GT 5.2.2 GRAM5: System Administrator's Guide
Introduction

This guide contains configuration information for system administrators working with GRAM5. It describes procedures typically performed by system administrators, including GRAM5 software installation, configuration, testing, and debugging. Readers should be familiar with the GRAM5 Key Concepts to understand the motivation for and interaction between the various deployed components.
## List of Tables

1.1. GRAM5 LRM Adapters ................................................................. 2  
3.1. Gatekeeper Configuration Path .................................................. 5  
3.2. Scheduler Event Generator Configuration Path ................................ 6  
3.3. Job Manager Configuration Path .................................................. 7  
3.4. GRAM5 Log Levels ................................................................... 7  
3.5. LRM Adapter Configuration Path .................................................. 8  
3.6. GRAM Audit Configuration Path .................................................... 10  
3.7. Audit Configuration Attributes ...................................................... 11
List of Examples

2.1. Using globus-gatekeeper-admin to set the default service ................................................................. 4
2.2. Using globus-scheduler-event-generator-admin to disable a SEG module ........................................... 4
Chapter 1. GRAM5 Installation

1. Introduction

The Globus Toolkit provides GRAM5: a service to submit, monitor, and cancel jobs on Grid computing resources. In GRAM5, a job consists of a computation and, optionally, file transfer and management operations related to the computation. Some users, particularly interactive ones, benefit from accessing output data files as the job is running. Monitoring consists of querying for and/or subscribing to status information, such as job state changes.

GRAM5 relies on GSI C mechanisms for security, and interacts with GridFTP services to stage files to compute resources. Please see their respective Administrator's guides for information about installing, configuring, and managing those systems. In particular, you must understand the tasks in Installing GT and install the basic GRAM5 packages, and complete the tasks in Basic Security Configuration.

2. Planning your GRAM5 installation

Before installing GRAM5 on a server, you'll first need to plan what Local Resource Managers (LRMs) you want GRAM5 to interface with, what LRM you want to have as your default GRAM5 service, and whether you'll be using the globus-scheduler-event-generator to process LRM events.

GRAM5 requires a few services to be running to function: the Gatekeeper and the Scheduler Event Generator (SEG). The supported way to run these services is via the System-V style init scripts provided with the GRAM5-related packages. The gatekeeper daemon can also be configured to start via an internet superserver such as inetd or xinetd though that is beyond the scope of this document. The globus-scheduler-event-generator can not be run in that way.

2.1. Choosing an LRM Adapter

GRAM5 in GT 5.2.2 supports the following LRM adapters: Condor, PBS, GridEngine, and Fork. These LRM adapters translate GRAM5 job specifications into LRM-specific job descriptions and scripts to run them, as well as interfaces to the LRM to determine job termination status.

If you're not familiar with the supported LRMs, you might want to start with the Fork one to get familiar with how GRAM5 works. This adapter simply forks the job and runs it on the GRAM5 node. You can then install one of the other LRMs and its adapter to provide batch or high-throughput job scheduling.

2.1.1. Default GRAM5 Service

GRAM5 can be configured to support multiple LRMs on the same service machine. In that case, one LRM is typically configured as the default LRM which is used when a client uses a shortened version of a GRAM5 resource name. A common configuration is to configure a batch system interface as the default, and provide the jobmanager-fork service as well for simple jobs, such as creating directories or staging data.

2.1.2. Job Status Method

GRAM5 has two ways of determining job state transitions: polling the LRM and using the Scheduler Event Generator (SEG) service. When polling, each user's globus-job-manager will periodically execute an LRM-specific command to determine the state of each job. On systems with many users, or with users submitting a large number of jobs, this can cause significant resource use on the GRAM5 service machine. Instead, the GRAM5 service can be configured (on a per-LRM basis) to use the globus-scheduler-event-generator service to more efficiently process LRM state changes.
Note

Not all LRM adapters provide an interface to the `globus-scheduler-event-generator`, and some require LRM-specific configuration to work properly. This is described in more detail.

3. Installing LRM Adapter Packages

There are several LRM adapters included in the GT 5.2.2. For some, there is a `-setup-poll` and `-setup-seg` package which installs the adapter and configuration file needed for job status via polling or the `globus-scheduler-event-generator` program.

There are three ways to get LRM adapters: as RPM packages, as Debian packages, and from the source installer. These installation methods are described in Installing GT 5.2.2.

LRM adapter packages included in the GT 5.2.2 release are:

Table 1.1. GRAM5 LRM Adapters

<table>
<thead>
<tr>
<th>LRM Adapter</th>
<th>Poll Package</th>
<th>SEG Package</th>
<th>Installer Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>fork</td>
<td><code>globus-gram-job-manager-fork-setup-poll</code></td>
<td><code>globus-gram-job-manager-fork-setup-seg</code></td>
<td><code>globus_gram_job_manager_fork</code></td>
</tr>
<tr>
<td>pbs¹</td>
<td><code>globus-gram-job-manager-pbs-setup-poll</code></td>
<td><code>globus-gram-job-manager-pbs-setup-seg</code></td>
<td><code>globus_gram_job_manager_pbs</code></td>
</tr>
<tr>
<td>Condor²</td>
<td>N/A</td>
<td><code>globus-gram-job-manager-condor</code></td>
<td><code>globus_gram_job_manager_condor</code></td>
</tr>
<tr>
<td>SGE</td>
<td><code>globus-gram-job-manager-sge-setup-poll</code></td>
<td><code>globus-gram-job-manager-sge-setup-seg</code></td>
<td><code>globus_gram_job_manager_sge</code></td>
</tr>
</tbody>
</table>

¹Not recommended for production use
²This module does not work with torque 3.0.1-5 in Fedora 15 because of a bug causing `qstat` to hang. This bug is mentioned on the TORQUE user list [http://www.supercluster.org/pipermail/torqueusers/2011-May/012911.html] and is fixed in newer versions.
³This LRM uses a SEG-like mechanism included in the `globus-job-manager` program, but not the `globus-scheduler-event-generator` service.

² http://www.cs.wisc.edu/condor/
Chapter 2. Common Administrative Tasks

There are several tools provided with GT 5.2.2 to manage GRAM5, as well as OS-specific tools to start and stop some of the services. There are tools to manage user authorization, which services are enabled, which scheduler event generator modules are enabled, and to test the globus-gatekeeper service.

1. Managing GRAM5 Users

Before a user may interact with the GRAM5 service to submit jobs, he or she must be authorized to use the service. In order to be authorized, a GRAM5 administrator must add the user's credential name and local account mapping to the /etc/grid-mapfile. This can be done using the grid-mapfile-add-entry and grid-mapfile-delete-entry tools. For more information, see the GSI C manual.

2. Starting and Stopping GRAM5 services

In order to run the service, the globus-gatekeeper, and, if applicable to your configuration, the globus-scheduler-event-generator services must be running on your system. The packages for these services include init scripts and configuration files which can be used to configure, start, and stop the service.

The globus-gatekeeper and globus-scheduler-event-generator init scripts handle the following actions: start, stop, status, restart, condrestart, try-restart, reload, and force-reload. The globus-scheduler-event-generator script also accepts another optional parameter to start or stop a particular globus-scheduler-event-generator module. If the second parameter is not present, then all services will be acted on.

2.1. Debian Specifics

If you installed using Debian packaging tools, then the services will automatically be started upon installation. To start or stop the service, use the command invoke-rc.d with the service name and action.

2.2. RPM Specifics

If you installed using the RPM packaging tools, then the services will be installed but not enabled by default. To enable the services to start at boot time, use the commands:

```
# chkconfig globus-gatekeeper on
# chkconfig globus-scheduler-event-generator on
```

To start or stop the services, use the service command to run the init scripts with the service name and action and optional globus-scheduler-event-generator module.

3. Enabling and Disabling GRAM5 Services

The GRAM5 packages described in Section 3, “Installing LRM Adapter Packages” will automatically register themselves with the globus-gatekeeper and globus-scheduler-event-generator services. The first LRM adapter installed will be configured as the default Job Manager service. To list the installed services, change the default, or disable a service, use the (8) tool.
Common Administrative Tasks

Example 2.1. Using globus-gatekeeper-admin to set the default service

This example shows how to use the `globus-gatekeeper-admin` tool to list the available services and then choose one as the default:

```
# globus-gatekeeper-admin -l
jobmanager-condor [ENABLED]
jobmanager-fork-poll [ENABLED]
jobmanager-fork [ALIAS to jobmanager-fork-poll]
# globus-gatekeeper-admin -e jobmanager-condor -n jobmanager
# globus-gatekeeper-admin -l
jobmanager-condor [ENABLED]
jobmanager-fork-poll [ENABLED]
jobmanager [ALIAS to jobmanager-condor]
jobmanager-fork [ALIAS to jobmanager-fork-poll]
```

4. Enabling and Disabling SEG Modules

The `-setup-seg` packages described in Section 3, “Installing LRM Adapter Packages” will automatically register themselves with the `globus-scheduler-event-generator` service. To disable a module from running when the `globus-scheduler-event-generator` service is started, use the `globus-scheduler-event-generator-admin` tool.

Example 2.2. Using globus-scheduler-event-generator-admin to disable a SEG module

This example shows how to stop the `pbs` `globus-scheduler-event-generator` module and disable it so it will not restart when the system is rebooted:

```
# /etc/init.d/globus-scheduler-event-generator stop pbs
Stopped globus-scheduler-event-generator                  [  OK  ]
# globus-scheduler-event-generator-admin -d pbs
# globus-scheduler-event-generator-admin -l
pbs [DISABLED]
```
Chapter 3. Configuring GRAM5

GRAM5 is designed to be usable by default without any manual configuration. However, there are many ways to customize a GRAM5 installation to better interact with site policies, filesystem layouts, LRM interactions, logging, and auditing. In addition to GRAM5-specific configuration, see Configuring GSI for information about configuring GSI security.

1. Gatekeeper Configuration

The `globus-gatekeeper` has many configuration options related to network configuration, security, logging, service path, and nice level. This configuration is located in:

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>Configuration Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>/etc/sysconfig/globus-gatekeeper</td>
</tr>
<tr>
<td>Debian Package</td>
<td>/etc/default/globus-gatekeeper</td>
</tr>
<tr>
<td>Source Installer</td>
<td>PREFIX/etc/globus-gatekeeper.conf</td>
</tr>
</tbody>
</table>

The following configuration variables are available in the `globus-gatekeeper` configuration file:

- **GLOBUS_GATEKEEPER_PORT**: Gatekeeper Service Port. If not set, the `globus-gatekeeper` uses the default of 2119.
- **GLOBUS_LOCATION**: Globus Installation Path. If not set, the `globus-gatekeeper` uses the paths defined at package compilation time.
- **GLOBUS_GATEKEEPER_LOG**: Gatekeeper Log Filename. If not set, the `globus-gatekeeper` logs to syslog using the `GRAM-gatekeeper` log identification prefix. The default configuration value is `var/log/globus-gatekeeper.log`.
- **GLOBUS_GATEKEEPER_GRID_SERVICES**: Path to grid service definitions. If not set, the `globus-gatekeeper` uses the default of `/etc/grid-services`.
- **GLOBUS_GATEKEEPER_GRIDMAP**: Path to grid-mapfile for authorization. If not set, the `globus-gatekeeper` uses the default of `/etc/grid-security/grid-mapfile`.
- **GLOBUS_GATEKEEPER_CERT_DIR**: Path to a trusted certificate root directory. If not set, the `globus-gatekeeper` uses the default of `/etc/grid-security/certificates`.
- **GLOBUS_GATEKEEPER_CERT_FILE**: Path to the gatekeeper's certificate. If not set, the `globus-gatekeeper` uses the default of `/etc/grid-security/hostcert.pem`.
- **GLOBUS_GATEKEEPER_KEY_FILE**: Path to the gatekeeper's private key. If not set, the `globus-gatekeeper` uses the default of `/etc/grid-security/hostkey.pem`. 
Configuring GRAM5

GLOBUS_GATEKEEPER_KERBEROS_ENABLED Flag indicating whether or not the globus-gatekeeper will use a kerberos GSSAPI implementation instead of the GSI GSSAPI implementation (untested).

GLOBUS_GATEKEEPER_KMAP Path to the KMAP authentication module. (untested).

GLOBUS_GATEKEEPER_PIDFILE Path to a file where the globus-gatekeeper's process ID is written. If not set, globus-gatekeeper uses /var/run/globus-gatekeeper.pid

GLOBUS_GATEKEEPER_NICE_LEVEL Process nice level for globus-gatekeeper and globus-job-manager processes. If not set, the default system process nice level is used.

After modifying the configuration file, restart the globus-gatekeeper using the methods described in Section 2, “Starting and Stopping GRAM5 services”.

2. Scheduler Event Generator Configuration

The globus-scheduler-event-generator has several configuration options related to filesystem paths. This configuration is located in:

Table 3.2. Scheduler Event Generator Configuration Path

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>Configuration Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>/etc/sysconfig/globus-scheduler-event-generator</td>
</tr>
<tr>
<td>Debian Package</td>
<td>/etc/default/globus-scheduler-event-generator</td>
</tr>
<tr>
<td>Source Installer</td>
<td>PREFIX/etc/globus-scheduler-event-generator.conf</td>
</tr>
</tbody>
</table>

The following configuration variables are available in the globus-scheduler-event-generator configuration file:

GLOBUS_SEG_PIDFMT Scheduler Event Generator PID file path format. Modify this to be the location where the globus-scheduler-event-generator writes its process IDs (one per configured LRM). The format is a printf format string with one %s to be replaced by the LRM name. By default, globus-scheduler-event-generator uses /var/run/globus-scheduler-event-generator-%s.pid.

GLOBUS_SEG_LOGFMT Scheduler Event Generator Log path format. Modify this to be the location where globus-scheduler-event-generator writes its event logs. The format is a printf format string with one %s to be replaced by the LRM name. By default, globus-scheduler-event-generator uses /var/lib/globus/globus-seg-%s. If you modify this value, you'll need to also update the LRM configuration file to look for the log file in the new location.

GLOBUS_SEG_NICE_LEVEL Process nice level for globus-scheduler-event-generator processes. If not set, the default system process nice level is used.

After modifying the configuration file, restart the globus-scheduler-event-generator using the methods described in Section 2, “Starting and Stopping GRAM5 services”.

6
3. Job Manager Configuration

The `globus-job-manager` process is started by the `globus-gatekeeper` and uses the configuration defined in the service entry for the resource name. By default, these service entries use a common configuration file for most job manager features. This configuration is located in:

**Table 3.3. Job Manager Configuration Path**

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>Configuration Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td><code>/etc/globus/globus-gram-job-manager.conf</code></td>
</tr>
<tr>
<td>Debian Package</td>
<td><code>/etc/globus/globus-gram-job-manager.conf</code></td>
</tr>
<tr>
<td>Source Installer</td>
<td><code>PREFIX/etc/globus/globus-gram-job-manager.conf</code></td>
</tr>
</tbody>
</table>

This configuration file is used to construct the command-line options for the `globus-job-manager` program. Thus, all of the options described in (8) may be used.

### 3.1. Job Manager Logging

From an administrator's perspective, the most important job manager configuration options are likely the ones related to logging and auditing. The default GRAM5 configuration puts logs in `/var/log/globus/gram_USERNAME.log`, with logging enabled at the `FATAL` and `ERROR` levels. To enable more fine-grained logging, add the option `-log-levels LEVELS` to `/etc/globus/globus-gram-job-manager.conf`. The value for `LEVELS` is a set of log levels joined by the `|` character. The available log levels are:

**Table 3.4. GRAM5 Log Levels**

<table>
<thead>
<tr>
<th>Level</th>
<th>Meaning</th>
<th>Default Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>FATAL</td>
<td>Problems which cause the job manager to terminate prematurely.</td>
<td>Enabled</td>
</tr>
<tr>
<td>ERROR</td>
<td>Problems which cause a job or operation to fail.</td>
<td>Enabled</td>
</tr>
<tr>
<td>WARN</td>
<td>Problems which cause minor problems with job execution or monitoring.</td>
<td>Disabled</td>
</tr>
<tr>
<td>INFO</td>
<td>Major events in the lifetime of the job manager and its jobs.</td>
<td>Disabled</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Minor events in the lifetime of jobs.</td>
<td>Disabled</td>
</tr>
<tr>
<td>TRACE</td>
<td>Job processing details.</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

In RPM or Debian package installs, these logs will be configured to be rotated via `logrotate`. See `/etc/logrotate.d/globus-job-manager` for details on the default log rotation configuration.

### 3.2. Firewall Configuration

There are also a few configuration options related to the TCP ports the Job Manager users. This port configuration is useful when dealing with firewalls that restrict incoming or outgoing ports. To restrict incoming ports (those
Configuring GRAM5

that the Job Manager listens on), add the command-line option \texttt{-globus-tcp-port-range} to the Job Manager configuration file like this:

\texttt{-globus-tcp-port-range MIN-PORT,MAX-PORT}

Where \textit{MIN-PORT} is the minimum TCP port number the Job Manager will listen on and \textit{MAX-PORT} is the maximum TCP port number the Job Manager will listen on.

Similarly, to restrict the outgoing port numbers that the job manager connects form, use the command-line option \texttt{-globus-tcp-source-range}, like this:

\texttt{-globus-tcp-source-range MIN-PORT,MAX-PORT}

Where \textit{MIN-PORT} is the minimum outgoing TCP port number the Job Manager will use and \textit{MAX-PORT} is the maximum TCP outgoing port number the Job Manager will use.

For more information about Globus and firewalls, see Section 4, “Firewall configuration”.

4. LRM Adapter Configuration

Each LRM adapter has its own configuration file which can help customize the adapter to the site configuration. Some LRMs use non-standard programs to launch parallel or MPI jobs, and some might want to provide queue or project validation to make it easier to translate job failures into problems that can be described by GRAM5. All of the LRM adapter configuration files consist of simple \texttt{variable="value"} pairs, with a leading \# starting a comment until end-of-line.

Generally, the GRAM5 LRM configuration files are located in the globus configuration directory, with each configuration file named by the LRM name (fork, condor, pbs, sge). The following table contains the paths to these configurations:

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>Configuration Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>/etc/globus/globus-LRM.conf</td>
</tr>
<tr>
<td>Debian Package</td>
<td>/etc/globus/globus-LRM.conf</td>
</tr>
<tr>
<td>Source Installer</td>
<td>PREFIX/etc/globus/globus-LRM.conf</td>
</tr>
</tbody>
</table>

4.1. Fork

The \texttt{globus-fork.conf} configuration file can define the following configuration parameters:

- \texttt{log_path} Path to the \texttt{globus-fork.log} file used by the \texttt{globus-fork-starter} and fork SEG module.
- \texttt{mpiexec, mpirun} Path to \texttt{mpiexec} and \texttt{mpirun} for parallel jobs which use MPI. By default, these are not configured. The LRM adapter will use \texttt{mpiexec} over \texttt{mpirun} if both are defined.
- \texttt{softenv_dir} Path to an installation of softenv\textsuperscript{1}, which is used on some systems to manage application environment variables.

4.2. Condor

The \texttt{globus-condor.conf} configuration file can define the following configuration parameters:

\footnote{1 \url{http://www.mcs.anl.gov/hs/software/systems/softenv/softenv-intro.html}}
Configuring GRAM5

condor_os
Custom value for the `OpSys` requirement for condor jobs. If not specified, the system-wide default will be used.

condor_arch
Custom value for the `OpSys` requirement for condor jobs. If not specified, the system-wide default will be used.

condor_submit, condor_rm
Path to the condor commands that the LRM adapter uses. These are usually determined when the LRM adapter is compiled if the commands are in the `PATH`.

condor_config
Value of the `CONDOR_CONFIG` environment variable, which might be needed to use condor in some cases.

check_vanilla_files
Enable checking if executable, standard input, and directory are valid paths for vanilla universe jobs. This can detect some types of errors before submitting jobs to condor, but only if the filesystems between the condor submit host and condor execution hosts are equivalent. In other cases, this may cause unnecessary job failures.

condor_mpi_script
Path to a script to launch MPI jobs on condor

4.3. PBS

The `globus-pbs.conf` configuration file can define the following configuration parameters:

log_path
Path to PBS server_logs directory. The PBS SEG module parses these logs to generate LRM events.

pbs_default
Name of the PBS server node, if not the same as the GRAM service node.

mpiexec, mpirun
Path to `mpiexec` and `mpirun` for parallel jobs which use MPI. By default these are not configured. The LRM adapter will use `mpiexec` over `mpirun` if both are defined.

qsub, qstat, qdel
Path to the LRM-specific command to submit, check, and delete PBS jobs. These are usually determined when the LRM adapter is compiled if they are in the `PATH`.

cluster
If this value is set to `yes`, then the LRM adapter will attempt to use a remote shell command to launch multiple instances of the executable on different nodes, as defined by the file named by the `PBS_NODEFILE` environment variable.

remote_shell
Remote shell command to launch processes on different nodes when `cluster` is set to `yes`.

cpu_per_node
Number of instances of the executable to launch per allocated node.

softenv_dir
Path to an installation of softenv\(^2\) which is used on some systems to manage application environment variables.

4.4. SGE

The `globus-sge.conf` configuration file can define the following configuration parameters:

sge_root
Root location of the GridEngine installation. If this is set to `undefined`, then the LRM adapter will try to determine it from the `globus-job-manager` environment, or if not there, the contents of the file named by the `sge_config` configuration parameter.

\(^2\)http://www.mcs.anl.gov/hs/software/systems/softenv/softenv-intro.html
sge_cell  Name of the GridEngine cell to interact with. If this is set to undefined, then the LRM adapter will try to determine it from the globus-job-manager environment, or if not there, the contents of the file named by the sge_config configuration parameter.

gse_config  Path to a file which defines the SGE_ROOT and the SGE_CELL environment variables.

log_path  Path to GridEngine reporting file. This value is used by the SGE SEG module. If this is used, GridEngine must be configured to write a reporting file and not load reporting data into an ARCo database.

qsub, qstat, qdel, qconf  Path to the LRM-specific command to submit, check, and delete GridEngine jobs. These are usually determined when the LRM adapter is compiled if they are in the PATH.

sun_mprun, mpirun  Path to mprun and mpirun for parallel jobs which use MPI. By default these are not configured. The LRM adapter will use mprun over mpirun if both are defined.

default_pe  Default parallel environment to submit parallel jobs to. If this is not set, then clients must use the parallel_environment RSL attribute to choose one.

validate_pes  If this value is set to yes, then the LRM adapter will verify that the parallel_environment RSL attribute value matches one of the parallel environments supported by this GridEngine service.

available_pes  If this value is defined, use it as a list of parallel environments supported by this GridEngine deployment for validation when validate_pes is set to yes. If validation is being done but this value is not set, then the LRM adapter will query the GridEngine service to determine available parallel environments at startup.

default_queue  Default queue to use if the job description does not name one.

validate_queues  If this value is set to yes, then the LRM adapter will verify that the queue RSL attribute value matches one of the queues supported by this GridEngine service.

available_queues  If this value is defined, use it as a list of queues supported by this GridEngine deployment for validation when validate_queues is set to yes. If validation is being done but this value is not set, then the LRM adapter will query the GridEngine service to determine available queues at startup.

4.4.1. Enabling reporting for the GridEngine Scheduler Event Generator

In order to use the Scheduler Event Generator with GridEngine, the job reporting feature must be enabled, and ARCo database storage must not be enabled. To enable this, use the command qconf -mconf and modify the reporting_params parameter so that the options reporting and joblog are set to true.

5. Auditing

The globus-gram-audit configuration defines information about the database to load the GRAM5 audit records into. This configuration is located in:

Table 3.6. GRAM Audit Configuration Path

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>Configuration Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>/etc/globus/gram-audit.conf</td>
</tr>
</tbody>
</table>
This configuration file contains the following attributes. Each attribute is defined by a ATTRIBUTE:VALUE pair.

