

N.5.6.2. Manually remove an XSEDE GFFS user:

```
$GENII_INSTALL_DIR/grid script \n local:$XSEDE_TEST_ROOT/tools/xsede_admin/delete-xsede-user.xml \n {XSEDE-User-Name}
```

Note: this command moves namespace entries to “deleted” directories rather than actually deleting anything.

N.5.6.3. Manually add an XSEDE user to a group:

Modifying the administrative group gffs-admins requires being a member of gffs-admins (or logging into all administrative keystores for the grid’s central containers).

```
$GENII_INSTALL_DIR/grid script \n local:$XSEDE_TEST_ROOT/tools/xsede_admin/link-xsede-user-to-group.xml \n {XSEDE-User-Name} {XSEDE-Group-Name}
```

N.5.6.4. Manually remove an XSEDE user from a group:
Modifying a group in general requires membership in gffs-admins (or a login with all administrative keystores for the grid’s central containers). Additionally, members of gffs-amie can modify membership in the gffs-users group.

```bash
$GENII_INSTALL_DIR/grid script \
local:$XSEDE_TEST_ROOT/tools/xsede_admin/unlink-xsede-user-from-group.xml \
{XSEDE-User-Name} {XSEDE-Group-Name}
```

N.5.6.5. Manually creating a new admin group:

Must also have membership in gffs-admins group; if not yet a member of the gffs-admins group, run from Admin Client created during Step 3 of XSEDE Central Deployment:

```bash
$GENII_INSTALL_DIR/grid script \
local:$XSEDE_TEST_ROOT/tools/xsede_admin/create-sp-admin-group.xml \
{service-provider-name}
```

N.5.7. Prepare for Service Provider GFFS Deployment

Before an SP administrator can deploy a GFFS container, an XSEDE Operations GFFS namespace (central) administrator with gffs-admin privileges should perform the following steps to define the SP GFFS admin role, grant it to SP GFFS admins, and create the directory structure for linking SP resources. In the commands below, the `{service-provider-name}` is the short name for an SP, such as nics or sdsc. The `{resource-name}` is the specific resource being managed, such as “alamo” or “kraken”.

1. If the group gffs-admins.{service-provider-name} does not already exist, create it using the command:

```bash
$GENII_INSTALL_DIR/grid script \
local:$XSEDE_TEST_ROOT/tools/xsede_admin/create-sp-admin-group.xml \
{service-provider-name}
```

2. If the SP GFFS admin is not already added to their gffs-admins.{service-provider-name} group, add them using the command:

```bash
$GENII_INSTALL_DIR/grid script \
local:$XSEDE_TEST_ROOT/tools/xsede_admin/link-xsede-user-to-group.xml \
{username} gffs-admins.{service-provider-name}
```

3. Create the directory structure for each resource:

```bash
$XSEDE_TEST_ROOT/tools/xsede_admin/configure_sp_resource_tree.sh -t -i \
gffs-admins.{service-provider-name} -n {resource-name}.{service-provider-name}.xsede.org
```

N.5.7.1. Replicating Service Provider GFFS Resources
Once an SP administrator has deployed a GFFS container, an XSEDE Operations GFFS namespace (central) administrator with gffs-admins privileges should perform the following steps to replicate the SPs resources:

1. After the SP admin has linked in all their containers and queues, run the following command to replicate the resource tree to the Root Replica Container:

   ```bash
   $XSEDE_TEST_ROOT/tools/xsede_admin/configure_sp_resource_tree.sh -r -i \
   gffs-admins.(service-provider-name) -n {resource-name}.{service-provider-name}.xsede.org
   ```

   Note, that you can ignore the errors “no EPI”, “EndpointIdentifier not found”, and “Type does not support replication”.