**Table 3.7. Audit Configuration Attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIVER</td>
<td>The name of the Perl 5 DBI driver for the database to be used. The supported drivers for this program are SQLite, Pg (for PostgreSQL), and mysql.</td>
<td>SQLite</td>
</tr>
<tr>
<td>DATABASE</td>
<td>The DBI data source specification to contact the audit database.</td>
<td>dbname=/var/gram_audit_database/gram_audit.db</td>
</tr>
<tr>
<td>USERNAME</td>
<td>Username to authenticate as to the database</td>
<td></td>
</tr>
<tr>
<td>PASSWORD</td>
<td>Password to use to authenticate with the database</td>
<td></td>
</tr>
<tr>
<td>AUDITVERSION</td>
<td>Version of the audit database table schemas to use. May be 1 or 1TG for this version of the software.</td>
<td>1</td>
</tr>
</tbody>
</table>

**6. RSL Attributes**

GRAM5 uses the RSL language to encode job descriptions. The attributes supported by gram are defined in RSL Validation Files. These definitions contain information about when the different RSL attributes are valid and what their default values might be if not present. GRAM5 will look in /etc/globus/gram/job-manager.rvf and /etc/globus/gram/LRM.rvf for site-specific changes to the RSL validation file.
Chapter 4. Audit Logging

1. Overview

GRAM5 includes mechanisms to provide access to audit and accounting information associated with jobs that GRAM5 submits to a local resource manager (LRM) such as Torque, GridEngine, or Condor.

In some scenarios, it is desirable to get general information about the usage of the underlying LRM, such as:

- What kinds of jobs were submitted via GRAM?
- How long did the processing of a job take?
- How many jobs were submitted by user X?

The following three use cases give a better overview of the meaning and purpose of auditing and accounting:

1. **Group Access**: A grid resource provider allows a remote service (e.g., a gateway or portal) to submit jobs on behalf of multiple users. The grid resource provider only obtains information about the identity of the remote submitting service and thus does not know the identity of the users for which the grid jobs are submitted. This group access is allowed under the condition that the remote service stores audit information so that, if and when needed, the grid resource provider can request and obtain information to track a specific job back to an individual user.

2. **Query Job Accounting**: A client that submits a job needs to be able to obtain, after the job has completed, information about the resources consumed by that job. In portal and gateway environments where many users submit many jobs against a single allocation, this per-job accounting information is needed soon after the job completes so that client-side accounting can be updated. Accounting information is sensitive and thus should only be released to authorized parties.

3. **Auditing**: In a distributed, multi-site environment, it can be necessary to investigate various forms of suspected intrusion and abuse. In such cases, we may need to access an audit trail of the actions performed by a service. When accessing this audit trail, it will frequently be important to be able to relate specific actions to the user.

Audit logging in GRAM5 is done when a job completes.

2. Audit and Accounting Records

While audit and accounting records may be generated and stored by different entities in different contexts, we make the following assumptions in this chapter:

<table>
<thead>
<tr>
<th></th>
<th>Audit Records</th>
<th>Accounting Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generated by:</td>
<td>GRAM service</td>
<td>LRM to which the GRAM service submits jobs</td>
</tr>
<tr>
<td>Stored in:</td>
<td>Database, indexed by GJID</td>
<td>LRM, indexed by JID</td>
</tr>
<tr>
<td>Data that is stored:</td>
<td>See list below.</td>
<td>May include all information about the duration and resource-usage of a job</td>
</tr>
</tbody>
</table>

The audit record of each job contains the following data:

- **job_grid_id**: String representation of the resource EPR
Audit Logging

- **local_job_id**: Job/process id generated by the scheduler
- **subject_name**: Distinguished name (DN) of the user
- **username**: Local username
- **idempotence_id**: Job id generated on the client-side
- **creation_time**: Date when the job resource is created
- **queued_time**: Date when the job is submitted to the scheduler
- **stage_in_grid_id**: String representation of the stageIn-EPR (RFT)
- **stage_out_grid_id**: String representation of the stageOut-EPR (RFT)
- **clean_up_grid_id**: String representation of the cleanUp-EPR (RFT)
- **globus_toolkit_version**: Version of the server-side GT
- **resource_manager_type**: Type of the resource manager (Fork, Condor, ...)
- **job_description**: Complete job description document
- **success_flag**: Flag that shows whether the job failed or finished successfully
- **finished_flag**: Flag that shows whether the job is already fully processed or still in progress
- **gateway_user**: Teragrid identity of the user which submitted the job.

### 3. For More Information

The rest of this chapter focuses on how to configure GRAM5 to enable Audit-Logging.

### 4. Configuration

Audit logging is turned off by default. To enable GRAM5 audit logging, in the job manager, add the command-line option `-audit-directory AUDIT-DIRECTORY` to the job manager configuration in one of the following locations:

- `$GLOBUS_LOCATION /etc/globus-job-manager.conf` to enable it for all job manager services
- `$GLOBUS_LOCATION /etc/grid-services/LRM_SERVICE_NAME` to enable it for a particular job manager service for a particular LRM.

### 5. Audit Database Interface

The `globus-gram-audit` program reads GRAM5 audit records and loads those records into a SQL database. This program is available as part of the `globus-gram_job_manager_auditing` package. It must be configured by installing and running the `globus_job_manager_auditing_setup_scripts` setup package via `gpt-postinstall`. This setup script creates the `$GLOBUS_LOCATION/etc/globus-job-manager-audit.conf` configuration file described below and creates database tables needed by the audit system.

The `globus-gram-audit` program support three database systems: MySQL, PostgreSQL, and SQLite.
Chapter 5. Security Considerations

1. Security Considerations

1.1. Gatekeeper Security Considerations

GRAM5 runs different parts of itself under different privilege levels. The `globus-gatekeeper` runs as root, and uses its root privilege to access the host's private key. It uses the `grid map file` to map `Grid Certificates` to local user ids and then uses the `setuid()` function to change to that user and execute the `globus-job-manager` program.

1.2. Job Manager Security Considerations

The `globus-job-manager` program runs as a local non-root account. It receives a delegated limited `proxy certificate` from the GRAM5 client which it uses to access Grid storage resources via GridFTP and to authenticate job signals (such as client cancel requests), and send job state callbacks to registered clients. This proxy is generally short-lived, and is automatically removed by the job manager when the job completes.

The `globus-job-manager` program uses a publicly-writable directory for job state files. This directory has the `sticky` bit set, so users may not remove other users files. Each file is named by a UUID, so it should be unique.

1.3. Fork SEG Module Security Considerations

The Fork Scheduler Event Generator module uses a globally writable file for job state change events. This is not recommended for production use.
Chapter 6. Troubleshooting

1. Admin Troubleshooting

1.1. Security

GRAM requires a host certificate and private key in order for the `globus-gatekeeper` service to run. These are typically located in `/etc/grid-security/hostcert.pem` and `/etc/grid-security/hostkey.pem`, but the path is configurable in the `gatekeeper` configuration file. The key must be protected by file permissions allowing only the root user to read it.

GRAM also (by default) uses a `grid-mapfile` to authorize Grid users as local users. This file is typically located in `/etc/grid-security/grid-mapfile`, but is configurable in the `gatekeeper` configuration file.

Problems in either of these configurations will show up in the gatekeeper log described below. See the GSI documentation for more detailed information about obtaining and installing host certificates and maintaining a `grid-mapfile`.

1.2. Verify that Services are Running

GRAM relies on the `globus-gatekeeper` program and (in some cases) the `globus-scheduler-event-generator` programs to process jobs. If the former is not running, jobs requests will fail with a "connection refused" error. If the latter is not running, GRAM jobs will appear to "hang" in the PENDING state.

The `globus-gatekeeper` is typically started via an init script installed in `/etc/init.d/globus-gatekeeper`. The command `/etc/init.d/globus-gatekeeper status` will indicate whether the service is running. See Section 2, “Starting and Stopping GRAM5 services” for more information about starting and stopping the `globus-gatekeeper` program.

If the `globus-gatekeeper` service fails to start, the output of the command `globus-gatekeeper -test` will output information describing some types of configuration problems.

The `globus-scheduler-event-generator` is typically started via an init script installed in `/etc/init.d/globus-scheduler-event-generator`. It is only needed when the LRM-specific "setup-seg" package is installed. The command `/etc/init.d/globus-scheduler-event-generator status` will indicate whether the service is running. See Section 2, “Starting and Stopping GRAM5 services” for more information about starting and stopping the `globus-scheduler-event-generator` program.

1.3. Verify that LRM packages are installed

The `globus-gatekeeper` program starts the `globus-job-manager` service with different command-line parameters depending on the LRM being used. Use the command `globus-gatekeeper-admin -l` to list which LRMs the gatekeeper is configured to use.

The `globus-job-manager-script.pl` is the interface between the GRAM job manager process and the LRM adapter. The command `/usr/share/globus/globus-job-manager-script.pl -h` will print the list of available adapters.

```
% /usr/share/globus/globus-job-manager-script.pl -h
USAGE: /usr/share/globus/globus-job-manager-script.pl -m MANAGER -f FILE -c COMMAND
Installed managers: condor fork
```

The `globus-scheduler-event-generator` also uses an LRM-specific module to generate scheduler events for GRAM to reduce the amount of resources GRAM uses on the machine where it runs. To determine which LRMs are installed and configured, use the command `globus-scheduler-event-generator-admin -l`. 
% globus-scheduler-event-generator-admin -l
fork [DISABLED]

If any of these do not show the LRM you are trying to use, install the relevant packages related to that LRM and restart the GRAM services. See the GRAM Administrator's Guide for more information about starting and stopping the GRAM services.

1.4. Verify that the LRM packages are configured

All GRAM5 LRM adapters have a configuration file for site customizations, such as queue names, paths to executables needed to interface with the LRM, etc. Check that the values in these files are correct. These files are described in Section 4, “LRM Adapter Configuration”.

1.5. Check the Gatekeeper Log

The /var/log/globus-gatekeeper.log file contains information about service requests from clients, and will be useful when diagnosing service startup failures, authentication failures, and authorization failures.

1.5.1. Authorization failures

GRAM uses GSI to authenticate client job requests. If there is a problem with the GSI configuration for your host, or a client is trying to connect with a certificate signed by a CA your host does not trust, the job request will fail. This will show up in the log as a “GSS authentication failure”. See the GSI Administrator's Guide for information about diagnosing authentication failures.

1.5.2. Gridmap failures

After authentication is complete, GRAM maps the Grid identity to a local user prior to starting the globus-job-manager process. If this fails, an error will show up in the log as “globus_gss_assist_gridmap() failed authorization”. See the GSI Administrator's Guide for information about managing gridmap files.

1.6. Job Manager Logs

A per-user job manager log is typically located in /var/log/globus/gram_\$USERNAME.log. This log contains information from the job manager as it attempts to execute GRAM jobs via a local resource manager. The logs can be fairly verbose. Sometimes looking for log entries near those containing the string level=ERROR will show more information about what caused a particular failure.

Once you’ve found an error in the log, it is generally useful to find log entries related to the job which hit that error. There are two job IDs associated with each job, one a GRAM-specific ID, and one an LRM-specific ID. To determine the GRAM ID associated with a job, look for the attribute gramid in the log message. Finding that, looking for all other log messages which contain that gramid value will give a better picture of what the job manager is doing. To determine the LRM-specific ID, look for a message at TRACE level with the matching GRAM ID found above with the response value matching GRAM_SCRIPT_JOB_ID:LRM-ID. You can then find follow the state of the LRM-ID as well as the GRAM ID in the log, and correlate the LRM-ID information with local resource manager logs and administrative tools.

1.7. Email Support

If all else fails, please send information about your problem to <gram-user@globus.org>. You’ll have to subscribe to a list before you can send an e-mail to it. See here for general e-mail lists and information on how to subscribe.

---

1. http://dev.globus.org/wiki/Mailing_Lists
scribe to a list and here\textsuperscript{2} for GRAM-specific lists. Depending on the problem, you may be requested to file a bug report to the Globus project's Issue Tracker\textsuperscript{3}.

\textsuperscript{2} http://dev.globus.org/wiki/GRAM#Mailing_Lists
\textsuperscript{3} http://jira.globus.org
Chapter 7. Admin Tools
Name
globus-gatekeeper — Authorize and execute a grid service on behalf of a user

Synopsis

globus-gatekeeper [-help]
[-conf PARAMETER_FILE]
[-test] [-d] [-debug]
[-inetd] [-f]
[-p PORT] [-port PORT]
[-home PATH] [-logfile LOGFILE] [-logf LOG_FACILITY]
[-acctfile ACCTFILE]
[-e LIBEXECDIR]
[-launch_method { fork_and_exit | fork_and_wait | dont_fork }]
[-grid_services SERVICEDIR]
[-globusid GLOBUSID]
[-gridmap GRIDMAP]
[-x509_cert_dir TRUSTED_CERT_DIR]
[-x509_cert_file TRUSTED_CERT_FILE]
[-x509_user_cert CERT_PATH]
[-x509_user_key KEY_PATH]
[-x509_user_proxy PROXY_PATH]
[-k]
[-globuskmap KMAP]
[-pidfile PIDFILE]

Description

The globus-gatekeeper program is a meta-server similar to inetd or xinetd that starts other services after authenticating a TCP connection using GSSAPI and mapping the client’s credential to a local account.

The most common use for the globus-gatekeeper program is to start instances of the globus-job-manager(8) service. A single globus-gatekeeper deployment can handle multiple different service configurations by having entries in the /etc/grid-services directory.

Typically, users interact with the globus-gatekeeper program via client applications such as globusrun(1), globus-job-submit, or tools such as CoG jglobus or Condor-G.

The full set of command-line options to globus-gatekeeper consists of:

- **-help** Display a help message to standard error and exit

- **-conf** PARAMETER_FILE Load configuration parameters from PARAMETER_FILE. The parameters in that file are treated as additional command-line options.

- **-test** Parse the configuration file and print out the POSIX user id of the globus-gatekeeper process, service home directory, service execution directory, and X.509 subject name and then exits.

- **-d**, **-debug** Run the globus-gatekeeper process in the foreground.

- **-inetd** Flag to indicate that the globus-gatekeeper process was started via inetd or a similar super-server. If this flag is set and the globus-gatekeeper was not started via inetd, a warning will be printed in the gatekeeper log.
Admin Tools

-f Flag to indicate that the **globus-gatekeeper** process should run in the foreground. This flag has no effect when the **globus-gatekeeper** is started via inetd.

-p PORT, -port PORT Listen for connections on the TCP/IP port PORT. This option has no effect if the **globus-gatekeeper** is started via inetd or a similar service. If not specified and the gatekeeper is running as root, the default of 2119 is used. Otherwise, the gatekeeper defaults to an ephemeral port.

-home PATH Sets the gatekeeper deployment directory to PATH. This is used to interpret relative paths for accounting files, libexecdir, certificate paths, and also to set the GLOBUS_LOCATION environment variable in the service environment. If not specified, the gatekeeper looks for service executables in /usr/sbin, configuration in /etc, and writes logs and accounting files to /var/log.

-l LOGFILE, -logfile LOGFILE Write log entries to LOGFILE. If LOGFILE is equal to logoff or LOGOFF, then logging will be disabled, both to file and to syslog.

-if LOG_FACILITY Open syslog using the LOG_FACILITY. If not specified, LOG_DAEMON will be used as the default when using syslog.

-acctfile ACCTFILE Set the path to write accounting records to ACCTFILE. If not set, records will be written to the log file.

-e LIBEXECDIR Look for service executables in LIBEXECDIR. If not specified, the sbin subdirectory of the parameter to -home is used, or /usr/sbin if that is not set.

-launch_method Determine how to launch services. The method may be either fork_and_exit (the service runs completely independently of the gatekeeper, which exits after creating the new service process), fork_and_wait (the service is run in a separate process from the gatekeeper but the gatekeeper does not exit until the service terminates), or dont_fork, where the gatekeeper process becomes the service process via the exec() system call.

-grid_services SERVICEDIR Look for service descriptions in SERVICEDIR.

-globusid GLOBUSID Sets the GLOBUSID environment variable to GLOBUSID. This variable is used to construct the gatekeeper contact string if it can not be parsed from the service credential.

-gridmap GRIDMAP Use the file at GRIDMAP to map GSSAPI names to POSIX user names.

-x509_cert_dir TRUSTED_CERT_DIR Use the directory TRUSTED_CERT_DIR to locate trusted CA X.509 certificates. The gatekeeper sets the environment variable X509_CERT_DIR to this value.

-x509_user_cert CERT_PATH Read the service X.509 certificate from CERT_PATH. The gatekeeper sets the X509_USER_CERT environment variable to this value.

-x509_user_key KEY_PATH Read the private key for the service from KEY_PATH. The gatekeeper sets the X509_USER_KEY environment variable to this value.

-x509_user_proxy PROXY_PATH Read the X.509 proxy certificate from PROXY_PATH. The gatekeeper sets the X509_USER_PROXY environment variable to this value.
-k Use the \texttt{globus-k5} command to acquire Kerberos 5 credentials before starting the service.

-\texttt{globuskmap KMAP} Use \texttt{KMAP} as the path to the Grid credential to kerberos initialization mapping file.

-\texttt{pidfile PIDFILE} Write the process id of the \texttt{globus-gatekeeper} to the file named by \texttt{PIDFILE}.

\section*{ENVIRONMENT}

If the following variables affect the execution of \texttt{globus-gatekeeper}:

\begin{itemize}
\item \texttt{X509_CERT_DIR} Directory containing X.509 trust anchors and signing policy files.
\item \texttt{X509_USER_PROXY} Path to file containing an X.509 proxy.
\item \texttt{X509_USER_CERT} Path to file containing an X.509 user certificate.
\item \texttt{X509_USER_KEY} Path to file containing an X.509 user key.
\item \texttt{GLOBUS_LOCATION} Default path to gatekeeper service files.
\end{itemize}

\section*{Files}

\begin{itemize}
\item /\texttt{etc/grid-services/SERVICENAME} Service configuration for \texttt{SERVICENAME}.
\item /\texttt{etc/grid-security/grid-mapfile} Default file mapping Grid identities to POSIX identities.
\item /\texttt{etc/globusmap} Default file mapping Grid identities to Kerberos 5 principals.
\item /\texttt{etc/globus-nologin} File to disable the \texttt{globus-gatekeeper} program.
\item /\texttt{var/log/globus-gatekeeper.log} Default gatekeeper log.
\end{itemize}

\section*{See also}

globus-k5(8), globusrun(1), globus-job-manager(8)
Name
globus-gatekeeper-admin — Manage globus-gatekeeper services

Synopsis
globus-gatekeeper-admin [-h]
globus-gatekeeper-admin [-l] [-n NAME]
globus-gatekeeper-admin [-e SERVICE] [-n NAME]
globus-gatekeeper-admin [-E]
globus-gatekeeper-admin [-d SERVICE]

Description
The globus-gatekeeper-admin program manages service entries which are used by the globus-gatekeeper to execute services. Service entries are located in the /etc/grid-services directory. The globus-gatekeeper-admin can list, enable, or disable specific services, or set a service as the default. The -h command-line option shows a brief usage message.

Listing services
The -l command-line option to globus-gatekeeper-admin will cause it to list all of the services which are available to be run by the globus-gatekeeper. In the output, the service name will be followed by its status in brackets. Possible status strings are ENABLED, DISABLED, and ALIAS to NAME, where NAME is another service name.

If the -n NAME is used, then only information about the service named NAME is printed.

Enabling services
The -e SERVICE command-line option to globus-gatekeeper-admin will cause it to enable a service so that it may be run by the globus-gatekeeper.

If the -n NAME option is used as well, then the service will be enabled with the alias NAME.

Enabling a default service
The -E command-line option to globus-gatekeeper-admin will cause it to enable a service alias with the name jobmanager. The globus-gatekeeper-admin program will choose the first service it finds as the default. To enable a particular service as the default, use the -e parameter described above with the -n parameter.

Disabling services
The -d SERVICE command-line option to globus-gatekeeper-admin will cause it to disable a service so that it may not be run by the globus-gatekeeper. All aliases to a disabled service are also disabled.

Files
/etc/grid-services Default location of enabled gatekeeper service descriptions.
Name

globus-gram-audit — Load GRAM4 and GRAM5 audit records into a database

Synopsis


Description

The globus-gram-audit program loads audit records to an SQL-based database. It reads $GLOBUS_LOCATION/etc/globus-job-manager.conf by default to determine the audit directory and then uploads all files in that directory that contain valid audit records to the database configured by the globus_gram_job_manager_auditing_setup_scripts package. If the upload completes successfully, the audit files will be removed.

The full set of command-line options to globus-gram-audit consist of:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--conf CONFIG_FILE</td>
<td>Use CONFIG_FILE instead of the default from the configuration file for audit database configuration.</td>
</tr>
<tr>
<td>--check</td>
<td>Check whether the insertion of a record was successful by querying the database after inserting the records. This is used in tests.</td>
</tr>
<tr>
<td>--delete</td>
<td>Delete audit records from the database right after inserting them. This is used in tests to avoid filling the database with test records.</td>
</tr>
<tr>
<td>--audit-directory DIR</td>
<td>Look for audit records in DIR, instead of looking in the directory specified in the job manager configuration. This is used in tests to control which records are loaded to the database and then deleted.</td>
</tr>
<tr>
<td>--query SQL</td>
<td>Perform the given SQL query on the audit database. This uses the database information from the configuration file to determine how to contact the database.</td>
</tr>
<tr>
<td>--quiet</td>
<td>Reduce the amount of output for common operations.</td>
</tr>
</tbody>
</table>

FILES

The globus-gram-audit uses the following files (paths relative to $GLOBUS_LOCATION).

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>etc/globus-gram-job-manager.conf</td>
<td>GRAM5 job manager configuration. It includes the default path to the audit directory</td>
</tr>
<tr>
<td>etc/globus-gram-audit.conf</td>
<td>Audit configuration. It includes the information needed to contact the audit database.</td>
</tr>
</tbody>
</table>
Name
globus-job-manager — Execute and monitor jobs

Synopsis

Description
The globus-job-manager program is a service which starts and controls GRAM jobs which are executed by a local resource management system, such as LSF or Condor. The globus-job-manager program is typically started by the globus-gatekeeper program and not directly by a user. It runs until all jobs it is managing have terminated or its delegated credentials have expired.

Typically, users interact with the globus-job-manager program via client applications such as globusrun, globus- job-submit, or tools such as CoG jglobus or Condor-G.

The full set of command-line options to globus-job-manager consists of:

- **-help**
  Display a help message to standard error and exit

- **-type LRM**
  Execute jobs using the local resource manager named LRM.

- **-conf CONFIG_PATH**
  Read additional command-line arguments from the file CONFIG_PATH. If present, this must be the first command-line argument to the globus-job-manager program.