N.5.8.  **Link an External Grid into XSEDE Grid**

A series of link commands as well as the EPR file of the root of the external grid namespace is needed to link an external grid into the XSEDE namespace. The “XSEDE Tests and Tools Suite” contains the EPR file for UVA’s XCG grid as well as an example script for linking it into the XSEDE namespace. Modify it accordingly for other external grids. The example script links UVA’s XCG grid and can be called as follows:

   ```bash
   $XSEDE_TEST_ROOT/tools/xsede_admin/link-grid-xcg.sh
   ```

   To verify the links worked, you should see about 100 accounts when you run the following command:

   ```bash
   $GENII_INSTALL_DIR/grid ls /users/xcg.virginia.edu
   ```

   To unlink the XCG namespace from the XSEDE namespace, use the following command:

   ```bash
   $GENII_INSTALL_DIR/grid script local:$XSEDE_TEST_ROOT/tools/xsede_admin/unlink-grid-xcg.xml
   ```

Create a global queue

To create a global queue under the “/resources/xsede.org/queues” path, execute the following steps with suitable values for the variables used below:

- **QUEUE_CONTAINER**: The name of the container where the queue will reside, e.g. BootstrapContainer.
- **QUEUE_NAME**: The name for the new queue, which must be unique within the queues directory. Naming by function may be helpful, such as “tutorial-queue” or “long-running”.
- **SP_NAME**: The name of the service provider or institution that provides one or more BES for running jobs.
- **U6_BES_NAME**: The name of the UNICORE BES being added to the queue. Multiple resources can be added to the queue by repeating the steps below to add the resource (given different values of the variable).
• SLOT_COUNT: The number of slots to configure the resource for, where the number of slots controls how many concurrent jobs may run on the resource at a time.

1. Create the queue {queue_name} using the following command:

   a. `$GENII_INSTALL_DIR/grid create-resource \
      /resources/xsede.org/containers/$QUEUE_CONTAINER/Services/QueuePortType \
      /resources/xsede.org/queues/$QUEUE_NAME`

2. Allow users to run jobs on the queue using the following command. The group “gffs-users” below could be replaced with a different group if desired:

   a. `$GENII_INSTALL_DIR/grid chmod /resources/xsede.org/queues/$QUEUE_NAME +rx \
      /groups/xsede.org/gffs-users`

3. For each resource you want to add into the queue, run the following commands. The first command links the resource into the queue and the second sets the number of slots. Adjust the U6_BES_NAME and SLOT_COUNT as needed for the different BES resources to be added:

   a. `$GENII_INSTALL_DIR/grid ln \
      /resources/xsede.org/$SP_NAME/bes-containers/$U6_BES_NAME \
      /resources/xsede.org/queues/$QUEUE_NAME/resources/${SP_NAME}-${U6_BES_NAME}; \
      $GENII_INSTALL_DIR/grid qconfigure /resources/xsede.org/queues/$QUEUE_NAME \
      $U6_BES_NAME $SLOT_COUNT`

O. References


P. Document History

<table>
<thead>
<tr>
<th>Relevant Sections</th>
<th>Version</th>
<th>Date</th>
<th>Changes</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entire Document</td>
<td>0.8</td>
<td>04/23/2012</td>
<td>Document Plan</td>
<td>XSEDE@UVa</td>
</tr>
<tr>
<td>Entire Document</td>
<td>1.1</td>
<td>05/21/2012</td>
<td>Plan + Deployment Section</td>
<td>XSEDE@UVa</td>
</tr>
<tr>
<td>Entire Document</td>
<td>1.9</td>
<td>05/30/2012</td>
<td>Containers, BES, Accounting</td>
<td>XSEDE@UVa</td>
</tr>
<tr>
<td>Entire Document</td>
<td>2.4</td>
<td>06/04/2012</td>
<td>Testing, Replication</td>
<td>XSEDE@UVa</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Management, Replication</td>
<td></td>
</tr>
<tr>
<td>Section Description</td>
<td>Date</td>
<td>Notes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------------------</td>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entire Document</td>
<td>3.4</td>
<td>06/08/2012 Deliverable Prototype, needs campus bridging</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entire Document</td>
<td>3.7</td>
<td>06/18/2012 Content complete, needs readability edits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entire Document</td>
<td>4.8</td>
<td>06/27/2012 Final Deliverable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Various Sections, esp. H and F.6.3</td>
<td>4.9-6.6</td>
<td>Through 05/21/2013 Updates for XSEDE Increment 3 and Ongoing Development</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minor edits plus new section G.1.10</td>
<td>6.7-6.8</td>
<td>5/23/2013 Addition of Kerberos authorization configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sections F.6.3, G.1.10, F.7, F.8</td>
<td>6.9</td>
<td>6/14/2013 Update for Unicore Configuration Steps, Kerberos configuration, grid tls and signing certificate descriptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sections F.2.* and F.6.3</td>
<td>7.0, 7.1</td>
<td>6/28/2013 Changes for new XSEDE namespace definition, updated U6 BES info, additional info re certificates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sections K.6.2.9, H.3, H.2, G.2.5.6, F.2.2, B.4, G.2.5.5, H.4.8, F.7 =&gt; F.3.4, F.8 =&gt; F.3.5, G.2.5.7</td>
<td>7.2</td>
<td>7/9/2013 Additions / corrections for SDIACT 126.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sections F.2.2.1.1, G.2.5.7</td>
<td>7.3</td>
<td>7/10/2013 More updates for SDIACT 126.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Section G.2.5.5</td>
<td>7.4</td>
<td>7/11/2013 Fixes for top-level replication steps.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sections F.2.1, F.2.2, G.2.5.5</td>
<td>7.5</td>
<td>7/12/2013 Additional edits from SDIACT126 testing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Section G.2.5.7</td>
<td>7.6</td>
<td>7/12/2013 Fix for STS replication.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Section G.2.5.5-G.2.5.7</td>
<td>7.7</td>
<td>7/16/2013 Simplification of STS replication, addition of policy and recursion for users/groups.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sections G.2.5.5, G.2.5.7</td>
<td>7.8</td>
<td>7/17/2013 Added variable to clean up code listings.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sections H.3, H.4.6, E.3.2</td>
<td>7.9</td>
<td>7/18/2013 Revised developer notes, added details for extended ACL exports.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sections K.6.2.9, F.6.3, H.4.3.3, H.5, H.6, F.3.6</td>
<td>8.0</td>
<td>7/31/2013 Updated details for SDIACT 126, added info re eclipse and debug tools, new material re quotas.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Date</td>
<td>Description</td>
<td>Author</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td>G.2.5.7</td>
<td>8.1</td>
<td>Revised approach for STS migration.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>G.2.5.5, G.2.5.7</td>
<td>8.2</td>
<td>Simplified approach for STS migration.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>G.2.5.7</td>
<td>8.3</td>
<td>Improved redundancy for STS configuration.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>G.2.5.5</td>
<td>8.4</td>
<td>Tweak to not recurse in top-level replication of groups and users.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>G.2.5.5, G.2.5.7, G.3</td>
<td>8.5</td>
<td>Group replication steps, wording improvements.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>I.1, G.4.8</td>
<td>8.6</td>
<td>Reflecting xsede tools and tests moving into gffs codebase.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>Entire document</td>
<td>8.7</td>