- **-globus-host-manufacturer MANUFACTURER**
  Indicate the manufacturer of the system which the jobs will execute on. This parameter sets the value of the ${GLOBUS_HOST_MANUFACTURER} RSL substitution to MANUFACTURER

- **-globus-host-cputype CPUTYPE**
  Indicate the CPU type of the system which the jobs will execute on. This parameter sets the value of the ${GLOBUS_HOST_CPUTYPE} RSL substitution to CPUTYPE

- **-globus-host-osname OSNAME**
  Indicate the operating system type of the system which the jobs will execute on. This parameter sets the value of the ${GLOBUS_HOST_OSNAME} RSL substitution to OSNAME

- **-globus-host-osversion OSVERSION**
  Indicate the operating system version of the system which the jobs will execute on. This parameter sets the value of the ${GLOBUS_HOST_OSVERSION} RSL substitution to OSVERSION

- **-globus-gatekeeper-host HOST**
  Indicate the host name of the machine which the job was submitted to. This parameter sets the value of the ${GLOBUS_GATEKEEPER_HOST} RSL substitution to HOST
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-globus-gatekeeper-port PORT</code></td>
<td>Indicate the TCP port number of gatekeeper to which jobs are submitted to. This parameter sets the value of the $(GLOBUS_GATEKEEPER_PORT) RSL substitution to <code>PORT</code></td>
</tr>
<tr>
<td><code>-globus-gatekeeper-subject SUBJECT</code></td>
<td>Indicate the X.509 identity of the gatekeeper to which jobs are submitted to. This parameter sets the value of the $(GLOBUS_GATEKEEPER_SUBJECT) RSL substitution to <code>SUBJECT</code></td>
</tr>
<tr>
<td><code>-home GLOBUS_LOCATION</code></td>
<td>Indicate the path where the Globus Toolkit(r) is installed on the service node. This is used by the job manager to locate its support and configuration files.</td>
</tr>
<tr>
<td><code>-target-globus-location TARGET_GLOBUS_LOCATION</code></td>
<td>Indicate the path where the Globus Toolkit(r) is installed on the execution host. If this is omitted, the value specified as a parameter to <code>-home</code> is used. This parameter sets the value of the $(GLOBUS_LOCATION) RSL substitution to <code>TARGET_GLOBUS_LOCATION</code></td>
</tr>
<tr>
<td><code>-history HISTORY_DIRECTORY</code></td>
<td>Configure the job manager to write job history files to <code>HISTORY_DIRECTORY</code>. These files are described in the FILES section below.</td>
</tr>
<tr>
<td><code>-scratch-dir-base SCRATCH_DIRECTORY</code></td>
<td>Configure the job manager to use <code>SCRATCH_DIRECTORY</code> as the default scratch directory root if a relative path is specified in the job RSL’s <code>scratch_dir</code> attribute.</td>
</tr>
<tr>
<td><code>-enable-syslog</code></td>
<td>Configure the job manager to write log messages via syslog. Logging is further controlled by the argument to the <code>-log-levels</code> parameter described below.</td>
</tr>
<tr>
<td><code>-log-pattern PATTERN</code></td>
<td>Configure the job manager to write log messages to files named by the string <code>PATTERN</code>. The <code>PATTERN</code> string may contain job-independent RSL substitutions such as $(HOME), $(LOGNAME), etc, as well as the special RSL substitution $(DATE) which will be resolved at log time to the date in YYYYMMDD form.</td>
</tr>
<tr>
<td><code>-stdio-log LOG_DIRECTORY</code></td>
<td>Configure the job manager to write log messages to files in the <code>LOG_DIRECTORY</code> directory. This is a backward-compatible parameter, equivalent to <code>-log-pattern LOG_DIRECTORY/gram_$(DATE).log</code>.</td>
</tr>
<tr>
<td><code>-log-levels LEVELS</code></td>
<td>Configure the job manager to write log messages of certain levels to syslog and/or log files. The available log levels are FATAL, ERROR, WARN, INFO, DEBUG, and TRACE. Multiple values can be combined with the</td>
</tr>
<tr>
<td><code>-state-file-dir STATE_DIRECTORY</code></td>
<td>Configure the job manager to write state files to <code>STATE_DIRECTORY</code>. If not specified, the job manager uses the default of $GLOBUS_LOCATION/tmp/gram_job_state/. This directory must be writable by all users and be on a file system which supports POSIX advisory file locks.</td>
</tr>
<tr>
<td><code>-globus-tcp-port-range PORT_RANGE</code></td>
<td>Configure the job manager to restrict its TCP/IP communication to use ports in the range described by <code>PORT_RANGE</code>. This value is also made available in the job environment via the <code>GLOBUS_TCP_PORT_RANGE</code> environment variable.</td>
</tr>
<tr>
<td><code>-globus-tcp-source-range SOURCE_RANGE</code></td>
<td>Configure the job manager to restrict its TCP/IP communication to use source ports in the range described by <code>SOURCE_RANGE</code>. This value is also made</td>
</tr>
</tbody>
</table>
available in the job environment via the `GLOBUS_TCP_SOURCE_RANGE` environment variable.

- `x509-cert-dir` Configure the job manager to search `TRUSTED_CERTIFICATE_DIRECTORY` for its list of trusted CA certificates and their signing policies. This value is also made available in the job environment via the `X509_CERT_DIR` environment variable.

- `cache-location` Configure the job manager to use the path `GASS_CACHE_DIRECTORY` for its temporary GASS-cache files. This value is also made available in the job environment via the `GLOBUS_GASS_CACHE_DEFAULT` environment variable.

- `k` Configure the job manager to assume it is using Kerberos for authentication instead of X.509 certificates. This disables some certificate-specific processing in the job manager.

- `extra-envvars` Configure the job manager to define a set of environment variables in the job environment beyond those defined in the base job environment. The format of the parameter to this argument is a comma-separated sequence of `VAR=VAL` pairs, where `VAR` is the variable name and `VAL` is the variable's value. If the value is not specified, then the value of the variable in the job manager's environment is used. This option may be present multiple times on the command-line or the job manager configuration file to append multiple environment settings.

- `seg-module SEG_MODULE` Configure the job manager to use the schedule event generator module named by `SEG_MODULE` to detect job state changes events from the local resource manager, in place of the less efficient polling operations used in GT2. To use this, one instance of the `globus-job-manager-event-generator` must be running to process events for the LRM into a generic format that the job manager can parse.

- `audit-directory` Configure the job manager to write audit records to the directory named by `AUDIT_DIRECTORY`. This records can be loaded into a database using the `globus-gram-audit` program.

- `globus-toolkit-version` Configure the job manager to use `TOOLKIT_VERSION` as the version for audit and usage stats records.

- `service-tag SERVICE_TAG` Configure the job manager to use `SERVICE_TAG` as a unique identifier to allow multiple GRAM instances to use the same job state directories without interfering with each other's jobs. If not set, the value `untagged` will be used.

- `disable-streaming` Configure the job manager to disable file streaming. This is propagated to the LRM script interface but has no effect in GRAM5.

- `disable-usagestats` Disable sending of any usage stats data, even if `usagestats-targets` is present in the configuration.

- `usagestats-targets TARGET` Send usage packets to a data collection service for analysis. The `TARGET` string consists of a comma-separated list of HOST:PORT combinations, each containing an optional list of data to send. See Usage Stats Packets¹ for more information about the tags. Special tag strings of all (which enables all

¹ http://confluence.globus.org/display/~bester/GRAM5+Usage+Stats+Packets
tags) and default may be used, or a sequence of characters for the various
tags. If this option is not present in the configuration, then the default of us-
age-stats.globus.org:4810 is used.

-**condor-arch** **ARCH**

Set the architecture specification for condor jobs to be **ARCH** in job classi-
fied ads generated by the GRAM5 condor LRM script. This is required for the
condor LRM but ignored for all others.

-**condor-os** **OS**

Set the operating system specification for condor jobs to be **OS** in job classi-
fied ads generated by the GRAM5 condor LRM script. This is required for the
condor LRM but ignored for all others.

**Environment**

If the following variables affect the execution of **globus-job-manager**

**HOME**
User's home directory.

**LOGNAME**
User's name.

**JOBMANAGER_SYSLOG_ID**
String to prepend to syslog audit messages.

**JOBMANAGER_SYSLOG_FAC**
Facility to log syslog audit messages as.

**JOBMANAGER_SYSLOG_LVL**
Priority level to use for syslog audit messages.

**GATEKEEPER_JM_ID**
Job manager ID to be used in syslog audit records.

**GATEKEEPER_PEER**
Peer information to be used in syslog audit records

**GLOBUS_ID**
Credential information to be used in syslog audit records

**GLOBUS_JOB_MANAGER_SLEEP**
Time (in seconds) to sleep when the job manager is started. [For debugging
purposes only]

**GRID_SECURITY_HTTP_BODY_FD**
File descriptor of an open file which contains the initial job request and to
which the initial job reply should be sent. This file descriptor is inherited from
the **globus-gatekeeper**.

**X509_USER_PROXY**
Path to the X.509 user proxy which was delegated by the client to the **globus-
gatekeeper** program to be used by the job manager.

**GRID_SECURITY_CONTEXT_FD**
File descriptor containing an exported security context that the job manager
should use to reply to the client which submitted the job.

**GLOBUS_USAGE_TARGETS**
Default list of usagestats services to send usage packets to.

**GLOBUS_TCP_PORT_RANGE**
Default range of allowed TCP ports to listen on. The **-globus-tcp-
port-range** command-line option overrides this.

**GLOBUS_TCP_SOURCE_RANGE**
Default range of allowed TCP ports to bind to. The **-globus-tcp-
source-range** command-line option overrides this.

**Files**

$HOME/.globus/job/HOST-
NAME/LRM.TAG.red
Job manager delegated user credential.
<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$HOME/.globus/job/HOST-NAMELRM.TAG.lock</td>
<td>Job manager state lock file.</td>
</tr>
<tr>
<td>$HOME/.globus/job/HOST-NAMELRM.TAG.pid</td>
<td>Job manager pid file.</td>
</tr>
<tr>
<td>$HOME/.globus/job/HOST-NAMELRM.TAG.sock</td>
<td>Job manager socket for inter-job manager communications.</td>
</tr>
<tr>
<td>$HOME/.globus/job/HOST-NAMELRM.TAG/JOB_ID/</td>
<td>Job-specific state directory.</td>
</tr>
<tr>
<td>$HOME/.globus/job/HOST-NAMELRM/JOB_ID/stdin</td>
<td>Standard input which has been staged from a remote URL.</td>
</tr>
<tr>
<td>$HOME/.globus/job/HOST-NAMELRM/JOB_ID/stdout</td>
<td>Standard output which will be staged from a remote URL.</td>
</tr>
<tr>
<td>$HOME/.globus/job/HOST-NAMELRM/JOB_ID/stderr</td>
<td>Standard error which will be staged from a remote URL.</td>
</tr>
<tr>
<td>$HOME/.globus/job/HOST-NAMELRM/JOB_ID/x509_user_proxy</td>
<td>Job-specific delegated credential.</td>
</tr>
<tr>
<td>$GLOBUS_LOCATION/tmp/gram_job_state/job.HOSTNAME.JOB_ID</td>
<td>Job state file.</td>
</tr>
<tr>
<td>$GLOBUS_LOCATION/tmp/gram_job_state/job.HOSTNAME.JOB_ID.lock</td>
<td>Job state lock file. In most cases this will be a symlink to the job manager lock file.</td>
</tr>
<tr>
<td>$GLOBUS_LOCATION/etc/globus-job-manager.conf</td>
<td>Default location of the global job manager configuration file.</td>
</tr>
<tr>
<td>$GLOBUS_LOCATION/etc/grid-services/jobmanager-LRM</td>
<td>Default location of the LRM-specific gatekeeper configuration file.</td>
</tr>
<tr>
<td>$GLOBUS_LOCATION/etc/globus/gram/job--manager.rvf</td>
<td>Default location of the site-specific job manager RSL validation file.</td>
</tr>
<tr>
<td>$GLOBUS_LOCATION/etc/globus/gram/lrm.rvf</td>
<td>Default location of the site-specific job manager RSL validation file for the named lrm.</td>
</tr>
</tbody>
</table>

**See Also**

globusrun(1), globus-gatekeeper(8), globus-personal-gatekeeper(1), globus-gram-audit(8)
**Name**
globus-rvf-check — Edit a GRAM5 RSL validation file

**Synopsis**

globus-rvf-check [-h] [-help]
globus-rvf-check [-d] {FILENAME...}

**Description**

The **globus-rvf-check** command is a utility which checks the syntax of a RSL validation file, and prints out parse errors when encountered. It can also parse the RVF file contents and then dump file's contents to stdout, after canonicalizing values and quoting. The exit code of **globus-rvf-check** is 0 if all files specified on the command line exist and have no parse errors.

The full set of command-line options to **globus-rvf-check** consists of:

- `-h`, `--help`
  Print command-line option summary and exit

- `-d`
  Dump the RVF contents to stdout. In the output, Each file which is parsed will be prefixed by an RVF comment which contains the input filename. If not specified, **globus-rvf-check** just prints a diagnostic message to standard output indicating whether the file could be parsed.
**Name**  
globus-rvf-edit — Edit a GRAM5 RSL validation file

**Synopsis**

globus-rvf-edit [-h]

globus-rvf-edit [[-s] | [-l LRM] | [-f PATH]]

**Description**

The `globus-rvf-edit` command is a utility which opens the default editor on a specified RSL validation file, and then, when editing completes, runs the `globus-rvf-check` command to verify that the RVF file syntax is correct. If a parse error occurs, the user will be given an option to rerun the editor or discard the modifications.

The full set of command-line options to `globus-rvf-edit` consists of:

- `-h`  
  Print command-line option summary and exit

- `-s`  
  Edit of the site-specific RVF file, which provides override values applicable to all LRMs installed on the system.

- `-l LRM`  
  Edit the site-specific LRM overrides for the LRM named by the `LRM` parameter to the option.

- `-f PATH`  
  Edit the RVF file located at `PATH`
Name
globus-scheduler-event-generator — Process LRM events into a common format for use with GRAM

Synopsis
globus-scheduler-event-generator -s LRM
[-t TIMESTAMP] [-d DIRECTORY]
[-b] [-p PIDFILE]

Description

The globus-scheduler-event-generator program processes information from a local resource manager to generate LRM-independent events which GRAM can use to track job state changes. Typically, the globus-scheduler-event-generator is started at system boot time for all LRM adapters which have been installed. The only required parameter to globus-scheduler-event-generator is -s LRM, which indicates what LRM-specific module to load. A list of available modules can be found by using the globus-scheduler-event-generator-admin -l command.

Other options control how the globus-scheduler-event-generator program runs and where its output goes. These options are:

- **-t TIMESTAMP**  Start processing events which start at TIMESTAMP in seconds since the UNIX epoch. If not present, the globus-scheduler-event-generator will process events from the time it was started, and not look for historical events.

- **-d DIRECTORY**  Write the event log to files in DIRECTORY, instead of printing them to standard output. Within DIRECTORY, logs will be named by the time when they were created in YYYYMMDD format.

- **-b**  Run the globus-scheduler-event-generator program in the background.

- **-p PIDFILE**  Write the process-id of globus-scheduler-event-generator to PIDFILE.

Files

/var/lib/globus/globus-seg-LRM/YYYYMMDD  LRM-independent event log generated by globus-scheduler-event-generator

See Also

globus-scheduler-event-generator-admin(8), globus-job-manager(8)
Name
globus-scheduler-event-generator-admin — Manage SEG modules

Synopsis
globus-scheduler-event-generator-admin [-h]
globus-scheduler-event-generator-admin [-l]
globus-scheduler-event-generator-admin [-e MODULE]
globus-scheduler-event-generator-admin [-d MODULE]

Description
The globus-scheduler-event-generator-admin program manages SEG modules which are used by the globus
scheduler-event-generator to monitor a local resource manager or batch system for events. The globus-sched
uler-event-generator-admin can list, enable, or disable specific SEG modules. The -h command-line option shows
a brief usage message.

Listing SEG Modules
The -l command-line option to globus-scheduler-event-generator-admin will cause it to list all of the SEG mod-
dules which are available to be run by the globus-scheduler-event-generator. In the output, the service name will be
followed by its status in brackets. Possible status strings are ENABLED and DISABLED.

Enabling SEG Modules
The -e MODULE command-line option to globus-scheduler-event-generator-admin will cause it to enable the
module so that the init script for the globus-scheduler-event-generator will run it.

Disabling SEG Modules
The -d MODULE command-line option to globus-scheduler-event-generator-admin will cause it to disable the
module so that it will not be started by the globus-scheduler-event-generator init script.

Files
/etc/globus/sched
uler-event-generator  Default location of enabled SEG modules.

See Also
globus-scheduler-event-generator(8)
Chapter 8. Usage statistics collection by the Globus Alliance

1. GRAM5-specific usage statistics

The following usage statistics are sent by default in a UDP packet (in addition to the GRAM component code, packet version, timestamp, and source IP address) at the end of each job.

- Job Manager Session ID
- dryrun used
- RSL Host Count
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_UNSUBMITTED
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_FILE_STAGE_IN
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_FILE_STAGE_OUT
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE
- Job Failure Code
- Number of times status is called
- Number of times register is called
- Number of times signal is called
- Number of times refresh is called
- Number of files named in file_clean_up RSL
- Number of files being staged in (including executable, stdin) from http servers
- Number of files being staged in (including executable, stdin) from https servers
- Number of files being staged in (including executable, stdin) from ftp servers
- Number of files being staged in (including executable, stdin) from gsiftp servers
- Number of files being staged into the GASS cache from http servers
- Number of files being staged into the GASS cache from https servers
- Number of files being staged into the GASS cache from ftp servers
- Number of files being staged into the GASS cache from gsiftp servers
Usage statistics collection by the Globus Alliance

- Number of files being staged out (including stdout and stderr) to http servers
- Number of files being staged out (including stdout and stderr) to https servers
- Number of files being staged out (including stdout and stderr) to ftp servers
- Number of files being staged out (including stdout and stderr) to gsiftp servers
- Bitmask of used RSL attributes (values are $2^{\text{id}}$ from the gram5_rsl_attributes table)
- Number of times unregister is called
- Value of the count RSL attribute
- Comma-separated list of string names of other RSL attributes not in the set defined in \texttt{globus-gram-job-manager.rvf}
- Job type string
- Number of times the job was restarted
- Total number of state callbacks sent to all clients for this job

The following information can be sent as well in a job status packet but it is not sent unless explicitly enabled by the system administrator:

- Value of the executable RSL attribute
- Value of the arguments RSL attribute
- IP address and port of the client that submitted the job
- User DN of the client that submitted the job

In addition to job-related status, the job manager sends information periodically about its execution status. The following information is sent by default in a UDP packet (in addition to the GRAM component code, packet version, timestamp, and source IP address) at job manager start and every 1 hour during the job manager lifetime:

- Job Manager Start Time
- Job Manager Session ID
- Job Manager Status Time
- Job Manager Version
- LRM
- Poll used
- Audit used
- Number of restarted jobs
- Total number of jobs
- Total number of failed jobs
- Total number of canceled jobs
• Total number of completed jobs
• Total number of dry-run jobs
• Peak number of concurrently managed jobs
• Number of jobs currently being managed
• Number of jobs currently in the UNSUBMITTED state
• Number of jobs currently in the STAGE_IN state
• Number of jobs currently in the PENDING state
• Number of jobs currently in the ACTIVE state
• Number of jobs currently in the STAGE_OUT state
• Number of jobs currently in the FAILED state
• Number of jobs currently in the DONE state

Also, please see our policy statement¹ on the collection of usage statistics.

¹/toolkit/docs/latest-stable/Usage_Stats.html


Glossary

some terms not in the docs but wanted in glossary: Local Resource Manager TORQUE Oracle GridEngine Sun GridEngine Portable Batch System

C

certificate A public key plus information about the certificate owner bound together by the digital signature of a CA. In the case of a CA certificate, the certificate is self signed, i.e. it was signed using its own private key.

Condor A Local Resource Manager mechanism supported by GRAM. See the Condor Project Website\(^1\) for more information. See Also Local Resource Manager.

F

fork A POSIX-specific way of creating new processes. GRAM implements a basic fork LRM Adapter which runs jobs on the GRAM head node.

G

Gatekeeper A part of GRAM that runs as root and authenticates clients prior to starting the Job Manager.

grid map file A file containing entries mapping certificate subjects to local user names. This file can also serve as a access control list for GSI enabled services and is typically found in /etc/grid-security/grid-mapfile. For more information see the Gridmap section here.

Oracle GridEngine A Local Resource Manager supported by GRAM. See Oracle's Web Site\(^2\) for more information. See Also Local Resource Manager.

J

Job Manager A part of GRAM that runs as a local user and interfaces with a Local Resource Manager for that user.

L

Local Resource Manager (LRM) A system which controls access to a compute resource, such as a compute cluster or parallel computer. Such systems provide batch execution interfaces, which GRAM uses to execute jobs. Condor, Portable Batch System, GridEngine are examples of local resource managers. See Also Condor, Portable Batch System, Oracle GridEngine.

LRM Adapter The interface code between a Local Resource Manager and GRAM. In most cases, this consists of a Perl module that implements the

\(^1\) \url{http://www.cs.wisc.edu/condor/}

\(^2\) \url{http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html}
Glossary

Globus::GRAM::JobManager class and a Scheduler Event Generator module.
See Also Local Resource Manager.

P

Portable Batch System (PBS) A Local Resource Manager mechanism supported by GRAM. Multiple implementations of PBS exist: GRAM currently supports TORQUE. See also TORQUE.
See Also Local Resource Manager, TORQUE.

proxy certificate A short lived certificate issued using a EEC. A proxy certificate typically has the same effective subject as the EEC that issued it and can thus be used in its place. GSI uses proxy certificates for single sign on and delegation of rights to other entities.
For more information about types of proxy certificates and their compatibility in different versions of GT, see http://dev.globus.org/wiki/Security/ProxyCertificateTypes.

S

Scheduler Event Generator (SEG) The Scheduler Event Generator (SEG) is a program which uses scheduler-specific monitoring modules to generate job state change events. Depending on scheduler-specific requirements, the SEG may need to run with privileges to enable it to obtain scheduler event notifications. As such, one SEG runs per scheduler resource. For example, on a host which provides access to both PBS and fork jobs, two SEGs, running at (potentially) different privilege levels will be running. One SEG instance exists for any particular scheduled resource instance (one for all homogeneous PBS queues, one for all fork jobs, etc). The SEG is implemented in an executable called the globus-scheduler-event-generator, located in the Globus Toolkit's libexec directory.

Sun GridEngine (SGE) The old name for Oracle GridEngine.
See Also Oracle GridEngine.

T

TORQUE An implementation of the Portable Batch System interface by Adaptive Computing. ¹¹
See Also Portable Batch System.