<td>Unifying method of referring to xsede_tests and GFFS install dir. Removed usage of tilde for $HOME. Clarified G.2.5.7 re which STS container is involved. Other clean-ups.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>J.1, E.6</td>
<td>8.8</td>
<td>Added FAQ item for null pointer issue, added client-ui document from Vana as new section.</td>
<td>Chris Koeritz, Vanamala Venkataswamy</td>
<td></td>
</tr>
<tr>
<td>Entire document</td>
<td>8.9</td>
<td>Added bookmarks to persist through into html version for linking.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>TOC for Figures, Section E.6</td>
<td>9.0</td>
<td>Fixed odd formatting around figures that was only revealed in html conversion.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>D, K</td>
<td>9.1</td>
<td>Updated installer documentation, cleaned up some older material in arcana.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>F.3.7, D.8.5, D.8.7</td>
<td>9.3</td>
<td>Added Vana’s DB tools and DB compaction tutorial, updated the instructions for how to use the deployment override RPM, added section for how to create new deployment override RPMs.</td>
<td>Chris Koeritz</td>
<td></td>
</tr>
<tr>
<td>Sections</td>
<td>Page</td>
<td>Date</td>
<td>Description</td>
<td>Author</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>E.3.1.2.3, D.8.6.5, D.8.2, G.5.1, I.2.1, F.2.3.4, K.6.2.9, E.2.2.6, G.1.1.10</td>
<td>9.4</td>
<td>4/21/2014</td>
<td>Modified text regarding copying files out of grid, added info about deployments folder, added section about converting source-based containers, added section about grid interconnect, added note for new testing config file pointer variable, added sanity check after container link, fixed error in certificate copy line, new material for InCommon authent.</td>
<td>Chris Koeritz</td>
</tr>
<tr>
<td>G.1.11</td>
<td>9.5</td>
<td>6/6/2014</td>
<td>Revised users path for incommon.</td>
<td>Chris Koeritz</td>
</tr>
<tr>
<td>D.8.<em>, I.</em></td>
<td>9.6</td>
<td>8/26/2014</td>
<td>Updated default install paths, removed text about using deprecated deployment override RPM, updated source-based container conversion process, updated references to inputfile.txt to use new config file name xsede_tools.cfg, cleaned old references to xsede_tests (became xsede_tools).</td>
<td>Chris Koeritz</td>
</tr>
<tr>
<td>F.2.2.*, H.1, H.3, F.2.3</td>
<td>9.7</td>
<td>9/24/2014</td>
<td>Updated usage of deployment generator, revised java and build instructions, dropped old text re deployment generator, added info re admin certs, assorted text cleaning.</td>
<td>Chris Koeritz</td>
</tr>
<tr>
<td>H.1, F.2.6, F.6.3</td>
<td>9.8</td>
<td>10/8/2014</td>
<td>Minor wording changes, updated content for deployment process, additional Kerberos configuration content, CRL section, caution re using right tls cert for unicore.</td>
<td>Chris Koeritz</td>
</tr>
<tr>
<td>F.2.4, E.3.3.2, F.6.3, I.1.4, F.2.6, J.2.3</td>
<td>9.9</td>
<td>12/2/2014</td>
<td>Updated customization section re using RPM install, FUSE install info, xsede test dependencies, revised CRL section,</td>
<td>Chris Koeritz</td>
</tr>
<tr>
<td>E.3.2.1, E.3.2.3, D.8.1, F.3.1, F.2.4, E.6, G.2.3, G.2.4</td>
<td>10.0</td>
<td>01/27/2015</td>
<td>Added information about export permissions on port type, repaired broken references, added more info about linking a</td>
<td>Chris Koeritz</td>
</tr>
<tr>
<td>Section</td>
<td>Date</td>
<td>Added/Updated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>---------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E.3.3</td>
<td>02/09/2015</td>
<td>Fixed syntax for fuse mount with sandbox. Chris Koeritz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A, B, L, M, K</td>
<td>03/24/2015</td>
<td>Improved grid command info, switched to bare grid command instead of using GENII_INSTALL_DIR prefix everywhere, moved xscript reference to own appendix, moved doc history to end of doc, added welcome section. Chris Koeritz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M, N, E.3.2, J, H</td>
<td>08/21/2015</td>
<td>Added xede central admin guide as appendix (allowing all references to become internal), updated exports guide after sdiact-175 edits, moved exports guide to own appendix since numbering became abusive, left E.3.2 with basic cookbook info and reference to real exports appendix, cleaned up central admin guide for section headings and code boxes, various other cleaning and new material. Chris Koeritz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J.2.4</td>
<td>10/02/2015</td>
<td>Added FAQ item for replacing an expired container certificate, added missing anchors for faq items. Chris Koeritz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H.4</td>
<td>12/10/2015</td>
<td>Eclipse configuration updated. Chris Koeritz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>