Index

A
audit logging, 12

U
usage statistics, 33
GT 5.2.2 GRAM5: User's Guide
Introduction

GRAM services provide secure, remote job submission to different local resource managers in a Grid environment. This document describes the features of the GRAM service and an overview of tools to use the service.
# Table of Contents

1. GRAM5 Overview ................................................................. 1  
   1. GRAM Client Tools .......................................................... 1  
   2. Portals and Science Gateways that use GRAM ............................. 2  
2. Using GRAM5 ........................................................................ 3  
   1. Before Getting Started ....................................................... 3  
   2. Basic Client Interface ....................................................... 4  
   3. Advanced GRAM Client with the globusrun tool ......................... 8  
3. Troubleshooting .................................................................... 15  
   1. GRAM Client Troubleshooting ............................................. 15  
4. Command-line Client Reference Pages ..................................... 19  
   globusrun ........................................................................ 20  
   globus-job-cancel .................................................................. 24  
   globus-job-clean .................................................................. 25  
   globus-job-get-output ........................................................... 26  
   globus-job-run .................................................................... 28  
   globus-job-status .................................................................. 31  
   globus-job-submit .................................................................. 33  
5. GRAM RSL Quick Reference .................................................... 36  
6. GRAM Error Message Reference ............................................ 40  
   1. Errors .............................................................................. 40  
7. Known Problems in GRAM5 ...................................................... 53  
   1. Known Problems in GRAM5 ............................................... 53  
8. Usage statistics collection by the Globus Alliance ....................... 54  
   1. GRAM5-specific usage statistics ........................................... 54  
Glossary .................................................................................. 57  
Index ...................................................................................... 58
List of Tables

6.1. GRAM5 Errors .................................................................................................................. 40
List of Examples

2.1. Minimal job using globus-job-run .......................................................... 4
2.2. Multiprocess job using globus-job-run .................................................... 5
2.3. Staging an executable file using globus-job-run ...................................... 5
2.4. Providing an input file to a job using globus-job-run .............................. 5
2.5. Staging an input file to a job using globus-job-run ............................... 5
2.6. Canceling an interactive job ..................................................................... 5
2.7. Setting job environment variables with globus-job-run ......................... 6
2.8. globus-job-submit .................................................................................. 7
2.9. Using custom RSL clauses with globus-job-run ..................................... 7
2.10. Constructing RSL strings with globus-job-run ...................................... 8
2.11. Checking RSL Syntax ........................................................................... 8
2.12. GRAM Authentication test .................................................................... 9
2.13. GRAM version check ........................................................................... 9
2.14. Basic Interactive Job ............................................................................. 10
2.15. Basic Batch Job .................................................................................... 10
2.16. Refreshing a Credential ........................................................................ 10
2.17. Proxy Expiration Example ..................................................................... 11
2.18. File stage in .......................................................................................... 11
2.19. File stage in shared .............................................................................. 12
2.20. File stage out ........................................................................................ 12
2.21. Staging to scratch directory .................................................................... 12
2.22. Cleaning up a file .................................................................................. 12
2.23. Two phase commit example ................................................................... 13
2.24. Restart example ..................................................................................... 13
2.25. Java example .......................................................................................... 14
Chapter 1. GRAM5 Overview

GRAM provides a uniform, remote interface for executing jobs on compute resources. GRAM jobs consist of file transfers and program execution on one or more compute elements managed by a local resource manager. The GRAM client can submit the job and then later poll for its status, or it can request that the GRAM service notify it when the job changes state or completes. While the job is executing, the client may send control messages to the GRAM service to monitor or modify the job. GRAM provides reliable job submission, job recovery in case of service or client failures, file staging, and asynchronous notification messages.

GRAM achieves its uniform interface by implementing a domain-specific language called the Resource Specification Language (RSL) which provides a simple way to express job requirements, environment, and commands in a specification which is independent of the local resource manager which will actually execute the job.

The GRAM protocol is a two-phase protocol, so that when jobs are submitted to a GRAM service, they will not start until a client has received a contact handle to the job. The GRAM service will not clean up a job until it has received acknowledgment from the client that the job completion state has been received. In the case of transient errors, GRAM clients can reconnect to the GRAM service to determine job state, or to update information the job will need to stage output files.

The GRAM service has been built to work in the presence of client and service failures without losing state information about jobs. If a client exits and is restarted, it can request job state information, update URLs for output files to be staged to, and register a new address to receive job state callbacks. If a service exits and is restarted, it will resume processing all existing GRAM jobs from their previous state, and continue to send state updates to any clients which are registered for them.

GRAM provides file staging before and after a job runs, scratch directory management, and a cache location for common files. File staging is Grid-aware and access remote storage resources via the GridFTP, ftp https, http protocols.

Because the GRAM service implements client callbacks for job state changes, clients can submit a number of jobs and be notified when each completes. This allows clients to be more responsive to changes in state than services which require polling for job completion.

1. GRAM Client Tools

There are a number of GRAM clients which can be used to interact with the GRAM service. The Globus Toolkit includes globusrun, globus-job-submit, and globus-job-run. Other projects provide higher-level tools which can be used to manage large sets of jobs.

1.1. Condor-G

Condor is a high-throughput job scheduler from the University of Wisconsin. It provides a facility called Condor-G to run jobs via GRAM. See the Condor documentation\(^1\), especially the section on Grid Universe, which describes how to write Condor Classified Ads to run jobs using GRAM services. The \texttt{gt5} Grid type provides the best performance for using GRAM with Condor-G.

1.2. Swift

The Swift\(^2\) system from the University of Chicago is a data-oriented coarse grained scripting language that supports dataset typing and mapping, dataset iteration, conditional branching, and procedural composition. The SwiftScript

\(^1\) http://research.cs.wisc.edu/condor/manual/latest/index.html
\(^2\) http://www.ci.uchicago.edu/swift/main/
language can be used to create workflows that are executed on various services, including GRAM. See the Swift User's Guide\(^3\) for information about using Swift.

### 1.3. GridWay

The GridWay Metascheduler\(^4\) enables large-scale, reliable and efficient sharing of computing resources: clusters, supercomputers, stand-alone servers. It supports different LRM systems (PBS, SGE, LSF, Condor) within a single organization or scattered across several administrative domains. The GridWay\(^5\) manual describes how to use GridWay.

### 1.4. GRAM APIs

In addition to the tools above, you can write your own GRAM clients, using the public APIs described in the GRAM5 Developer's Guide. The client APIs there can be used to write custom applications that interact with GRAM services in C/C++ or Java.

### 2. Portals and Science Gateways that use GRAM

#### 2.1. XSEDE

XSEDE\(^6\) provides a number of domain-specific science gateways and portals\(^7\), which provide interfaces to various computation and data resources, including some managed by GRAM.

These include CGD's Atmospheric Modeling & Predictability Section\(^8\) from NCAR, the UltraScan LIMS Portal\(^9\) at the Bioinformatics Core Facility at the University of Texas Health Science Center at San Antonio, the Social Informatics Data (SID) Grid\(^10\) at the University of Chicago, and Southern California Earthquake Center\(^11\) headquartered at the University of Southern California.

---

\(^3\) http://www.ci.uchicago.edu/swift/guides/userguide.php
\(^4\) http://www.gridway.org/
\(^5\) http://www.gridway.org/doku.php?id=documentation
\(^6\) https://www.xsede.org
\(^7\) https://www.xsede.org/gateways-overview
\(^8\) http://www.cgd.ucar.edu/amp/
\(^9\) http://uslims.uthscsa.edu
\(^10\) https://sidgrid.ci.uchicago.edu
\(^11\) http://scec.org
Chapter 2. Using GRAM5

1. Before Getting Started

1.1. GRAM and Security

GRAM uses the Grid Security Infrastructure for its security implementation, based on X.509 certificates and the TLSv1 protocol to authenticate user identities with GRAM services. Before using GRAM, you must first obtain a security credential. This is typically done by requesting a certificate from a site-specific CA, or by using a portal to obtain a temporary credential. In typical use, GRAM uses a proxy certificate which is a short-term credential digitally signed by a private key. Please read the Section 1, “Basic procedure for using GSI C” to learn more about how to obtain and use a GSI credential before continuing this guide.

1.2. GRAM Resource Names

Before interacting with a GRAM service, you must know its contact address. GRAM uses a very flexible URL-like syntax to contain information about the service's hostname, TCP port number, service name, and security identity. In the basic case, you will only need to use the service's hostname to contact the service. However, if the service is configured to run on a non-standard port, or with a custom service name, or credential which doesn't match its hostname, you will need to use one of the longer forms.

A fully-qualified resource name looks something like grid.example.org:2120/jobmanager-sge:/C=US/O=Example/OU=Grid/CN=host/grid.example.org. Breaking this down, the resource name includes:

<table>
<thead>
<tr>
<th>Name Component</th>
<th>Example</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Name</td>
<td>grid.example.org</td>
<td>Host which the GRAM service is running on.</td>
<td>None. This is always a required component.</td>
</tr>
<tr>
<td>TCP Port</td>
<td>2120</td>
<td>TCP port which the GRAM service is listening on. If multiple GRAM services are running on the same machine, they may use alternate TCP ports.</td>
<td>2119</td>
</tr>
<tr>
<td>Service Name</td>
<td>jobmanager-sge</td>
<td>The name of the GRAM service on the given host. A host may provide access to multiple resources using different local resource managers. This name is used to distinguish which service to use for a particular job request. Typically, a host will provide a default entry called jobmanager which will interface</td>
<td>jobmanager</td>
</tr>
</tbody>
</table>

1 http://www.ietf.org/rfc/rfc3820.txt
Using GRAM5

<table>
<thead>
<tr>
<th>Name Component</th>
<th>Example</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>with a batch computing or high-throughput scheduling system, and another called jobmanager-fork for simple non-compute jobs.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Credential Name</td>
<td>/C=US/O=Example/OU=Grid/CN=host/grid.example.org</td>
<td>The name of the credential which the GRAM service is using. This is only needed if the credential's common name does not match the host name.</td>
<td>host@hostname</td>
</tr>
</tbody>
</table>

Any component of the resource name may be omitted, except for the host name, and defaults will be used. The field separator : must be retained when skipping between name components.

2. Basic Client Interface

This section contains the basic command-line interface for interacting with gram services. For these examples, we will use the GRAM resource named grid.example.org:2119/jobmanager-pbs. You will need to change that to resources which you have been granted access.

2.1. Batch and Interactive Use

The tools `globus-job-run` and `globus-job-submit` an both be used to submit jobs to GRAM resources. The difference is that `globus-job-run` will wait until the job terminates before exiting and prints job standard output and stderr after the job completes, while `globus-job-submit` will submit the job and then exit immediately, printing the job contact to its standard output stream. The job can be then polled for status with the `globus-job-status` command, its output can be fetched with the `globus-job-get-output` and cleaned up with the `globus-job-clean` command.

2.2. Running Basic Jobs with `globus-job-run`

The `globus-job-run` provides a simple blocking command-line interface to the GRAM service. The `globus-job-run` program submits a job to a GRAM resource and waits for the job to terminate. After the job terminates, the output and error streams of the job are sent to the output and error streams of `globus-job-run`. Note that truly interactive jobs are not supported with GRAM.

The `globus-job-run` program has command-line options to control most aspects of jobs run by GRAM. However, certain behaviors must be specified by definition of an RSL string containing various job attributes. A more detailed description about the RSL language is included on the section on running jobs with `globusrun` below.

The following examples show some of the common command-line options to `globus-job-run`. Full `globus-job-run` documentation is available in the GRAM5 public interface guide.

Example 2.1. Minimal job using `globus-job-run`

The following command line submits a single instance of the `/bin/hostname` executable to the resource named by grid.example.org/jobmanager-pbs.

```
% globus-job-run grid.example.org/jobmanager-pbs /bin/hostname
node1.grid.example.org
```
Example 2.2. Multiprocess job using globus-job-run

The following command line submits ten instances of an executable /bin/hostname. The output of the job is the name of the ten hosts that the job ran on. The -np COUNT option causes globus-job-run to run COUNT instances of the executable.

% globus-job-run grid.example.org/jobmanager-pbs -np 4 /bin/hostname
node1.grid.example.org
node3.grid.example.org
node2.grid.example.org
node10.grid.example.org

Example 2.3. Staging an executable file using globus-job-run

The following command line submits an executable which is local to the submit machine to the GRAM resource, then executes it. The executable is removed automatically from the GRAM resource after the job completes. The -s option prior to the executable name causes globus-job-run to stage the executable using GASS (an https-based protocol) from the machine running globus-job-run to the GRAM resource.

% globus-job-run grid.example.org/jobmanager-pbs -s my-executable
node1.grid.example.org

Example 2.4. Providing an input file to a job using globus-job-run

The following command line submits a job to a GRAM resource. When this job runs, its standard input will read from the file $HOME/inputfile.txt, which is located on the GRAM resource. The -stdin command-line option indicates this path.

% globus-job-run grid.example.org/jobmanager-pbs -stdin inputfile.txt /bin/cat
Hello, Grid

Example 2.5. Staging an input file to a job using globus-job-run

The following command line submits a job to a GRAM resource. When this job runs, its standard input will read from the file inputfile.txt, which is located on the submit client machine. The -stdin -s command-line option combination causes the input to be staged in the above executable staging example.

% globus-job-run grid.example.org/jobmanager-pbs -stdin -s inputfile.txt /bin/cat
Hello, staged input on the Grid

Example 2.6. Canceling an interactive job

This example shows how using the Control+C (or other system-specific mechanism for sending the SIGINT signal) can be used to cancel a GRAM job.

% globus-job-run grid.example.org/jobmanager-pbs /bin/sleep 90
Control+C
GRAM Job failed because the user cancelled the job (error code 8)
Example 2.7. Setting job environment variables with globus-job-run

The following command line submits one instances of the executable /usr/bin/env, setting some environment variables in the job environment beyond those set by GRAM5. The -env VARIABLE=VALUE command-line option adds the named variable to the job environment. It may be present multiple times in the command-line to set multiple environment variables.

```
% globus-job-run grid.example.org/jobmanager-pbs -env TEST=1 -env GRID=1 /usr/bin/env
HOME=/home/juser
LOGNAME=juser
GLOBUSGRAM_JOB_CONTACT=https://client.example.org:3882/16001579536700793196/5295612977485997184/
GLOBUS_LOCATION=/opt/globus-5.2.2
GLOBUSGASSCACHEDEFAULT=/home/juser/.globus/.gass_cache
TEST=1
X509USER_PROXY=/home/juser/.globus/job/mactop.local/16001579536700793196.5295612977485997184/
GRID=1
```

2.3. Submitting Basic Jobs with globus-job-submit

A related tool to globus-job-run is globus-job-submit. This command submits a job to a GRAM5 service then exits without waiting for the job to terminate. Other tools (globus-job-cancel, globus-job-clean, and globus-job-get-output) allow further interaction with the job.

⚠️ Important

When using globus-job-submit, the job output and state will remain on disk on the GRAM resource until one of globus-job-clean or globus-job-cancel is run for that job. Be sure to clean up your jobs!

The globus-job-submit program has most of the same command-line options as globus-job-run. When run, instead of displaying the output and error streams of the job, it prints the job contact, which is used with the other globus-job tools to interact with the job.
Using GRAM5

Example 2.8. globus-job-submit

This example shows the interaction of submitting a job via `globus-job-submit`, checking its status with `globus-job-status`, getting its output with `globus-job-get-output`, and then cleaning the job with `globus-job-clean`. Note that this example uses the `jobmanager-fork` service when retrieving input and cleaning the job. This allows those tasks to be done without waiting in the batch system. Most sites will allow these sorts of administrative jobs to be run on the GRAM node, but consult your system administrator to be sure.

Also, note, that the job contact returned from `globus-job-submit` can be used to get information about the job from any computer, provided you have GRAM tools installed and your security environment set up.

```
globus-job-submit grid.example.org/jobmanager-pbs /bin/hostname
https://grid.example.org:38843/16001600430615223386/5295612977486013582/
globus-job-status https://grid.example.org:38843/16001600430615223386/5295612977486013582/
PENDING
% globus-job-status https://grid.example.org:38843/16001600430615223386/5295612977486013582/
ACTIVE
% globus-job-status https://grid.example.org:38843/16001600430615223386/5295612977486013582/
DONE
% globus-job-get-output -r grid.example.org/jobmanager-fork 
   https://grid.example.org:38843/16001600430615223386/5295612977486013582/
   node1.grid.example.org
% globus-job-clean -r grid.example.org/jobmanager-fork 
   https://grid.example.org:38843/16001600430615223386/5295612977486013582/

WARNING: Cleaning a job means:
- Kill the job if it still running, and
- Remove the cached output on the remote resource

Are you sure you want to cleanup the job now (Y/N) ?
y
Cleanup successful.
```

2.4. Advanced Jobs with globus-job-run

Example 2.9. Using custom RSL clauses with globus-job-run

The following command line submits an mpi job using `globus-job-run`, setting the `jobtype` RSL attribute to `mpi`. Any RSL attribute understood by the LRM can be added to a job via this method.

```
globus-job-run grid.example.org/jobmanager-pbs -np 5 -x '&(jobtype=mpi)' a.out
Hello, MPI (rank: 0, count: 5)
Hello, MPI (rank: 3, count: 5)
Hello, MPI (rank: 1, count: 5)
Hello, MPI (rank: 4, count: 5)
Hello, MPI (rank: 2, count: 5)
```
Example 2.10. Constructing RSL strings with globus-job-run

The *globus-job-run* program can also generate the RSL language description of a job based on the command-line options given to it. This example combines some of the features above and prints out the resulting RSL. This RSL string can be passed to tools such as *globusrun* to be run later.

```
% globus-job-run -dumprsl grid.example.org/jobmanager-pbs -np 5 -x '&(jobtype=mpi)' -env GRID=1 -env TEST=1 a.out
&(jobtype=mpi)
  (executable="a.out")
  (environment= ("GRID" "1") ("TEST" "1"))
  (count=5)
```

3. Advanced GRAM Client with the globusrun tool

The *globusrun* tool provides a more flexible tool for submitting, monitoring, and canceling jobs. With this tool, most of the functionality of the GRAM5 APIs are made available from the command-line.

One major difference between *globusrun* and the other tools described above is that *globusrun* uses the RSL language to provide the job description, instead of multiple command-line options to describe the various aspects of the job. The section on *globus-job-run* contained a brief example RSL in the -dumprsl example above.

The following sections show examples of the different modes that *globusrun* can run in. Full information about *globusrun* command-line options is available in the public interface guide.

3.1. Checking RSL Syntax

This example shows how to check that an RSL document contains a syntactically correct job description. Note that this mode does not do semantic validation of the RSL, so an RSL document that passes this test may not work when submitted to a GRAM5 service.

**Example 2.11. Checking RSL Syntax**

```
% globusrun -p "& (executable=a.out)"
RSL Parsed Successfully...

% globusrun -p "/executable=a.out"
ERROR: cannot parse RSL "/executable=/adfadf/adf /adf /adf)
Syntax: globusrun [-help] [-f RSL file] [-s][-b][-d][...] [-r RM] [RSL]
```

Use -help to display full usage

3.2. Checking Service Contacts

This example shows how to check that a *globus-gatekeeper* is running at a particular contact and that the client and service have mutually-trusted credentials.
Example 2.12. GRAM Authentication test

% globusrun -a -r grid.example.org/jobmanager-pbs
GRAM Authentication test successful
% globusrun -a -r grid.example.org/jobmanager-lsf
GRAM Authentication test failure: the gatekeeper failed to find the requested service
% globusrun -a -r grid.example.org/jobmanager-pbs:host@not.example.org
GRAM Authentication test failure: an authorization operation failed
globus_xio_gsi: gss_init_sec_context failed.
GSS Major Status: Unexpected Gatekeeper or Service Name
globus_gsi_gssapi: Authorization denied: The name of the remote host
(host@not.example.org), and the expected name for the remote host
(grid.example.org) do not match. This happens when the name in the host
certificate does not match the information obtained from DNS and is often a DNS
configuration problem.

Note

The DNS configuration problem was a common issue in GRAM2, but GRAM5 will not depend on DNS to
resolve names for mutual authentication.

3.3. Checking GRAM service version

This example shows how to determine what software version of GRAM5 is deployed at a particular service contact.

Example 2.13. GRAM version check

% globusrun -j -r grid.example.org/jobmanager-pbs:host@not.example.org
Toolkit version: 4.3.0-HEAD
Job Manager version: 10.5 (1256257907-0)

Note

This example shows the version number for an unreleased development version of GRAM5. The actual
numbers returned will be different.

Note

This feature is new in GRAM5. When contacting a GRAM2 service, globusrun will display the following
error message:

GRAM version check failed : an incoming HTTP message did not contain the expected inf

3.4. Basic Interactive job with globusrun

This example shows how to submit interactive job with globusrun. When the -s is used, the output of the job com-
mand is returned to the client and displayed as if the command ran locally. This is similar to the behavior of the
globus-job-run program described above.
Using GRAM5

Example 2.14. Basic Interactive Job

% globusrun -s -r example.grid.org/jobmanager-pbs "&(executable=/bin/hostname)(count=5)"
node03.grid.example.org
node01.grid.example.org
node02.grid.example.org
node05.grid.example.org
node04.grid.example.org

3.5. Basic batch job with globusrun

This example shows how to submit, monitor, and cancel a batch job using globusrun. This method is useful for the case where the job may run for a long time, the job may be queued for a long time, or when there are network reliability issues between the client and service.

Example 2.15. Basic Batch Job

% globusrun -b -r grid.example.org/jobmanager-pbs "&(executable=/bin/sleep)(arguments=500)"
globus_gram_client_callback_allow successful
GRAM Job submission successful
https://grid.example.org:38824/16001608125017717261/5295612977486019989/
GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING
% globusrun -status https://grid.example.org:38824/16001608125017717261/5295612977486019989/
PENDING
% globusrun -k https://grid.example.org:38824/16001608125017717261/5295612977486019989/

3.6. Refreshing a GRAM5 Credential

The following example shows how to refresh the credential used by a job manager and a job.

Example 2.16. Refreshing a Credential

% globusrun -refresh-proxy https://grid.example.org:38824/16001608125017717261/5295612977486019989/
% echo $?
0

Note

In GT 5.2.2, globusrun does not print any diagnostics when given the -refresh-proxy command-line option. Therefore, check the exit code as above to ensure that the refresh is successful.

3.7. Dealing with credential expiration

When the Job Manager's credential is about to expire, it sends a message to all clients registered for GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED notifications that the job manager is terminating and that the job will continue to run without the job manager.

Any client which receives such a message can (if necessary) generate a new proxy as described above and then submit a restart request to start a job manager with a new credential. This job manager will resume monitoring the jobs which were started prior to proxy expiration.

In this example, the globusrun displays an error message when the job manager's proxy is about to expire. The user creates a new proxy and resumes monitoring the job with globusrun.
Example 2.17. Proxy Expiration Example

```
% globusrun -r grid.example.org "&(executable=a.out)"
globus_gram_client_callback_allow successful
GRAM Job submission successful
GLOBUSGRAM_PROTOCOL_JOB_STATE_ACTIVE
GLOBUSGRAM_PROTOCOL_JOB_STATE_FAILED
GRAM Job failed because the user proxy expired (job is still running) (error code 131)
% grid-proxy-init
Your identity: /DC=org/DC=example/OU=grid/CN=Joe User
Enter GRID pass phrase for this identity:
Creating proxy .......................................................... Done
Your proxy is valid until: Tue Nov 10 04:25:03 2009
% globusrun -r grid.example.org "&(restart="https://grid.example.org:1997/16001700477575114131/5295612977486005428/"
gram_gram_client_callback_allow successful
GRAM Job submission successful
GLOBUSGRAM_PROTOCOL_JOB_STATE_ACTIVE
GLOBUSGRAM_PROTOCOL_JOB_STATE_DONE
```

3.8. File staging

In addition to the standard output and error stream output done by `globusrun`, GRAM5 can do basic file management tasks to stage files to the GRAM5 service node before submitting a job and to stage files from the GRAM5 service node to a file service after the job completes.

GRAM5 file staging supports four URL schemes: `ftp`, `gsiftp`, `http`, and `https`. Note, that for the `https` scheme, GRAM expects the file server to be running with the same identity as the client.

General file staging is controlled by three RSL attributes: `file_stage_in`, `file_stage_in_shared`, and `file_stage_out`. In addition, the files named by the RSL attributes `executable`, `stdin` may be staged in and the files named by the RSL attributes `stdout` and `stderr` may be staged out.

The `file_stage_in_shared` RSL attribute instructs GRAM to store a local copy of the resource named by the URL in the GASS cache. This is useful if multiple concurrent jobs will be accessing one or more common files. The GASS cache will manage a reference count for files in the cache and remove them when all jobs that refer to them complete.

The following example shows how to stage a few files from a GridFTP server to the GRAM node. It uses the `rsl_substitution` mechanism to define a substitution variable to reduce the amount of redundancy in the job description.

Example 2.18. File stage in

```
% globusrun -s -r grid.example.org/jobmanager-pbs \n"&(rsl_substitution = (GRIDFTP_SERVER gsiftp://gridftp.example.org)) \n(executable=/bin/ls) \n(arguments=/tmp/staged_file) \n(file_stage_in = ($GRIDFTP_SERVER)/staged_file /tmp/staged_file))" 
/tmp/staged_file
```

The next example uses the `file_stage_in_shared` RSL attribute to stage a file into the cache. The file is transferred from the client using the GASS `https` server embedded in the `globusrun` program when the `-s` option is used.
Example 2.19. File stage in shared

```bash
% globusrun -s -r grid.example.org/jobmanager-pbs \
  "& (executable=/bin/ls) \ 
    (arguments = -l /tmp/staged_file_link1 /tmp/staged_file_link1) \ 
    (file_stage_in_shared = \ 
      (GLOBUSRUN_GASS_URL)/staged_file1 /tmp/staged_file_link1))"
```

lrwxr-xr-x 1 juser juser 120 Nov 11 20:37 /tmp/staged_file1 -> /home/juser/.globus/.gass_cache/local/md5/ff/771bded8a2c7dacc1a1c0fecafa0ce/md5/39/13ab3db7fc002ed54012083ae6ed1c/data

The final staging example uses the `file_stage_out` RSL attribute to transfer a file from the GRAM service to an FTP server using anonymous FTP.

Example 2.20. File stage out

```bash
% globusrun -r grid.example.org/jobmanager-pbs \
  "&(executable=a.out) \ 
    (file_stage_out = (results.txt ftp://anonymous:nopass@ftp.example.org/incoming/results.txt))"
```

Note

In all of the above cases, multiple files may be staged using any combination of the supported URL schemes.

3.9. Temporary files and cleanup

GRAM5 supports creating a per-job scratch directory which can be used as a place to store files that will be automatically removed by GRAM when the job completes. It also supports an explicit list of files to remove when the job completes.

This example shows how to stage files into a scratch directory. It again uses the embedded GASS https server, stages to the GRAM service, then runs `/bin/ls` in the temporary directory. After the job completes, the contents of `$(SCRATCH_DIRECTORY)` and the directory itself are removed.

Example 2.21. Staging to scratch directory

```bash
% globusrun -s grid.example.org/jobmanager-pbs \
  "&(scratch_dir = $(HOME)) \ 
    (directory = $(SCRATCH_DIRECTORY)) \ 
    (file_stage_in = \ 
      (GLOBUSRUN_GASS_URL)/inputfile $(SCRATCH_DIRECTORY)/inputfile)) \ 
    (executable = /bin/ls)"
```

inputfile

This example shows how to explicitly remove a file that was created by the job.

Example 2.22. Cleaning up a file

```bash
% globusrun -s grid.example.org/jobmanager-pbs \
  "&(executable = /bin/touch) \ 
    (arguments = temporary_file) \ 
    (file_clean_up = temporary_file)"
```

%
3.10. Reliable job submit

The `globusrun` command supports a two-phase commit protocol to ensure that the client knows the contact of the job which has been created so that it can be monitored or canceled in the case of a client or service error. The two-phase commit affects both job submission and termination.

The two-phase protocol is enabled by using the `two_phase` RSL attribute, as in the next example. When this is enabled, job submission will fail with the error `GLOBUS_GRAM_PROTOCOL_ERROR_WAITING_FOR_COMMIT`. The client must respond to this signal with either the `GLOBUS_GRAM_PROTOCOL_JOB_SIGNAL_COMMIT_REQUEST` or `GLOBUS_GRAM_PROTOCOL_JOB_SIGNAL_COMMIT_EXTEND` signals to either commit the job to execution or delay the commit timeout. One of these signals must be sent prior to the two phase commit timeout, or the job will be discarded by the GRAM service.

A two phase protocol is also used at job termination if the `save_state` RSL attribute is used along with the `two_phase` attribute. When the job manager sends a callback with the job state set to `GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE` or `GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE`, it will wait to clean up the job until the two phase commit occurs. The client must reply with the `GLOBUS_GRAM_PROTOCOL_JOB_SIGNAL_COMMIT_END` signal to cause the job to be cleaned. Otherwise, the job will be unloaded from memory until a client restarts the job and sends the signal.

Example 2.23. Two phase commit example

In this example, the user submits a job with a `two_phase` timeout of 30 seconds and the `save_state` attribute. The client must send commit signals to ensure the job runs.

```
% globusrun -r grid.example.org/jobmanager-pbs \
  "&(two_phase = 30) \ 
    (save_state = yes) \ 
    (executable = a.out)"

globus_gram_client_callback_allow successful
GRAM Job submission successful
GLOBUSGRAM_PROTOCOL_JOB_STATE_PENDING
GLOBUSGRAM_PROTOCOL_JOB_STATE_ACTIVE
GLOBUSGRAM_PROTOCOL_JOB_STATE_DONE
%
```

3.11. Reconnecting to a job

If a job manager or client exits before a job has completed, the job will continue to run. The client can reconnect to a job manager and receive job state notifications and output using the `restart` RSL attribute.

Example 2.24. Restart example

This example uses `globus-job-submit` to submit a batch job and then `globusrun` to reconnect to the job.

```
% globus-job-submit grid.example.org/jobmanager-pbs /bin/sleep 90
https://grid.example.org:38824/16001746665595486521/5295612977486005662/
% globusrun -r grid.example.org/jobmanager-pbs \ 
  "&(restart = https://grid.example.org:38824/16001746665595486521/5295612977486005662/)

globus_gram_client_callback_allow successful
GRAM Job submission successful
GLOBUSGRAM_PROTOCOL_JOB_STATE_DONE
%
```
3.12. Submitting a Java job

To submit a job that runs a java program, the client must ensure that the job can find the Java interpreter and its classes. This example sets the default PATH and CLASSPATH environment variables and uses the shell to locate the path to the java program.

Example 2.25. Java example

This example uses globus-job-submit to submit a java job, staging a jar file from a remote service.

```bash
% globusrun -r grid.example.org/jobmanager-pbs "
  "& (environment = (PATH '/usr/bin:/bin') (CLASSPATH $(SCRATCH_DIRECTORY)))
  (scratch_dir = $(HOME))
  (directory = $(SCRATCH_DIRECTORY))
  (rsl_substitution = (JAVA_SERVER http://java.example.org))
  (file_stage_in =
    ($(JAVA_SERVER)/example.jar $(SCRATCH_DIRECTORY)/example.jar)
    ($(JAVA_SERVER)/support.jar $(SCRATCH_DIRECTORY)/support.jar))
  (executable=/bin/sh)
  (arguments=-c 'java -jar example.jar')"
Gram Job submission successful
GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE
GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE
%
```
Chapter 3. Troubleshooting

1. GRAM Client Troubleshooting

1.1. Credential Problems

GRAM requires a client certificate and private key in order to authenticate with the GRAM service. If these are not available, the GRAM client will fail. In typical use, a user will create a temporary proxy certificate either derived from their identity certificate issued by some certificate authority, or from a service such as myproxy. If a GRAM client command returns any error containing the string GSS Major Status you've hit a credential problem.

Look at the Troubleshooting Section of the GSI manual for details about how to diagnose and correct these errors. The grid-cert-diagnostics tool with the -p command-line option is especially helpful for diagnosing some of these types of problems.

1.2. Connection Problems

There are a few things which can go wrong when trying to contact a GRAM service. These have slightly different error types which can help diagnose which problem is occurring.

1.2.1. Invalid Resource Name

If the hostname or TCP port you are using for a GRAM resource name is not correct, then the GRAM client will be unable to access the service. Errors of this type will look like this:

```
% globus-job-run grid.example.org/jobmanager-fork /bin/hostname
GRAM Job submission failed because the connection to the server failed (check host and port) (error code 12)
```

When this occurs, check with the resource administrator for correct resource naming so that you can contact the service.

1.2.2. Mutual Authentication Failure

GRAM performs mutual authentication, that is, both the client and service provide certificates indicating who they are. The service uses the client's identity to map the user to a local unix account. The client uses the server's identity to verify that the service is running with a host credential. The failure of the client to trust the server's certificate will generate an error message that looks like this: globus_gsi_gssapi: Authorization denied: The expected name for the remote host (host@alias.example.org) does not match the authenticated name of the remote host (host@grid.example.org). This happens when the name in the host certificate does not match the information obtained from DNS and is often a DNS configuration problem.

This mismatch can happen for a number of reasons: a site administrator has multiple hosts sharing a certificate, a host has multiple DNS aliases, and the client is not aware of which name the server is using for its certificate, or a host's name has changed since the certificate was issued. The remedy for the client, after confirming with the GRAM administrator that the name after "authenticated name of the remote host" is the correct certificate name is to use a form of the GRAM resource name which includes this name. For example, explicitly adding a name to the abbreviated GRAM contact so that instead of alias.example.org, you would use alias.example.org::host@grid.example.org.
1.2.3. Certificate Trust Issues

Because of the mutual authentication, both GRAM users and services can hit problems if they do not trust their peer's certificate or the Certificate Authority which issued it. If the client doesn't trust the server's certificate, it is easier to diagnose, because the GRAM service doesn't send much information back to the client if it doesn't trust it. However, working with the system administrator to get information from the GRAM logs will usually fix these problems fairly easily.

If the service's certificate is not trusted, the client will receive a message like this:

```bash
% globus-job-run grid.example.org /bin/hostname
GRAM Job submission failed because an authentication operation failed
OpenSSL Error: s3_clnt.c:915 in library: SSL routines, function SSL3_GET_SERVER_CERTIFICATE
globus_gsi_callback_module: Could not verify credential
globus_gsi_callback_module: Can't get the local trusted CA certificate: Untrusted self-signed certificate
```

This error indicates that certificate chain from the service certificate to the client contained a self-signed certificate (usually an indication that it's a CA certificate), which the client doesn't trust, and includes the hash of the certificate name (bbfceedf in this case). If you hit this particular type of error, you should send the information to the GRAM administrator and determine which CA should be trusted and what its signing policy is, to determine if you want to add it to your local set of trust roots.

**Note**

Different versions of OpenSSL produce different hashes for the same certificate names. If you upgrade a system (or transfer CA certificates between systems) to a different version of OpenSSL, you may hit this problem even if you think you have the CA certificate in your trusted certificate directory. If so, run the `globus-update-certificate-dir` program to update your hashes.

There are other reasons why a certificate might not be trusted (it's in a revoked list, it has expired or was issued in the future, etc). For more details look at the troubleshooting information in the GSI user's guide.

If for some reason the service does not trust your certificate, you'll get a rather cryptic message from GRAM that looks like this:

```bash
% globus-job-run grid.example.org /bin/hostname
GRAM Job submission failed because an authentication operation failed
globus_gsi_gssapi: Unable to verify remote side's credentials
globus_gsi_gssapi: Unable to verify remote side's credentials: Couldn't verify the remote
OpenSSL Error: s3_pkt.c:1086 in library: SSL routines, function SSL3_READ_BYTES: sslv3 alert bad certificate
```

To remedy this, consult the GRAM administrator to get information from the `/var/log/globus-gatekeeper.log` file to determine the reason why the gatekeeper didn't like your certificate. Again it could be CA trust issues, clock skew, or a revoked certificate. The error in the gatekeeper log would typically look like the client-side trust issue above.

1.2.4. Authentication with the Remote Server Failed

Once the GRAM service has authenticated the client, it maps the client's identity to a local user account using a grid-mapfile or other mapping service. If this fails, the client will receive a message that looks like this:

```bash
% globus-job-run grid.example.org /bin/hostname
GRAM Job submission failed because authentication with the remote server failed (error code ?)
```

To remedy this, consult the GRAM administrator to get information from the `/var/log/globus-gatekeeper.log` file to determine the reason why the gatekeeper didn't like your certificate. Again it could be CA trust issues, clock skew, or a revoked certificate. The error in the gatekeeper log would typically look like the client-side trust issue above.
To remedy this, consult the system administrator of the GRAM resource to be added to the authorized user's list. Be sure to send your credential subject name to make it easier for them. To get that information, run the command `grid-cert-info -s`.

### 1.2.5. Unable to Find the Requested Service

Recall that a GRAM resource name includes a component called the service name. The default if not specified is `jobmanager`, but some sites may not use that name, or have a different LRM name than you expect. If you specify an incorrect service name, or the default is not present, you'll get an error that looks like this:

```
% grid-job-run grid.example.org /bin/hostname
GRAM Job submission failed because the gatekeeper failed to find the requested service (error code 93)
```

If you get this error, you'll need to determine which services are available on that GRAM resource, either by asking the admin or by looking at the entries in `/etc/grid-services`.

### 1.2.6. Failed to Run the Job Manager

The GRAM service is split between a privileged process called the `globus-gatekeeper` and a non-privileged process called the `globus-job-manager` which runs as a user process. If the `globus-gatekeeper` is unable to locate the `globus-job-manager` process, then this misconfiguration will show up like this:

```
% grid-job-run grid.example.org /bin/hostname
GRAM Job submission failed because the gatekeeper failed to run the job manager (error code 47)
```

This is an installation mistake, and the administrator of the GRAM resource must fix this.

### 1.3. Jobs are Hanging

One problem GRAM users sometimes encounter is that it looks like jobs submitted to GRAM are not making any progress, even though the local resource manager thinks they've run. There are a couple of reasons why this might occur: GRAM is not getting the information it needs from the local resource manager or the GRAM client is not getting the information it needs. We'll cover diagnosing and handling the latter case in this document, as the other is a system administrator issue.

The way `globus-job-run` and `globusrun` determine that jobs have completed is via GRAM job state callbacks. These are messages sent by the GRAM service to the client node indicating that something significant has happened in the lifecycle of the job. If for some reason the GRAM service can not get those messages to the client, the client will not be able to detect job state changes.

In order to determine if this is the case, submit a job using `globus-job-submit`, and then use the `globus-job-status` command to see if the job state changes. If it does not, then consult the GRAM administrator---there might be some problem with the installation. If it does, then for some reason the callbacks are not happening. This might be firewall issues or host naming issues.

The GRAM client sends a "callback contact" to the GRAM service when it submits a job, in order that it can receive notifications. This contact is a reference to a https server embedded in the GRAM client which only handles GRAM state callbacks. As with all web servers, it has a URL which defines how to contact it, which in this case consists of the client host name and the service port number. If the host name that is used is not resolvable (such as a for a laptop with a dynamic address), then the GRAM service will not be able to contact it. If that's the case, you can set the `GLOBUS_HOSTNAME` environment variable to the IP address that your client can be reached at, and then submit your jobs. This will cause GRAM to publish that address instead of what it thinks the client's host name is.

Another way that the GRAM service would be unable to send job state updates to a client would be if there's a firewall between the service and the client. If that's the case, you might need to set the `GLOBUS_TCP_PORT_RANGE`
environment variable to a comma-separated list of numbers which represent a range of minimum and maximum TCP port numbers to listen on. You might have to contact your site administrator to determine what TCP ports are allowed. If there are none, you can still use `globus-job-submit` and `globus-job-status` to track your job's state changes, or use another tool like those mentioned in the section about client tools.

### 1.4. Logs and Debugging

The GRAM service has a log file which contains information about the job as it is processed. These logs are located by default in `/var/log/globus/gram_$USERNAME.log`. There are some different logging levels available, as described in the GRAM Administrator's Guide. These can be controlled on a per-job basis by adding the `loglevel` RSL attribute to your job description. The default is to log only `FATAL` and `ERROR` messages, but other levels can sometimes help understand what is going on.

### 1.5. Diagnosing LRM Errors

Sometimes, bugs creep into the LRM adapter scripts. When that occurs, the GRAM job will usually fail with an error like this:

```
GRAM Job failed because the job manager detected an invalid script status (error code 25)
```

If this occurs, you may have to work with a GRAM administrator to help debug this problem. One helpful thing you can do when reporting it is to save the GRAM internal script data so that it can be used outside of the GRAM service to see what the low-level error looks like. To do this, add the RSL fragment `savejobdescription = yes` to your job request. This will cause GRAM to leave a file called something like `$HOME/gram_[0-9]*.pl` in your home directory. You can use this with the internal tool `/usr/share/globus/globus-job-manager-script.pl` to try to submit the job to the LRM without using the GRAM service. The command line `/usr/share/globus/globus-job-manager-script.pl -m LRM -c submit -f GRAM-PL-FILE` will attempt to submit the job to the LRM. It will show all the information the LRM script sends to the GRAM service, which might include some perl-language error or badly formatted output from the script (which must only output lines which begin with `GRAM_SCRIPT_`.

In some extreme cases, the `savejobdescription` option will not generate a file. If that's the case, pass `/dev/null` as the argument to the `-f` command-line option. The problem is likely a perl syntax error which will be reached before the job description is loaded.

### 1.6. Email Support

If all else fails, please send information about your problem to `<gram-user@globus.org>`. You'll have to subscribe to a list before you can send an e-mail to it. See [here](http://dev.globus.org/wiki/Mailing_Lists) for general e-mail lists and information on how to subscribe to a list and [here](http://dev.globus.org/wiki/GRAM#Mailing_Lists) for GRAM specific lists. Depending on the problem, you may be requested to file a bug report to the globus project's [Issue Tracker](http://jira.globus.org).
Chapter 4. Command-line Client Reference Pages

This section contains reference pages for all of the tools described in the previous section. These pages contain all the command-line options for these tools. These are available as manpages in the documentation subpackages for the globus-gram-client-tools package.
Name
globusrun — Execute and manage jobs via GRAM

Synopsis


globusrun [-p | -parse] 
   [-f RSL_FILENAME] [-file RSL_FILENAME] | RSL_SPECIFICATION }

globusrun [-n] [-no-interrupt]
   [-r RESOURCE_CONTACT] | -resource RESOURCE_CONTACT }
   [-a | -authenticate-only ]

.globusrun [-n] [-no-interrupt]
   [-r RESOURCE_CONTACT] | -resource RESOURCE_CONTACT }
   [-j | -jobmanager-version ]

.globusrun [-n] [-no-interrupt] { -k | -kill } {JOB_ID}


.globusrun { -status } {JOB_ID}

   [-r RESOURCE_CONTACT] | -resource RESOURCE_CONTACT }
   [-f RSL_FILENAME] | -file RSL_FILENAME] | RSL_SPECIFICATION }

Description

The globusrun program for submits and manages jobs run on a local or remote job host. The jobs are controlled by the globus-job-manager program which interfaces with a local resource manager that schedules and executes the job.

The globusrun program can be run in a number of different modes chosen by command-line options.

When -help, -usage, -version, or -versions command-line options are used, globusrun will print out diagnostic information and then exit.

When the -p or -parse command-line option is present, globusrun will verify the syntax of the RSL specification and then terminate. If the syntax is valid, globusrun will print out the string "RSL Parsed Successfully..." and exit with a zero exit code; otherwise, it will print an error message and terminate with a non-zero exit code.

When the -a or -authenticate-only command-line option is present, globusrun will verify that the service named by RESOURCE_CONTACT exists and the client's credentials are granted permission to access that service. If authentication is successful, globusrun will display the string "GRAM Authentication test successful" and exit with a zero exit code; otherwise it will print an explanation of the problem and exit with a non-zero exit code.

When the -j or -jobmanager-version command-line option is present, globusrun will attempt to determine the software version that the service named by RESOURCE_CONTACT is running. If successful, it will display both the Toolkit version and the Job Manager package version and exit with a zero exit code; otherwise, it will print an explanation of the problem and exit with a non-zero exit code.
When the \texttt{-k} or \texttt{-kill} command-line option is present, \texttt{globusrun} will attempt to terminate the job named by \texttt{JOB\_ID}. If successful, \texttt{globusrun} will exit with zero; otherwise it will display an explanation of the problem and exit with a non-zero exit code.

When the \texttt{-y} or \texttt{-refresh-proxy} command-line option is present, \texttt{globusrun} will attempt to delegate a new X.509 proxy to the job manager which is managing the job named by \texttt{JOB\_ID}. If successful, \texttt{globusrun} will exit with zero; otherwise it will display an explanation of the problem and exit with a non-zero exit code. This behavior can be modified by the \texttt{-full-proxy} or \texttt{-D} command-line options to enable full proxy delegation. The default is limited proxy delegation.

When the \texttt{-status} command-line option is present, \texttt{globusrun} will attempt to determine the current state of the job. If successful, the state will be printed to standard output and \texttt{globusrun} will exit with a zero exit code; otherwise, a description of the error will be displayed and it will exit with a non-zero exit code.

Otherwise, \texttt{globusrun} will submit the job to a GRAM service. By default, \texttt{globusrun} waits until the job has terminated or failed before exiting, displaying information about job state changes and at exit time, the job exit code if it is provided by the GRAM service.

The \texttt{globusrun} program can also function as a GASS file server to allow the \texttt{globus-job-manager} program to stage files to and from the machine on which \texttt{globusrun} is executed to the GRAM service node. This behavior is controlled by the \texttt{-s}, \texttt{-o}, and \texttt{-w} command-line options.

Jobs submitted by \texttt{globusrun} can be monitored interactively or detached. To have \texttt{globusrun} detach from the GRAM service after submitting the job, use the \texttt{-b} or \texttt{-F} command-line options.

## Options

The full set of options to \texttt{globusrun} consist of:

\begin{itemize}
  \item \texttt{-help} \hspace{1cm} Display a help message to standard error and exit.
  \item \texttt{-usage} \hspace{1cm} Display a one-line usage summary to standard error and exit.
  \item \texttt{-version} \hspace{1cm} Display the software version of \texttt{globusrun} to standard error and exit.
  \item \texttt{-versions} \hspace{1cm} Display the software version of all modules used by \texttt{globusrun} (including DiRT information) to standard error and then exit.
  \item \texttt{-p, -parse} \hspace{1cm} Do a parse check on the job specification and print diagnostics. If a parse error occurs, \texttt{globusrun} exits with a non-zero exit code.
  \item \texttt{-f RSL\_FILENAME, -file RSL\_FILENAME} \hspace{1cm} Read job specification from the file named by \texttt{RSL\_FILENAME}.
  \item \texttt{-n, -no-interrupt} \hspace{1cm} Disable handling of the \texttt{SIGINT} signal, so that the interrupt character (typically \texttt{Control+C}) causes \texttt{globusrun} to terminate without canceling the job.
  \item \texttt{-r RESOURCE\_CONTACT, -resource RESOURCE\_CONTACT} \hspace{1cm} Submit the request to the resource specified by \texttt{RESOURCE\_CONTACT}. A resource may be specified in the following ways:
    \begin{itemize}
      \item \texttt{HOST}
      \item \texttt{HOST:PORT}
      \item \texttt{HOST:PORT\_SERVICE}
    \end{itemize}
\end{itemize}
• \texttt{HOST/SERVICE}
• \texttt{HOST/SERVICE}
• \texttt{HOST:SUBJECT}
• \texttt{HOST:PORT:SUBJECT}
• \texttt{HOST/_SERVICE:SUBJECT}
• \texttt{HOST/ Service:SUBJECT}
• \texttt{HOST:PORT:SERVICE:SUBJECT}

If any of \texttt{PORT}, \texttt{SERVICE}, or \texttt{SUBJECT} is omitted, the defaults of 2811, \texttt{jobmanager}, and \texttt{host@HOST} are used respectively.

\texttt{-j, -jobmanager-version} \hspace{1em} Print the software version being run by the service running at \texttt{RESOURCE_CONTACT}.

\texttt{-k \texttt{JOB_ID}, -kill \texttt{JOB_ID}} \hspace{1em} Kill the job named by \texttt{JOB_ID}

\texttt{-D, -full-proxy} \hspace{1em} Delegate a full impersonation proxy to the service. By default, a limited proxy is delegated when needed.

\texttt{-y, -refresh-proxy} \hspace{1em} Delegate a new proxy to the service processing \texttt{JOB_ID}.

\texttt{-status} \hspace{1em} Display the current status of the job named by \texttt{JOB_ID}.

\texttt{-q, -quiet} \hspace{1em} Do not display job state change or exit code information.

\texttt{-o, -output-enable} \hspace{1em} Start a GASS server within the \texttt{globusrun} application that allows access to its standard output and standard error streams only. Also, augment the \texttt{RSL\_SPECIFICATION} with a definition of the \texttt{GLOBUSRUN\_GASS\_URL} RSL substitution and add \texttt{stdout} and \texttt{stderr} clauses which redirect the output and error streams of the job to the output and error streams of the interactive \texttt{globusrun} command. If this is specified, then \texttt{globusrun} acts as though the \texttt{-q} were also specified.

\texttt{-s, -server} \hspace{1em} Start a GASS server within the \texttt{globusrun} application that allows access to its standard output and standard error streams for writing and any file local to the \texttt{globusrun} invocation for reading. Also, augment the \texttt{RSL\_SPECIFICATION} with a definition of the \texttt{GLOBUSRUN\_GASS\_URL} RSL substitution and add \texttt{stdout} and \texttt{stderr} clauses which redirect the output and error streams of the job to the output and error streams of the interactive \texttt{globusrun} command. If this is specified, then \texttt{globusrun} acts as though the \texttt{-q} were also specified.

\texttt{-w, -write-allow} \hspace{1em} Start a GASS server within the \texttt{globusrun} application that allows access to its standard output and standard error streams for writing and any file local to the \texttt{globusrun} invocation for reading or writing. Also, augment the \texttt{RSL\_SPECIFICATION} with a definition of the \texttt{GLOBUSRUN\_GASS\_URL} RSL substitution and add \texttt{stdout} and \texttt{stderr} clauses which redirect the output and error streams of the job to the output and error streams of the interactive \texttt{globusrun} command. If this is specified, then \texttt{globusrun} acts as though the \texttt{-q} were also specified.
-b, -batch

Terminate after submitting the job to the GRAM service. The **globusrun** program will exit after the job hits any of the following states: PENDING, ACTIVE, FAILED, or DONE. The GASS-related options can be used to stage input files, but standard output, standard error, and file staging after the job completes will not be processed.

-F, -fast-batch

Terminate after submitting the job to the GRAM service. The **globusrun** program will exit after it receives a reply from the service. The **JOB_ID** will be displayed to standard output before terminating so that the job can be checked with the -status command-line option or modified by the -refresh-proxy or -kill command-line options.

-d, -dryrun

Submit the job with the dryrun attribute set to true. When this is done, the job manager will prepare to start the job but start short of submitting it to the service. This can be used to detect problems with the **RSL_SPECIFICATION**.

### Environment

If the following variables affect the execution of **globusrun**

- **X509_USER_PROXY** Path to proxy credential.
- **X509_CERT_DIR** Path to trusted certificate directory.

### Bugs

The **globusrun** program assumes any failure to contact the job means the job has terminated. In fact, this may be due to the **globus-job-manager** program exiting after all jobs it is managing have reached the DONE or FAILED states. In order to reliably detect job termination, the two_phase RSL attribute should be used.

### See Also

globus-job-submit(1), globus-job-run(1), globus-job-clean(1), globus-job-get-output(1), globus-job-cancel(1)
Name
globus-job-cancel — Cancel a GRAM batch job

Synopsis
globus-job-cancel [-f | -force] [-q | -quiet] JOBID

Description
The globus-job-cancel program cancels the job named by JOBID. Any cached files associated with the job will remain until globus-job-clean is executed for the job.

By default, globus-job-cancel prompts the user prior to canceling the job. This behavior can be overridden by specifying the -f or -force command-line options.

Options
The full set of options to globus-job-cancel are:

- -help, -usage
  Display a help message to standard error and exit.

- -version
  Display the software version of the globus-job-cancel program to standard output.

- -version
  Display the software version of the globus-job-cancel program including DiRT information to standard output.

- -force, -f
  Do not prompt to confirm job cancel and clean-up.

- -quiet, -q
  Do not print diagnostics for succesful cancel. Implies -f

ENVIRONMENT
If the following variables affect the execution of globus-job-cancel.

X509_USER_PROXY Path to proxy credential.
X509_CERT_DIR Path to trusted certificate directory.
Name

globus-job-clean — Cancel and clean up a GRAM batch job

Synopsis

globus-job-clean [-r RESOURCE | -resource RESOURCE]
[-f | -force] [-q | -quiet] JOBID


Description

The globus-job-clean program cancels the job named by JOBID if it is still running, and then removes any cached files on the GRAM service node related to that job. In order to do the file clean up, it submits a job which removes the cache files. By default this cleanup job is submitted to the default GRAM resource running on the same host as the job. This behavior can be controlled by specifying a resource manager contact string as the parameter to the -r or -resource option.

By default, globus-job-clean prompts the user prior to canceling the job. This behavior can be overridden by specifying the -f or -force command-line options.

Options

The full set of options to globus-job-clean are:

- help, -usage Display a help message to standard error and exit.
- version Display the software version of the globus-job-clean program to standard output.
- version Display the software version of the globus-job-clean program including DiRT information to standard output.
- resource RESOURCE, -r RESOURCE Submit the clean-up job to the resource named by RESOURCE instead of the default GRAM service on the same host as the job contact.
- force, -f Do not prompt to confirm job cancel and clean-up.
- quiet, -q Do not print diagnostics for succesful clean-up. Implies -f

ENVIRONMENT

If the following variables affect the execution of globus-job-clean.

X509_USER_PROXY Path to proxy credential.
X509_CERT_DIR Path to trusted certificate directory.
**Name**
globus-job-get-output — Retrieve the output and error streams from a GRAM job

**Synopsis**


**Description**
The `globus-job-get-output` program retrieves the output and error streams of the job named by `JOBID`. By default, `globus-job-get-output` will retrieve all output and error data from the job and display them to its own output and error streams. Other behavior can be controlled by using command-line options. The data retrieval is implemented by submitting another job which simply displays the contents of the first job's output and error streams. By default this retrieval job is submitted to the default GRAM resource running on the same host as the job. This behavior can be controlled by specifying a particular resource manager contact string as the `RESOURCE` parameter to the `-r` or `-resource` option.

**Options**
The full set of options to `globus-job-get-output` are:

- **-help, -usage**
  Display a help message to standard error and exit.

- **-version**
  Display the software version of the `globus-job-get-output` program to standard output.

- **-version**
  Display the software version of the `globus-job-get-output` program including DiRT information to standard output.

- **-resource RESOURCE, -r RESOURCE**
  Submit the retrieval job to the resource named by `RESOURCE` instead of the default GRAM service on the same host as the job contact.

- **-out**
  Retrieve only the standard output stream of the job. The default is to retrieve both standard output and standard error.

- **-err**
  Retrieve only the standard error stream of the job. The default is to retrieve both standard output and standard error.

- **-tail LINES, -t LINES**
  Print only the last `LINES` count lines of output from the data streams being retrieved. By default, the entire output and error file data is retrieved. This option can not be used along with the `-f` or `-follow` options.

- **-follow LINES, -f LINES**
  Print the last `LINES` count lines of output from the data streams being retrieved and then wait until canceled, printing any subsequent job output that occurs. By default, the entire output and error file data is retrieved. This option can not be used along with the `-t` or `-tail` options.

**ENVIRONMENT**

If the following variables affect the execution of `globus-job-get-output`. 
X509_USER_PROXY  Path to proxy credential.
X509_CERT_DIR    Path to trusted certificate directory.
Name
globus-job-run — Execute a job using GRAM

Synopsis

globus-job-run [-dumprsl] [-dryrun] [-verify]
[-file ARGUMENT_FILE]
SERVICE_CONTACT
[-np PROCESSES] [-count PROCESSES]
[-m MAX_TIME] [-maxtime MAX_TIME]
[-p PROJECT] [-project PROJECT]
[-q QUEUE] [-queue QUEUE]
[-d DIRECTORY] [-directory DIRECTORY] [-env NAME=VALUE]...
[-x RSL_CLAUSE]
[-l | -s] EXECUTABLE [ARGUMENT...]


Description

The globus-job-run program constructs a job description from its command-line options and then submits the job to the GRAM service running at SERVICE_CONTACT. The executable and arguments to the executable are provided on the command-line after all other options. Note that the -dumprsl, -dryrun, -verify, and -file command-line options must occur before the first non-option argument, the SERVICE_CONTACT.

The globus-job-run provides similar functionality to globusrun in that it allows interactive start-up of GRAM jobs. However, unlike globusrun, it uses command-line parameters to define the job instead of RSL expressions.

Options

The full set of options to globus-job-run are:

- **-help, -usage** Display a help message to standard error and exit.

- **-version** Display the software version of the globus-job-run program to standard output.

- **-version** Display the software version of the globus-job-run program including DiRT information to standard output.

- **-dumprsl** Translate the command-line options to globus-job-run into an RSL expression that can be used with tools such as globusrun.

- **-dryrun** Submit the job request to the GRAM service with the dryrun option enabled. When this option is used, the GRAM service prepares to execute the job but stops before submitting the job to the LRM. This can be used to diagnose some problems such as missing files.

- **-verify** Submit the job request to the GRAM service with the dryrun option enabled and then without it enabled if the dryrun is successful.

- **-file ARGUMENT_FILE** Read additional command-line options from ARGUMENT_FILE.
-np PROCESSES, -count PROCESSES  
Start PROCESSES instances of the executable as a single job.

-m MAX_TIME, -maxtime MAX_TIME  
Schedule the job to run for a maximum of MAX_TIME minutes.

-p PROJECT, -project PROJECT  
Request that the job use the allocation PROJECT when submitting the job to the LRM.

-q QUEUE, -queue QUEUE  
Request that the job be submitted to the LRM using the named QUEUE.

-d DIRECTORY, -directory DIRECTORY  
Run the job in the directory named by DIRECTORY. Input and output files will be interpreted relative to this directory. This directory must exist on the file system on the LRM-managed resource. If not specified, the job will run in the home directory of the user the job is running as.

-env NAME=VALUE  
Define an environment variable named by NAME with the value VALUE in the job environment. This option may be specified multiple times to define multiple environment variables.

-stdin [-l | -s] STDIN_FILE  
Use the file named by STDIN_FILE as the standard input of the job. If the -l option is specified, then this file is interpreted to be on a file system local to the LRM. If the -s option is specified, then this file is interpreted to be on the file system where globus-job-run is being executed, and the file will be staged via GASS. If neither is specified, the local behavior is assumed.

-stdout [-l | -s] STDOUT_FILE  
Use the file named by STDOUT_FILE as the destination for the standard output of the job. If the -l option is specified, then this file is interpreted to be on a file system local to the LRM. If the -s option is specified, then this file is interpreted to be on the file system where globus-job-run is being executed, and the file will be staged via GASS. If neither is specified, the local behavior is assumed.

-stderr [-l | -s] STDERR_FILE  
Use the file named by STDERR_FILE as the destination for the standard error of the job. If the -l option is specified, then this file is interpreted to be on a file system local to the LRM. If the -s option is specified, then this file is interpreted to be on the file system where globus-job-run is being executed, and the file will be staged via GASS. If neither is specified, the local behavior is assumed.

-x RSL_CLAUSE  
Add a set of custom RSL attributes described by RSL_CLAUSE to the job description. The clause must be an RSL conjunction and may contain one or more attributes. This can be used to include attributes which can not be defined by other command-line options of globus-job-run.

-1  
When included outside the context of -stdin, -stdout, or -stderr command-line options, -1 option alters the interpretation of the executable path. If the -1 option is specified, then the executable is interpreted to be on a file system local to the LRM.

-s  
When included outside the context of -stdin, -stdout, or -stderr command-line options, -s option alters the interpretation of the executable path. If the -s option is specified, then the executable is interpreted to be on the file system where globus-job-run is being executed, and the file will be staged via GASS. If neither is specified, the local behavior is assumed.
ENVIRONMENT

If the following variables affect the execution of `globus-job-run`.

- `X509_USER_PROXY`  Path to proxy credential.
- `X509_CERT_DIR`  Path to trusted certificate directory.

See Also

globusrun(1), globus-job-submit(1), globus-job-clean(1), globus-job-get-output(1), globus-job-cancel(1)
Name
globus-job-status — Check the status of a GRAM5 job

Synopsis
globus-job-status JOBID

Description
The globus-job-status program checks the status of a GRAM job by sending a status request to the job manager
contact for that job specified by the JOBID parameter. If successful, it will print the job status to standard output.
The states supported by globus-job-status are:

PENDING       The job has been submitted to the LRM but has not yet begun execution.
ACTIVE        The job has begun execution.
FAILED        The job has failed.
SUSPENDED     The job is currently suspended by the LRM.
DONE          The job has completed.
UNSUBMITTED   The job has been accepted by GRAM, but not yet submitted to the LRM.
STAGE_IN      The job has been accepted by GRAM and is currently staging files prior to being submitted to the
               LRM.
STAGE_OUT     The job has completed execution and is currently staging files from the service node to other http,
               GASS, or GridFTP servers.

Options
The full set of options to globus-job-status are:

-help, --help       Display a help message to standard error and exit.
-usage              Display a help message to standard error and exit.
-version            Display the software version of the globus-job-status program to standard output.
-versions            Display the software version of the globus-job-status program including DiRT information to stan-
                      dard output.

ENVIRONMENT
If the following variables affect the execution of globus-job-status.

X509_USER_PROXY  Path to proxy credential.
X509_CERT_DIR    Path to trusted certificate directory.
Bugs

The `globus-job-status` program cannot distinguish between the case of the job manager terminating for any reason and the job being in the `DONE` state.

See Also

globusrun(1)
Name
globus-job-submit — Submit a batch job using GRAM

Synopsis
globus-job-submit [-dumpsl] [-dryrun] [-verify]
[file ARGUMENT_FILE]
SERVICE_CONTACT
[ -np PROCESSES | -count PROCESSES ]
[ -m MAX_TIME | -maxtime MAX_TIME ]
[ -p PROJECT | -project PROJECT ]
[ -q QUEUE | -queue QUEUE ]
[ -d DIRECTORY | -directory DIRECTORY ] [ -env NAME=VALUE ]...
[ -x RSL_CLAUSE ]
[ -l | -s ] EXECUTABLE[ARGUMENT...]

Description
The globus-job-submit program constructs a job description from its command-line options and then submits the job to the GRAM service running at SERVICE_CONTACT. The executable and arguments to the executable are provided on the command-line after all other options. Note that the -dumpsl, -dryrun, -verify, and -file command-line options must occur before the first non-option argument, the SERVICE_CONTACT.

The globus-job-submit provides similar functionality to globusrun in that it allows batch submission of GRAM jobs. However, unlike globusrun, it uses command-line parameters to define the job instead of RSL expressions.

To retrieve the output and error streams of the job, use the program globus-job-get-output. To reclaim resources used by the job by deleting cached files and job state, use the program globus-job-clean. To cancel a batch job submitted by globus-job-submit, use the program globus-job-cancel.

Options
The full set of options to globus-job-submit are:

- help, -usage
  Display a help message to standard error and exit.

- version
  Display the software version of the globus-job-submit program to standard output.

- versions
  Display the software version of the globus-job-submit program including DiRT information to standard output.

- dumpsl
  Translate the command-line options to globus-job-submit into an RSL expression that can be used with tools such as globusrun.

- dryrun
  Submit the job request to the GRAM service with the dryrun option enabled. When this option is used, the GRAM service prepares to execute the job but stops before submitting the job to the LRM. This can be used to diagnose some problems such as missing files.

- verify
  Submit the job request to the GRAM service with the dryrun option enabled and then without it enabled if the dryrun is successful.
-file ARGUMENT_FILE  Read additional command-line options from ARGUMENT_FILE.

-np PROCESSES, -count PROCESSES  Start PROCESSES instances of the executable as a single job.

-m MAX_TIME, -maxtime MAX_TIME  Schedule the job to run for a maximum of MAX_TIME minutes.

-p PROJECT, -project PROJECT  Request that the job use the allocation PROJECT when submitting the job to the LRM.

-q QUEUE, -queue QUEUE  Request that the job be submitted to the LRM using the named QUEUE.

-d DIRECTORY, -directory DIRECTORY  Run the job in the directory named by DIRECTORY. Input and output files will be interpreted relative to this directory. This directory must exist on the file system on the LRM-managed resource. If not specified, the job will run in the home directory of the user the job is running as.

-env NAME=VALUE  Define an environment variable named by NAME with the value VALUE in the job environment. This option may be specified multiple times to define multiple environment variables.

-stdin [-l | -s] STDIN_FILE  Use the file named by STDIN_FILE as the standard input of the job. If the -l option is specified, then this file is interpreted to be on a file system local to the LRM. If the -s option is specified, then this file is interpreted to be on the file system where globus-job-submit is being executed, and the file will be staged via GASS. If neither is specified, the local behavior is assumed.

-stdout [-l | -s] STDOUT_FILE  Use the file named by STDOUT_FILE as the destination for the standard output of the job. If the -l option is specified, then this file is interpreted to be on a file system local to the LRM. If the -s option is specified, then this file is interpreted to be on the file system where globus-job-submit is being executed, and the file will be staged via GASS. If neither is specified, the local behavior is assumed.

-stderr [-l | -s] STDERR_FILE  Use the file named by STDERR_FILE as the destination for the standard error of the job. If the -l option is specified, then this file is interpreted to be on a file system local to the LRM. If the -s option is specified, then this file is interpreted to be on the file system where globus-job-submit is being executed, and the file will be staged via GASS. If neither is specified, the local behavior is assumed.

-x RSL_CLAUSE  Add a set of custom RSL attributes described by RSL_CLAUSE to the job description. The clause must be an RSL conjunction and may contain one or more attributes. This can be used to include attributes which can not be defined by other command-line options of globus-job-submit.

-l  When included outside the context of -stdin, -stdout, or -stderr command-line options, -l option alters the interpretation of the executable path. If the -l option is specified, then the executable is interpreted to be on a file system local to the LRM.

-s  When included outside the context of -stdin, -stdout, or -stderr command-line options, -s option alters the interpretation of the executable path. If the -s option is specified, then the executable is interpreted to be on
the file system where `globus-job-run` is being executed, and the file will be staged via GASS. If neither is specified, the local behavior is assumed.

**ENVIRONMENT**

If the following variables affect the execution of `globus-job-submit`.

- `X509_USER_PROXY` Path to proxy credential.
- `X509_CERT_DIR` Path to trusted certificate directory.

**See Also**

globusrun(1), globus-job-run(1), globus-job-clean(1), globus-job-get-output(1), globus-job-cancel(1)
Chapter 5. GRAM RSL Quick Reference

The GRAM RSL language is described in detail in the GRAM Developer's Guide. For basic use, job description RSLs consist of a set of RSL attributes preceded by the & character. The basic job description looks like:

```
&
  (attribute = value )
  (attribute = value )
  ...
```

The following list contains the RSL attributes which are available in the core job manager. Other LRM-specific RSL attributes may also be available in some situations.

- **arguments**: The command line arguments for the executable. Use quotes, if a space is required in a single argument.
- **count**: The number of executions of the executable.
- **directory**: Specifies the path of the directory the jobmanager will use as the default directory for the requested job.
- **dry_run**: If dryrun = yes then the jobmanager will not submit the job for execution and will return success.
- **environment**: The environment variables that will be defined for the executable in addition to default set that is given to the job by the jobmanager.
- **executable**: The name of the executable file to run on the remote machine. If the value is a GASS URL, the file is transferred to the remote gass cache before executing the job and removed after the job has terminated.
- **file_clean_up**: Specifies a list of files which will be removed after the job is completed.
- **file_stage_in**: Specifies a list of ("remote URL" "local file") pairs which indicate files to be staged to the nodes which will run the job.
- **file_stage_in_shared**: Specifies a list of ("remote URL" "local file") pairs which indicate files to be staged into the cache. A symlink from the cache to the "local file" path will be made.
- **file_stage_out**: Specifies a list of ("local file" "remote URL") pairs which indicate files to be staged from the job to a GASS-compatible file server.
- **gass_cache**: Specifies location to override the GASS cache location.
- **gram_my_job**: Obsolete and ignored.
- **host_count**: Only applies to clusters of SMP computers, such as newer IBM SP systems. Defines the number of nodes ("pizza boxes") to distribute the "count" processes across.
- **job_type**: This specifies how the jobmanager should start the job. Possible values are single (even if the count > 1, only start 1 process or thread), multiple (start count processes or threads), mpi (use the appropriate method (e.g. mpirun) to start a program compiled with a vendor-provided MPI library. Program is started with count nodes), and condor (starts condor jobs in the "condor" universe.)
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>library_path</td>
<td>Specifies a list of paths to be appended to the system-specific library path environment variables.</td>
</tr>
<tr>
<td>loglevel</td>
<td>Override the default log level for this job. The value of this attribute consists of a combination of the strings FATAL, ERROR, WARN, INFO, DEBUG, TRACE joined by the</td>
</tr>
<tr>
<td>logpattern</td>
<td>Override the default log path pattern for this job. The value of this attribute is a string (potentially containing RSL substitutions) that is evaluated to the path to write the log to. If the resulting string contains the string $(DATE) (or any other RSL substitution), it will be reevaluated at log time.</td>
</tr>
<tr>
<td>max_cpu_time</td>
<td>Explicitly set the maximum cpumtime for a single execution of the executable. The units is in minutes. The value will go through an atoi() conversion in order to get an integer. If the GRAM scheduler cannot set cpumtime, then an error will be returned.</td>
</tr>
<tr>
<td>max_memory</td>
<td>Explicitly set the maximum amount of memory for a single execution of the executable. The units is in Megabytes. The value will go through an atoi() conversion in order to get an integer. If the GRAM scheduler cannot set maxMemory, then an error will be returned.</td>
</tr>
<tr>
<td>max_time</td>
<td>The maximum walltime or cpumtime for a single execution of the executable. Walltime or cpumtime is selected by the GRAM scheduler being interfaced. The units is in minutes. The value will go through an atoi() conversion in order to get an integer.</td>
</tr>
<tr>
<td>max_wall_time</td>
<td>Explicitly set the maximum walltime for a single execution of the executable. The units is in minutes. The value will go through an atoi() conversion in order to get an integer. If the GRAM scheduler cannot set walltime, then an error will be returned.</td>
</tr>
<tr>
<td>min_memory</td>
<td>Explicitly set the minimum amount of memory for a single execution of the executable. The units is in Megabytes. The value will go through an atoi() conversion in order to get an integer. If the GRAM scheduler cannot set minMemory, then an error will be returned.</td>
</tr>
<tr>
<td>project</td>
<td>Target the job to be allocated to a project account as defined by the scheduler at the defined (remote) resource.</td>
</tr>
<tr>
<td>proxy_timeout</td>
<td>Obsolete and ignored. Now a job-manager-wide setting.</td>
</tr>
<tr>
<td>queue</td>
<td>Target the job to a queue (class) name as defined by the scheduler at the defined (remote) resource.</td>
</tr>
<tr>
<td>remote_io_url</td>
<td>Writes the given value (a URL base string) to a file, and adds the path to that file to the environment through the GLOBUS_REMOTE_IO_URL environment variable. If this is specified as part of a job restart RSL, the job manager will update the file's contents. This is intended for jobs that want to access files via GASS, but the URL of the GASS server has changed due to a GASS server restart.</td>
</tr>
<tr>
<td>restart</td>
<td>Start a new job manager, but instead of submitting a new job, start managing an existing job. The job manager will search for the job state file created by the original job manager. If it finds the file and successfully reads it, it will become the new manager of the job, sending callbacks on status and streaming stdout/err if appropriate. It will fail if it detects that the old jobmanager is still alive (via a timestamp in the state file). If stdout or stderr was being streamed over the network, new stdout and stderr attributes can be specified in the restart RSL and the jobmanager will stream to the new locations (useful when output is going to a GASS server started</td>
</tr>
</tbody>
</table>
by the client that's listening on a dynamic port, and the client was restarted). The new job manager will return a new contact string that should be used to communicate with it. If a jobmanager is restarted multiple times, any of the previous contact strings can be given for the restart attribute.

**rsl_substitution**

Specifies a list of values which can be substituted into other rsl attributes' values through the $(SUBSTITUTION) mechanism.

**save_state**

Causes the jobmanager to save it's job state information to a persistent file on disk. If the job manager exits or is suspended, the client can later start up a new job manager which can continue monitoring the job.

**save_jobdescription**

Save a copy of the job description to $HOME

**scratch_dir**

Specifies the location to create a scratch subdirectory in. A SCRATCH_DIRECTORY RSL substitution will be filled with the name of the directory which is created.

**stderr**

The name of the remote file to store the standard error from the job. If the value is a GASS URL, the standard error from the job is transferred dynamically during the execution of the job. There are two accepted forms of this value. It can consist of a single destination: stderr = URL, or a sequence of destinations: stderr = (DESTINATION) (DESTINATION). In the latter case, the DESTINATION may itself be a URL or a sequence of an x-gass-cache URL followed by a cache tag.

**stderr_position**

Specifies where in the file remote standard error streaming should be restarted from. Must be 0.

**stdin**

The name of the file to be used as standard input for the executable on the remote machine. If the value is a GASS URL, the file is transferred to the remote gass cache before executing the job and removed after the job has terminated.

**stdout**

The name of the remote file to store the standard output from the job. If the value is a GASS URL, the standard output from the job is transferred dynamically during the execution of the job. There are two accepted forms of this value. It can consist of a single destination: stdout = URL, or a sequence of destinations: stdout = (DESTINATION) (DESTINATION). In the latter case, the DESTINATION may itself be a URL or a sequence of an x-gass-cache URL followed by a cache tag.

**stdout_position**

Specifies where in the file remote output streaming should be restarted from. Must be 0.

**two_phase**

Use a two-phase commit for job submission and completion. The job manager will respond to the initial job request with a WAITING_FOR_COMMIT error. It will then wait for a signal from the client before doing the actual job submission. The integer supplied is the number of seconds the job manager should wait before timing out. If the job manager times out before receiving the commit signal, or if a client issues a cancel signal, the job manager will clean up the job's files and exit, sending a callback with the job status as GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED. After the job manager sends a DONE or FAILED callback, it will wait for a commit signal from the client. If it receives one, it cleans up and exits as usual. If it times out and save_state was enabled, it will leave all of the job's files in place and exit (assuming the client is down and will attempt a job restart later). The timeout value can be extended via a signal. When one of the following errors occurs, the job manager does not delete the job state file when it
exits: GLOBUS_GRAM_PROTOCOL_ERROR_COMMIT_TIMED_OUT, GLOBUS_GRAM_PROTOCOL_ERROR_TTL_EXPIRED, GLOBUS_GRAM_PROTOCOL_ERROR_JM_STOPPED, GLOBUS_GRAM_PROTOCOL_ERROR_USER_PROXY_EXPIRED. In these cases, it can not be restarted, so the job manager will not wait for the commit signal after sending the FAILED callback

username Verify that the job is running as this user.
## Chapter 6. GRAM Error Message Reference

### 1. Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Reason</th>
<th>Possible Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>one of the RSL parameters is not supported</td>
<td>Check RSL documentation</td>
</tr>
<tr>
<td>2</td>
<td>the RSL length is greater than the maximum allowed</td>
<td>Use RSL substitutions to reduce length of RSL strings</td>
</tr>
<tr>
<td>3</td>
<td>an I/O operation failed</td>
<td>Enable trace logging and report to <a href="mailto:gram-dev@globus.org">gram-dev@globus.org</a></td>
</tr>
<tr>
<td>4</td>
<td>jobmanager unable to set default to the directory requested</td>
<td>Check that RSL directory attribute refers to a directory that exists on the target system.</td>
</tr>
<tr>
<td>5</td>
<td>the executable does not exist</td>
<td>Check that the RSL executable attribute refers to an executable that exists on the target system.</td>
</tr>
<tr>
<td>6</td>
<td>of an unused INSUFFICIENT_FUNDS</td>
<td>Unimplemented feature.</td>
</tr>
<tr>
<td>7</td>
<td>authentication with the remote server failed</td>
<td>Check that the contact string contains the proper X.509 DN.</td>
</tr>
<tr>
<td>8</td>
<td>the user cancelled the job</td>
<td>Don't cancel jobs you want to complete.</td>
</tr>
<tr>
<td>9</td>
<td>the system cancelled the job</td>
<td>Check RSL requirements such as maximum time and memory are valid for the job.</td>
</tr>
<tr>
<td>10</td>
<td>data transfer to the server failed</td>
<td>Check gatekeeper and/or job manager logs to see why the process failed.</td>
</tr>
<tr>
<td>11</td>
<td>the stdin file does not exist</td>
<td>Check that the RSL stdin attribute refers to a file that exists on the target system or has a valid ftp, gsiftp, http, or https URL.</td>
</tr>
<tr>
<td>12</td>
<td>the connection to the server failed (check host and port)</td>
<td>Check that the service is running on the expected TCP/IP port. Check that no firewall prevents contacting that TCP/IP port. Check $GLOBUS_LOCATION/var/globus-gatekeeper.log for runtime configuration errors.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td>13</td>
<td>the provided RSL 'maxtime' value is not an integer</td>
<td>Check that the RSL maxtime value evaluates to an integer.</td>
</tr>
<tr>
<td>14</td>
<td>the provided RSL 'count' value is not an integer</td>
<td>Check that the RSL count value evaluates to an integer.</td>
</tr>
<tr>
<td>15</td>
<td>the job manager received an invalid RSL</td>
<td>Check that the RSL string can be parsed by using <code>globusrun -p RSL</code>.</td>
</tr>
<tr>
<td>16</td>
<td>the job manager failed in allowing others to make contact</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>17</td>
<td>the job failed when the job manager attempted to run it</td>
<td>Verify that the LRM is configured properly.</td>
</tr>
<tr>
<td>18</td>
<td>an invalid paradyn was specified</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>19</td>
<td>the provided RSL 'jobtype' value is invalid</td>
<td>The RSL jobtype attribute is not indicated as supported by the LRM. Valid jobtype values are single, multiple, mpi, and condor.</td>
</tr>
<tr>
<td>20</td>
<td>the provided RSL 'myjob' value is invalid</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>21</td>
<td>the job manager failed to locate an internal script argument file</td>
<td>Check that $GLOBUS_LOCATION/libexec/globus-job-manager-script.pl exists and is executable. Check that the LRM-specific perl module is located in $GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/ directory and is valid. The command <code>perl -I $GLOBUS_LOCATION/lib/perl $GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/LRM.pm</code> can be used to check if there are any syntax errors in the script.</td>
</tr>
<tr>
<td>22</td>
<td>the job manager failed to create an internal script argument file</td>
<td>Check that your home directory is writable and not full.</td>
</tr>
<tr>
<td>23</td>
<td>the job manager detected an invalid job state</td>
<td>Check job manager logs.</td>
</tr>
<tr>
<td>24</td>
<td>the job manager detected an invalid script response</td>
<td>Check job manager logs. This is likely a bug in the LRM script.</td>
</tr>
<tr>
<td>25</td>
<td>the job manager detected an invalid script status</td>
<td>Check job manager logs. This is likely a bug in the LRM script.</td>
</tr>
<tr>
<td>26</td>
<td>the provided RSL 'jobtype' value is not supported by this job manager</td>
<td>Check that the RSL jobtype attribute is implemented by the LRM script. Note that some job types require configuration</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>27</td>
<td>unused ERROR_UNIMPLEMENTED</td>
<td>LRM does not support some feature included in the job request.</td>
</tr>
<tr>
<td>28</td>
<td>the job manager failed to create an internal script submission file</td>
<td>Check that the user's home file system is not full. Check job manager log</td>
</tr>
<tr>
<td>29</td>
<td>the job manager cannot find the user proxy</td>
<td>Check that client is delegating a proxy when authenticating with the gatekeeper.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check that the user's home filesystem and the /tmp file system are not full.</td>
</tr>
<tr>
<td>30</td>
<td>the job manager failed to open the user proxy</td>
<td>Check that the user's home filesystem and the /tmp file system are not full.</td>
</tr>
<tr>
<td>31</td>
<td>the job manager failed to cancel the job as requested</td>
<td>Check that the user's home filesystem and the /tmp file system are not full.</td>
</tr>
<tr>
<td>32</td>
<td>system memory allocation failed</td>
<td>Check job manager log for details.</td>
</tr>
<tr>
<td>33</td>
<td>the interprocess job communication initialization failed</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>34</td>
<td>the interprocess job communication setup failed</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>35</td>
<td>the provided RSL 'host count' value is invalid</td>
<td>Check that the RSL host_count attribute evaluates to an integer.</td>
</tr>
<tr>
<td>36</td>
<td>one of the provided RSL parameters is unsupported</td>
<td>Check job manager log for details about invalid parameter.</td>
</tr>
<tr>
<td>37</td>
<td>the provided RSL 'queue' parameter is invalid</td>
<td>Check that the RSL queue attribute evaluates to a string that corresponds to an LRM-specific queue name.</td>
</tr>
<tr>
<td>38</td>
<td>the provided RSL 'project' parameter is invalid</td>
<td>Check that the RSL project attribute evaluates to a string that corresponds to an LRM-specific project name.</td>
</tr>
<tr>
<td>39</td>
<td>the provided RSL string includes variables that could not be identifed</td>
<td>Check that all RSL substitutions are defined before being used in the job description.</td>
</tr>
<tr>
<td>40</td>
<td>the provided RSL 'environment' parameter is invalid</td>
<td>Check that the RSL environment attribute contains a sequence of VARIABLE VALUE pairs.</td>
</tr>
<tr>
<td>41</td>
<td>the provided RSL 'dryrun' parameter is invalid</td>
<td>Remove the RSL dryrun attribute from the job description.</td>
</tr>
<tr>
<td>42</td>
<td>the provided RSL is invalid (an empty string)</td>
<td>Include a non-empty RSL string in your job submission request.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td>43</td>
<td>the job manager failed to stage the executable</td>
<td>Check that the file service hosting the executable is reachable from the GRAM5 service node. Check that the executable exists on the file service node. Check that there is sufficient disk space in the user’s home directory on the service node to store the executable.</td>
</tr>
<tr>
<td>44</td>
<td>the job manager failed to stage the stdin file</td>
<td>Check that the file service hosting the standard input file is reachable from the GRAM5 service node. Check that the standard input file exists on the file service node. Check that there is sufficient disk space in the user’s home directory on the service node to store the standard input file.</td>
</tr>
<tr>
<td>45</td>
<td>the requested job manager type is invalid</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>46</td>
<td>the provided RSL 'arguments' parameter is invalid</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>47</td>
<td>the gatekeeper failed to run the job manager</td>
<td>Check the gatekeeper or job manager logs for more information.</td>
</tr>
<tr>
<td>48</td>
<td>the provided RSL could not be properly parsed</td>
<td>Check that the RSL string can be parsed by using <code>globusrun -p RSL</code>.</td>
</tr>
<tr>
<td>49</td>
<td>there is a version mismatch between GRAM components</td>
<td>Ask system administrator to upgrade GRAM service to GRAM2 or GRAM5</td>
</tr>
<tr>
<td>50</td>
<td>the provided RSL 'arguments' parameter is invalid</td>
<td>Check that the RSL arguments attribute evaluates to a sequence of strings.</td>
</tr>
<tr>
<td>51</td>
<td>the provided RSL 'count' parameter is invalid</td>
<td>Check that the RSL count attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>52</td>
<td>the provided RSL 'directory' parameter is invalid</td>
<td>Check that the RSL directory attribute evaluates to a string.</td>
</tr>
<tr>
<td>53</td>
<td>the provided RSL 'dryrun' parameter is invalid</td>
<td>Check that the RSL dryrun attribute evaluates to either yes or no.</td>
</tr>
<tr>
<td>54</td>
<td>the provided RSL 'environment' parameter is invalid</td>
<td>Check that the RSL environment attribute evaluates to a sequence of VARIABLE, VALUE pairs.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>-------------------</td>
</tr>
<tr>
<td>55</td>
<td>the provided RSL 'executable' parameter is invalid</td>
<td>Check that the RSL <code>executable</code> attribute evaluates to a string value.</td>
</tr>
<tr>
<td>56</td>
<td>the provided RSL 'host_count' parameter is invalid</td>
<td>Check that the RSL <code>host_count</code> attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>57</td>
<td>the provided RSL 'jobtype' parameter is invalid</td>
<td>Check that the RSL <code>jobtype</code> attribute evaluates to one of <code>single</code>, <code>multiple</code>, <code>mpi</code>, or <code>condor</code>.</td>
</tr>
<tr>
<td>58</td>
<td>the provided RSL 'maxtime' parameter is invalid</td>
<td>Check that the RSL <code>maxtime</code> attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>59</td>
<td>the provided RSL 'myjob' parameter is invalid</td>
<td>OBSOLETE IN GRAM5.</td>
</tr>
<tr>
<td>60</td>
<td>the provided RSL 'paradyn' parameter is invalid</td>
<td>OBSOLETE IN GRAM2.</td>
</tr>
<tr>
<td>61</td>
<td>the provided RSL 'project' parameter is invalid</td>
<td>Check that the RSL <code>project</code> attribute evaluates to a string value.</td>
</tr>
<tr>
<td>62</td>
<td>the provided RSL 'queue' parameter is invalid</td>
<td>Check that the RSL <code>queue</code> attribute evaluates to a string value.</td>
</tr>
<tr>
<td>63</td>
<td>the provided RSL 'stderr' parameter is invalid</td>
<td>Check that the RSL <code>stderr</code> attribute evaluates to a string value or a sequence of <code>DESTINATION</code> URLs with optional <code>CACHE_TAG</code> string parameters.</td>
</tr>
<tr>
<td>64</td>
<td>the provided RSL 'stdin' parameter is invalid</td>
<td>Check that the RSL <code>stdin</code> attribute evaluates to a string value.</td>
</tr>
<tr>
<td>65</td>
<td>the provided RSL 'stdout' parameter is invalid</td>
<td>Check that the RSL <code>stdout</code> attribute evaluates to a string value or a sequence of <code>DESTINATION</code> URLs with optional <code>CACHE_TAG</code> string parameters.</td>
</tr>
<tr>
<td>66</td>
<td>the job manager failed to locate an internal script</td>
<td>Check job manager log for more details.</td>
</tr>
<tr>
<td>67</td>
<td>the job manager failed on the system call pipe()</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>68</td>
<td>the job manager failed on the system call fcntl()</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>69</td>
<td>the job manager failed to create the temporary stdout filename</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>70</td>
<td>the job manager failed to create the temporary stderr filename</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>71</td>
<td>the job manager failed on the system call fork()</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>72</td>
<td>the executable file permissions do not allow execution</td>
<td>Check that the RSL executable attribute refers to an executable program or script.</td>
</tr>
<tr>
<td>73</td>
<td>the job manager failed to open stdout</td>
<td>Check that the RSL stdout attribute refers to one or more valid destination files or URLs.</td>
</tr>
<tr>
<td>74</td>
<td>the job manager failed to open stderr</td>
<td>Check that the RSL stderr attribute refers to one or more valid destination files or URLs.</td>
</tr>
<tr>
<td>75</td>
<td>the cache file could not be opened in order to relocate the user proxy</td>
<td>Check that the user's home directory is writable and not full on the GRAM5 service node.</td>
</tr>
<tr>
<td>76</td>
<td>cannot access cache files in ~/.globus/.gass_cache, check permissions,</td>
<td>Check that the user's home directory is writable and not full on the GRAM5 service node.</td>
</tr>
<tr>
<td></td>
<td>quota, and disk space</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>the job manager failed to insert the contact in the client contact list</td>
<td>Check job manager log</td>
</tr>
<tr>
<td>78</td>
<td>the contact was not found in the job manager's client contact list</td>
<td>Don't attempt to unregister callback contacts that are not registered.</td>
</tr>
<tr>
<td>79</td>
<td>connecting to the job manager failed. Possible reasons: job terminated,</td>
<td>Check that the job manager process is running. Check that the job manager credential has not expired. Check that the job manager contact refers to the correct TCP/IP host and port. Check that the job manager contact is not blocked by a firewall.</td>
</tr>
<tr>
<td></td>
<td>invalid job contact, network problems, ...</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>the syntax of the job contact is invalid</td>
<td>Check the syntax of job contact string.</td>
</tr>
<tr>
<td>81</td>
<td>the executable parameter in the RSL is undefined</td>
<td>Include the RSL executable in all job requests.</td>
</tr>
<tr>
<td>82</td>
<td>the job manager service is misconfigured. condor arch undefined</td>
<td>Add the -condor-arch to the command-line or configuration file for a job manager configured to use the condor LRM.</td>
</tr>
<tr>
<td>83</td>
<td>the job manager service is misconfigured. condor os undefined</td>
<td>Add the -condor-os to the command-line or configuration file for a job manager configured to use the condor LRM.</td>
</tr>
<tr>
<td>84</td>
<td>the provided RSL 'min_memory' parameter is invalid</td>
<td>Check that the RSL min_memory attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>85</td>
<td>the provided RSL 'max_memory' parameter is invalid</td>
<td>Check that the RSL max_memory attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>-------------------</td>
</tr>
<tr>
<td>86</td>
<td>the RSL 'min_memory' value is not zero or greater</td>
<td>Check that the RSL min_memory attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>87</td>
<td>the RSL 'max_memory' value is not zero or greater</td>
<td>Check that the RSL max_memory attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>88</td>
<td>the creation of a HTTP message failed</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>89</td>
<td>parsing incoming HTTP message failed</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>90</td>
<td>the packing of information into a HTTP message failed</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>91</td>
<td>an incoming HTTP message did not contain the expected information</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>92</td>
<td>the job manager does not support the service that the client requested</td>
<td>Check that the client is talking to the correct service</td>
</tr>
<tr>
<td>93</td>
<td>the gatekeeper failed to find the requested service</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>94</td>
<td>the jobmanager does not accept any new requests (shutting down)</td>
<td>Execute queries before the job has been cleaned up.</td>
</tr>
<tr>
<td>95</td>
<td>the client failed to close the listener associated with the callback URL</td>
<td>Call <code>globus_gram_client_callbackDisallow()</code> with a valid the callback contact.</td>
</tr>
<tr>
<td>96</td>
<td>the gatekeeper contact cannot be parsed</td>
<td>Check the syntax of the gatekeeper contact string you are attempting to contact.</td>
</tr>
<tr>
<td>97</td>
<td>the job manager could not find the 'poe' command</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>98</td>
<td>the job manager could not find the 'mpirun' command</td>
<td>Configure the LRM script with mpirun in your path.</td>
</tr>
<tr>
<td>99</td>
<td>the provided RSL 'start_time' parameter is invalid</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>100</td>
<td>the provided RSL 'reservation_handle' parameter is invalid</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>101</td>
<td>the provided RSL 'max_wall_time' parameter is invalid</td>
<td>Check that the RSL max_wall_time attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>102</td>
<td>the RSL 'max_wall_time' value is not zero or greater</td>
<td>Check that the RSL max_wall_time attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td>103</td>
<td>the provided RSL 'max_cpu_time' parameter is invalid</td>
<td>Check that the RSL \texttt{max_cpu_time} attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>104</td>
<td>the RSL 'max_cpu_time' value is not zero or greater</td>
<td>Check that the RSL \texttt{max_cpu_time} attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>105</td>
<td>the job manager is misconfigured, a scheduler script is missing</td>
<td>Check that the administrator has configured the LRM by running its setup script.</td>
</tr>
<tr>
<td>106</td>
<td>the job manager is misconfigured, a scheduler script has invalid permissions</td>
<td>Check that the administrator has installed the \texttt{GLLOBUS_LOCATION/libexec/globus-job-manager-script.pl} script. Check that the file system containing that script allows file execution.</td>
</tr>
<tr>
<td>107</td>
<td>the job manager failed to signal the job</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>108</td>
<td>the job manager did not recognize/support the signal type</td>
<td>Check that your signal operation is using the correct signal constant.</td>
</tr>
<tr>
<td>109</td>
<td>the job manager failed to get the job id from the local scheduler</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>110</td>
<td>the job manager is waiting for a commit signal</td>
<td>Send a two-phase commit signal to the job manager to acknowledge receiving the job contact from the job manager.</td>
</tr>
<tr>
<td>111</td>
<td>the job manager timed out while waiting for a commit signal</td>
<td>Send a two-phase commit signal to the job manager to acknowledge receiving the job contact from the job manager. Increase the two-phase commit time out for your job. Check that the job manager contact TCP/IP port is reachable from your client.</td>
</tr>
<tr>
<td>112</td>
<td>the provided RSL 'save_state' parameter is invalid</td>
<td>Check that the RSL \texttt{save_state} attribute is set to \texttt{yes} or \texttt{no}.</td>
</tr>
<tr>
<td>113</td>
<td>the provided RSL 'restart' parameter is invalid</td>
<td>Check that the RSL \texttt{restart} attribute evaluates to a string containing a job contact string.</td>
</tr>
<tr>
<td>114</td>
<td>the provided RSL 'two_phase' parameter is invalid</td>
<td>Check that the RSL \texttt{two_phase} attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>115</td>
<td>the RSL 'two_phase' value is not zero or greater</td>
<td>Check that the RSL \texttt{two_phase} attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>116</td>
<td>the provided RSL 'stdout_position' parameter is invalid</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>117</td>
<td>the RSL 'stdout_position' value is not zero or greater</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>118</td>
<td>the provided RSL 'stderr_position' parameter is invalid</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>119</td>
<td>the RSL 'stderr_position' value is not zero or greater</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>120</td>
<td>the job manager restart attempt failed</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>121</td>
<td>the job state file doesn't exist</td>
<td>Check that the job contact you are trying to restart matches one that the job manager returned to you.</td>
</tr>
<tr>
<td>122</td>
<td>could not read the job state file</td>
<td>Check that the state file directory is not full.</td>
</tr>
<tr>
<td>123</td>
<td>could not write the job state file</td>
<td>Check that the state file directory is not full.</td>
</tr>
<tr>
<td>124</td>
<td>old job manager is still alive</td>
<td>Contact the returned job manager contact to manage the job you are trying to restart.</td>
</tr>
<tr>
<td>125</td>
<td>job manager state file TTL expired</td>
<td>OBSOLETE in GRAM2</td>
</tr>
<tr>
<td>126</td>
<td>it is unknown if the job was submitted</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>127</td>
<td>the provided RSL 'remote_io_url' parameter is invalid</td>
<td>Check that the RSL remote_io_url attribute evaluates to a string value.</td>
</tr>
<tr>
<td>128</td>
<td>could not write the remote io url file</td>
<td>Check that the user's home file system on the job manager service node is writable and not full.</td>
</tr>
<tr>
<td>129</td>
<td>the standard output/error size is different</td>
<td>Send a stdio update signal to redirect the job manager output to a new URL</td>
</tr>
<tr>
<td>130</td>
<td>the job manager was sent a stop signal (job is still running)</td>
<td>Submit a restart request to monitor the job.</td>
</tr>
<tr>
<td>131</td>
<td>the user proxy expired (job is still running)</td>
<td>Generate a new proxy and then submit a restart request to monitor the job.</td>
</tr>
<tr>
<td>132</td>
<td>the job was not submitted by original jobmanager</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>133</td>
<td>the job manager is not waiting for that commit signal</td>
<td>Do not send a commit signal to a job that is not waiting for a commit signal.</td>
</tr>
<tr>
<td>134</td>
<td>the provided RSL scheduler specific parameter is invalid</td>
<td>Check the LRM-specific documentation to determine what val-</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ues are legal for the RSL extensions implemented by the LRM.</td>
</tr>
<tr>
<td>135</td>
<td>the job manager could not stage in a file</td>
<td>Check that the file service hosting the file to stage is reachable from the GRAM5 service node. Check that the file to stage exists on the file service node. Check that there is sufficient disk space in the user’s home directory on the service node to store the file to stage.</td>
</tr>
<tr>
<td>136</td>
<td>the scratch directory could not be created</td>
<td>Check that the directory named by the RSL <code>scratch_dir</code> attribute exists and is writable. Check that the directory named by the RSL <code>scratch_dir</code> attribute is not full.</td>
</tr>
<tr>
<td>137</td>
<td>the provided 'gass_cache' parameter is invalid</td>
<td>Check that the RSL <code>gass_cache</code> attribute evaluates to a string.</td>
</tr>
<tr>
<td>138</td>
<td>the RSL contains attributes which are not valid for job submission</td>
<td>Do not use restart- or signal-only RSL attributes when submitting a job.</td>
</tr>
<tr>
<td>139</td>
<td>the RSL contains attributes which are not valid for stdio update</td>
<td>Do not use submit- or restart-only RSL attributes when sending a stdio update signal to a job.</td>
</tr>
<tr>
<td>140</td>
<td>the RSL contains attributes which are not valid for job restart</td>
<td>Do not use submit- or signal-only RSL attributes when restarting a job.</td>
</tr>
<tr>
<td>141</td>
<td>the provided RSL 'file_stage_in' parameter is invalid</td>
<td>Check that the RSL <code>file_stage_in</code> attribute evaluates to a sequence of <code>SOURCE DESTINATION</code> pairs.</td>
</tr>
<tr>
<td>142</td>
<td>the provided RSL 'file_stage_in_shared' parameter is invalid</td>
<td>Check that the RSL <code>file_stage_in_shared</code> attribute evaluates to a sequence of <code>SOURCE DESTINATION</code> pairs.</td>
</tr>
<tr>
<td>143</td>
<td>the provided RSL 'file_stage_out' parameter is invalid</td>
<td>Check that the RSL <code>file_stage_out</code> attribute evaluates to a sequence of <code>SOURCE DESTINATION</code> pairs.</td>
</tr>
<tr>
<td>144</td>
<td>the provided RSL 'gass_cache' parameter is invalid</td>
<td>Check that the RSL <code>gass_cache</code> attribute evaluates to a string.</td>
</tr>
<tr>
<td>145</td>
<td>the provided RSL 'file_cleanup' parameter is invalid</td>
<td>Check that the RSL <code>file_clean_up</code> attribute evaluates to a sequence of strings.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>146</td>
<td>the provided RSL 'scratch_dir' parameter is invalid</td>
<td>Check that the RSL scratch_dir attribute evaluates to a string.</td>
</tr>
<tr>
<td>147</td>
<td>the provided scheduler-specific RSL parameter is invalid</td>
<td>Check the LRM-specific documentation to determine what values are legal for the RSL extensions implemented by the LRM.</td>
</tr>
<tr>
<td>148</td>
<td>a required RSL attribute was not defined in the RSL spec</td>
<td>Check that the RSL executable attribute is present in your job request RSL. Check that the RSL restart attributes is present in your restart RSL.</td>
</tr>
<tr>
<td>149</td>
<td>the gass_cache attribute points to an invalid cache directory</td>
<td>Check that the RSL gass_cache attributes evaluates to a directory that exists or can be created. Check that the user's home file system is writable and not full.</td>
</tr>
<tr>
<td>150</td>
<td>the provided RSL 'save_state' parameter has an invalid value</td>
<td>Check that the RSL save_state attribute has a value of yes or no.</td>
</tr>
<tr>
<td>151</td>
<td>the job manager could not open the RSL attribute validation file</td>
<td>Check that $GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf is present and readable on the job manager service node. Check that $GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf is readable on the job manager service node if present.</td>
</tr>
<tr>
<td>152</td>
<td>the job manager could not read the RSL attribute validation file</td>
<td>Check that $GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf is valid. Check that $GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf is valid if present.</td>
</tr>
<tr>
<td>153</td>
<td>the provided RSL 'proxy_timeout' is invalid</td>
<td>Check that RSL proxy_timeout attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>154</td>
<td>the RSL 'proxy_timeout' value is not greater than zero</td>
<td>Check that RSL proxy_timeout attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>155</td>
<td>the job manager could not stage out a file</td>
<td>Check that the source file being staged exists on the job manager.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td></td>
<td>service node. Check that the directory of the destination file being staged exists on the file service node. Check that the directory of the destination file being staged is writable by the user. Check that the destination file service is reachable by the job manager service node.</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>the job contact string does not match any which the job manager is handling</td>
<td>Check that the job contact string matches one returned from a job request.</td>
</tr>
<tr>
<td>157</td>
<td>proxy delegation failed</td>
<td>Check that the job manager service node trusts the signer of your credential. Check that you trust the signer of the job manager service node's credential.</td>
</tr>
<tr>
<td>158</td>
<td>the job manager could not lock the state lock file</td>
<td>Check that the file system holding the job state directory supports POSIX advisory locking. Check that the job state directory is writable by the user on the service node. Check that the job state directory is not full.</td>
</tr>
<tr>
<td>159</td>
<td>an invalid globus_io_clientattr_t was used.</td>
<td>Check that you have initialized the globus_io_clientattr_t attribute prior to using it with the GRAM client API.</td>
</tr>
<tr>
<td>160</td>
<td>an null parameter was passed to the gram library</td>
<td>Check that you are passing legal values to all GRAM API calls.</td>
</tr>
<tr>
<td>161</td>
<td>the job manager is still streaming output</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>162</td>
<td>the authorization system denied the request</td>
<td>Check with your GRAM system administrator to allow a particular certificate to be authorized.</td>
</tr>
<tr>
<td>163</td>
<td>the authorization system reported a failure</td>
<td>Check with your system administrator to verify that the authorization system is configured properly.</td>
</tr>
<tr>
<td>164</td>
<td>the authorization system denied the request - invalid job id</td>
<td>Check with your system administrator to verify that the authorization system is configured properly. Use a credential which is authorized to interact with a particular GRAM job.</td>
</tr>
<tr>
<td>165</td>
<td>the authorization system denied the request - not authorized to run the specified executable</td>
<td>Check with your system administrator to verify that the authorization system is configured properly.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ly. Use a credential which is authorized to interact with a particular GRAM job.</td>
</tr>
<tr>
<td>166</td>
<td>the provided RSL 'user_name' parameter is invalid.</td>
<td>Check that the RSL <code>user_name</code> attribute evaluates to a string.</td>
</tr>
<tr>
<td>167</td>
<td>the job is not running in the account named by the 'user_name' parameter.</td>
<td>Ask with the GRAM system administrator to add an authorization entry to allow your credential to run jobs as the specified user account.</td>
</tr>
</tbody>
</table>
Chapter 7. Known Problems in GRAM5

1. Known Problems in GRAM5

- GT-48\(^1\): Held Condor jobs should be reported as SUSPENDED
- GT-53\(^2\): RSL eval doesn't indicate what symbol was not found

\(^1\) [http://jira.globus.org/browse/GT-48](http://jira.globus.org/browse/GT-48)
\(^2\) [http://jira.globus.org/browse/GT-53](http://jira.globus.org/browse/GT-53)
Chapter 8. Usage statistics collection by the Globus Alliance

1. GRAM5-specific usage statistics

The following usage statistics are sent by default in a UDP packet (in addition to the GRAM component code, packet version, timestamp, and source IP address) at the end of each job.

- Job Manager Session ID
- dryrun used
- RSL Host Count
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_UNSUBMITTED
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_FILE_STAGE_IN
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_FILE_STAGE_OUT
- Timestamp when job hit GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE
- Job Failure Code
- Number of times status is called
- Number of times register is called
- Number of times signal is called
- Number of times refresh is called
- Number of files named in file_clean_up RSL
- Number of files being staged in (including executable, stdin) from http servers
- Number of files being staged in (including executable, stdin) from https servers
- Number of files being staged in (including executable, stdin) from ftp servers
- Number of files being staged in (including executable, stdin) from gsiftp servers
- Number of files being staged into the GASS cache from http servers
- Number of files being staged into the GASS cache from https servers
- Number of files being staged into the GASS cache from ftp servers
- Number of files being staged into the GASS cache from gsiftp servers
• Number of files being staged out (including stdout and stderr) to http servers
• Number of files being staged out (including stdout and stderr) to https servers
• Number of files being staged out (including stdout and stderr) to ftp servers
• Number of files being staged out (including stdout and stderr) to gsiftp servers
• Bitmask of used RSL attributes (values are $2^{id}$ from the gram5_rsl_attributes table)
• Number of times unregister is called
• Value of the count RSL attribute
• Comma-separated list of string names of other RSL attributes not in the set defined in globus-gram-job-manager.rvf
• Job type string
• Number of times the job was restarted
• Total number of state callbacks sent to all clients for this job

The following information can be sent as well in a job status packet but it is not sent unless explicitly enabled by the system administrator:

• Value of the executable RSL attribute
• Value of the arguments RSL attribute
• IP address and port of the client that submitted the job
• User DN of the client that submitted the job

In addition to job-related status, the job manager sends information periodically about its execution status. The following information is sent by default in a UDP packet (in addition to the GRAM component code, packet version, timestamp, and source IP address) at job manager start and every 1 hour during the job manager lifetime:

• Job Manager Start Time
• Job Manager Session ID
• Job Manager Status Time
• Job Manager Version
• LRM
• Poll used
• Audit used
• Number of restarted jobs
• Total number of jobs
• Total number of failed jobs
• Total number of canceled jobs
• Total number of completed jobs
• Total number of dry-run jobs
• Peak number of concurrently managed jobs
• Number of jobs currently being managed
• Number of jobs currently in the UNSUBMITTED state
• Number of jobs currently in the STAGE_IN state
• Number of jobs currently in the PENDING state
• Number of jobs currently in the ACTIVE state
• Number of jobs currently in the STAGE_OUT state
• Number of jobs currently in the FAILED state
• Number of jobs currently in the DONE state

Also, please see our policy statement\(^1\) on the collection of usage statistics.

\(^1\)/toolkit/docs/latest-stable/Usage_Stats.html
Glossary

some terms not in the docs but wanted in glossary: Condor TORQUE Portable Batch System Oracle GridEngine

C
Condor
A Local Resource Manager mechanism supported by GRAM. See the Condor Project Website\(^1\) for more information.
See Also Local Resource Manager.

G
Oracle GridEngine
A Local Resource Manager supported by GRAM. See Oracle's Web Site\(^2\) for more information.
See Also Local Resource Manager.

L
Local Resource Manager (LRM)
A system which controls access to a compute resource, such as a compute cluster or parallel computer. Such systems provide batch execution interfaces, which GRAM uses to execute jobs. Condor, Portable Batch System, GridEngine are examples of local resource managers.
See Also Condor, Portable Batch System, Oracle GridEngine.

P
Portable Batch System (PBS)
A Local Resource Manager mechanism supported by GRAM. Multiple implementations of PBS exist: GRAM currently supports TORQUE. See also TORQUE.
See Also Local Resource Manager, TORQUE.

T
TORQUE
An implementation of the Portable Batch System interface by Adaptive Computing\(^{11}\).
See Also Portable Batch System.

---

\(^1\) http://www.cs.wisc.edu/condor/
\(^{11}\) http://www.adaptivecomputing.com/products/torque.php
Index

E
errors, 40

J
jobs
  preparing
  generate valid proxy, 3, 3

T
troubleshooting, 15
  errors, 15

U
usage statistics, 54
GT 5.2.2 GRAM5: Developer's Guide
GT 5.2.2 GRAM5: Developer’s Guide

Introduction

This guide is intended to help a developer interact with GRAM5. It includes sections on implementing clients in C and implementing a Local Resource Manager interface, as well as an overview of concepts and APIs used to interact with GRAM.
Table of Contents

I. Introduction ............................................................................................................................. 1
   1. Before you begin .................................................................................................................. 3
   2. GRAM5 Concepts for Developers ...................................................................................... 5
II. GRAM Client Developer's Guide ........................................................................................... 9
   3. Basic GRAM Client Scenarios .......................................................................................... 11
   4. Advanced GRAM Client Scenarios ................................................................................... 24
III. GRAM Server Developer's Guide .......................................................................................... 34
   5. LRM Adapter Tutorial ....................................................................................................... 36
IV. GRAM5 Developer's Reference ............................................................................................ 61
   6. APIs .................................................................................................................................. 63
   7. RSL Specification v1.1 ....................................................................................................... 83
   8. Debugging ......................................................................................................................... 92
   9. Troubleshooting .............................................................................................................. 94
  10. Semantics and syntax of protocols ................................................................................ 113
  11. Related Documentation .................................................................................................. 121
  12. Internal Components ....................................................................................................... 122
Glossary ....................................................................................................................................... 123
Index ........................................................................................................................................... 125
List of Figures

10.1. GRAM State Transitions .............................................................. 120
List of Tables

2.1. GRAM Contact String Types ................................................................. 5
6.1. GRAM Client APIs .............................................................................. 63
9.1. GRAM5 Errors ..................................................................................... 100
10.1. GRAM Job States .............................................................................. 119
List of Examples

2.1. Gatekeeper Service Contact Examples ................................................................. 7
5.1. fake.rvf ............................................................................................................. 39
7.1. Quoted Literal Examples ................................................................................... 85
7.2. GRAM5 Job Request Examples ......................................................................... 90
Part I. Introduction
# Table of Contents

1. Before you begin .................................................................................................................. 3  
   1. Feature summary ............................................................................................................. 3  
   2. Tested platforms .............................................................................................................. 3  
   3. Backward compatibility summary ................................................................................... 4  
   4. Technology dependencies ............................................................................................... 4  
   5. Security Considerations ................................................................................................. 4  
2. GRAM5 Concepts for Developers ...................................................................................... 5  
   1. Blocking and Nonblocking Function Variants ............................................................... 5  
   2. Service Contact Strings .................................................................................................. 5  
   3. Job State Callbacks and Polling ...................................................................................... 7  
   4. Credential Management ................................................................................................. 7  
   5. RSL ................................................................................................................................. 8
Chapter 1. Before you begin

1. Feature summary

New Features new since 5.2.1:

- Improved memory management and process management.
- Improved scalability and reliability

Other Standard Supported Features

- Remote job execution and management
- Uniform and flexible interface to local resource managers
- File staging before and after job execution
- File and directory clean up after job termination
- Service auditing for each submitted

Removed Features

- Condor SEG module is no longer included. Its functionality has been moved into the core of the job manager program.

2. Tested platforms

Tested platforms for GRAM5:

- Linux
  - CentOS 4 x86_64
  - CentOS 5, 6 x86_64, i386
  - Fedora 15, 16, 17 x86_64, i386
  - Red Hat Enterprise Linux 5, 6 x86_64, i386
  - Scientific Linux 5, 6 x86_64, i386
  - Debian 6, 7 (testing) x86_64, i386
  - Ubuntu 10.04LTS, 10.10, 11.04, 11.10, 12.04LTS x86_64, i386
- Mac OS X
  - Mac OS X 10.7 (Lion)
- Solaris
  - Solaris 11
3. Backward compatibility summary

Protocol changes in GRAM since GT4 series:

- The GRAM5 service uses a superset of the GRAM2 protocol for communication between the client and service. The extensions supported in GRAM5 are implemented in such a way that they are ignored by GRAM2 services or clients. These extensions provide improved error messages and version detection.

- GRAM5 does not support task co-allocation using DUROC and its related protocols. Jobs submitted using DUROC directives will fail.

- GRAM5 does not support file streaming. The standard output and standard error streams are sent after the job completes instead of during execution. As a special case, support for the Condor grid monitor program implements a small subset of the streaming capabilities of GRAM2 in GT 4.2.x.

4. Technology dependencies

GRAM depends on the following GT components:

- Globus Common
- GSI C
- GridFTP server

5. Security Considerations

5.1. Gatekeeper Security Considerations

GRAM5 runs different parts of itself under different privilege levels. The `globus-gatekeeper` runs as root, and uses its root privilege to access the host’s private key. It uses the `grid map file` to map `Grid Certificates` to local user ids and then uses the `setuid()` function to change to that user and execute the `globus-job-manager` program.

5.2. Job Manager Security Considerations

The `globus-job-manager` program runs as a local non-root account. It receives a delegated limited `proxy certificate` from the GRAM5 client which it uses to access Grid storage resources via GridFTP and to authenticate job signals (such as client cancel requests), and send job state callbacks to registered clients. This proxy is generally short-lived, and is automatically removed by the job manager when the job completes.

The `globus-job-manager` program uses a publicly-writable directory for job state files. This directory has the `sticky` bit set, so users may not remove other users files. Each file is named by a UUID, so it should be unique.

5.3. Fork SEG Module Security Considerations

The Fork Scheduler Event Generator module uses a globally writable file for job state change events. This is not recommended for production use.
Chapter 2. GRAM5 Concepts for Developers

1. Blocking and Nonblocking Function Variants

In the GRAM Client API, all functions that involve sending messages over the network have both blocking and non-blocking variants. These are useful in different programming situations.

The blocking variants, such as the `globus_gram_client_job_request` function require less application code, but will prevent subsequent instructions from executing until the request has been sent and the reply parsed. In a non-threaded environment, other callback functions registered with the Globus event driver may be invoked while the blocking function is running. In a threaded environment, other events may occur in other threads while the function is blocking, but the current thread will be blocked until the response is parsed.

The nonblocking variants, such as `globus_gram_client_register_job_request` require the application to include a callback function which will be called by the Globus event driver when the reply has been parsed. In a non-threaded environment, applications must poll the event driver using functions from the `globus_poll` or `globus_cond_wait` families of functions. In a threaded environment, the callback function may be invoked in another thread than the one calling the non-blocking function, even before the non-blocking function has returned. Application writers must be careful in using synchronization primitives such as `globus_mutex_t` and `globus_cond_t` when using non-blocking functions.

An application writer should use the non-blocking variants if the application will be submitting many jobs concurrently or requires custom network or security attributes. Using the non-blocking variants allows the Globus event driver to better schedule network I/O in these cases.

2. Service Contact Strings

GRAM uses three types of `contact strings` to describe how to contact different services. These service contacts are:

Table 2.1. GRAM Contact String Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gatekeeper Service Contact</td>
<td>This string describes how to contact a gatekeeper service. It is used to submit jobs, send &quot;ping&quot; requests to determine if a service is properly deployed, and version requests to determine what version of the software is deployed. Full details of the syntax of this contact is located in the next section.</td>
</tr>
<tr>
<td>Callback Contact</td>
<td>This string is an HTTPS URL that is an endpoint for GRAM job state callbacks. An https message is posted to this address when the Job Manager detects a job state change.</td>
</tr>
<tr>
<td>Job Contact</td>
<td>This string is an HTTPS URL that is an endpoint for contacting an existing GRAM job. An https message is posted to this address to cancel, signal, or query a GRAM job.</td>
</tr>
</tbody>
</table>
2.1. Resource Names

In GRAM5, a Gatekeeper Service Contact contains the host, port, service name, and service identity required to contact a particular GRAM service. For convenience, default values are used when parts of the contact are omitted. An example of a full gatekeeper service contact is `grid.example.org:2119/jobmanager:/C=US/O=Example/OU=Grid/CN=host/grid.example.org`.

The various forms of the resource name using default values follow:

- `HOST`
- `HOST:PORT`
- `HOST:PORT/SERVICE`
- `HOST/SERVICE`
- `HOST:/SERVICE`
- `HOST:PORT:SUBJECT`
- `HOST/SERVICE:SUBJECT`
- `HOST:/SERVICE:SUBJECT`
- `HOST:PORT/SERVICE:SUBJECT`

Where the various values have the following meaning:

`HOST` Network name of the machine hosting the service.

`PORT` Network port number that the service is listening on. If not specified, the default of 2119 is used.

`SERVICE` Path of the service entry in `$GLOBUS_LOCATION/etc/grid-services`. If not specified, the default of `jobmanager` is used.

`SUBJECT` X.509 identity of the credential used by the service. If not specified, the default of `host@HOST` is used.
Example 2.1. Gatekeeper Service Contact Examples

The following strings all name the service grid.example.org:2119/jobmanager:/C=US/O=Example/OU=Grid/CN=host/grid.example.org using the formats with the various defaults described above.

- grid.example.org
- grid.example.org:2119
- grid.example.org:2119/jobmanager
- grid.example.org/jobmanager
- grid.example.org:/jobmanager
- grid.example.org:2119:/C=US/O=Example/OU=Grid/CN=host/grid.example.org
- grid.example.org/jobmanager:/C=US/O=Example/OU=Grid/CN=host/grid.example.org
- grid.example.org:/jobmanager:/C=US/O=Example/OU=Grid/CN=host/grid.example.org
- grid.example.org:2119/jobmanager:/C=US/O=Example/OU=Grid/CN=host/grid.example.org

3. Job State Callbacks and Polling

GRAM clients and learn about a job’s state in two ways: by registering for job state callbacks and by polling for status. These two methods have different performance characteristics and costs.

In order to receive job state callbacks, a client application must create an HTTPS listener using the globus_gram_client_callback_allow or globus_gram_client_info_callback_allow functions. A non-threaded application must then periodically call a function from either the globus_cond_wait or globus_poll families in order to process the job state callbacks. Additionally, the network must be configured to allow the GRAM job manager to send messages to the port that the client is listening on. This may be difficult if there is a firewall between the client and service.

The GRAM service initiates the job state callbacks, and thus they are usually sent very shortly after the job state changes, so clients can be notified about the state changes quickly.

In order to poll for job states, a client can call either the blocking or nonblocking variant of the globus_gram_client_job_status or globus_gram_client_job_status_with_info functions. These functions require that the network be configured to allow the client to contact the network port that the GRAM service is listening on (the Job Contact).

The client initiates these polling operations, so they are only as accurate as the polling frequency of the client. If the client polls very often, it will receive job state changes more quickly, at the risk of increasing the computing and network cost of both the client and service.

4. Credential Management

The GRAM5 protocols all use GSSAPIv2 abstractions to provide authentication and authorization. By default, GRAM uses an SSL-based GSSAPI for its security.
The client delegates a credential to the gatekeeper service after authentication, and the GRAM job manager service uses this delegated credential as both a job-specific credential and for subsequent communication with GRAM clients.

If a client or clients submit multiple jobs to a gatekeeper service, they will eventually all be handled by a single job manager process. This process will use whichever delegated credential will remain valid the longest for accepting new connections and connecting to clients to send job state callbacks. When a client delegates a new credential to a job, this credential may also be used as the job manager's credential for future connections.

5. RSL

GRAM5 jobs are described using the RSL language. The GRAM client API submits jobs using the string representation of the RSL, rather than the RSL parse tree. Clients can, if they need to modify or construct RSL at runtime, use the functions in the RSL API to do so.
Part II. GRAM Client Developer's Guide
# Table of Contents

3. Basic GRAM Client Scenarios .................................................................................................................. 11
   1. "Ping" a Job Manager .......................................................................................................................... 11
   2. Check a Job Manager Version ........................................................................................................... 12
   3. Submitting a Job ................................................................................................................................. 14
   4. Submitting a Job and Processing Job State Callbacks ...................................................................... 16
   5. Polling Job Status ............................................................................................................................... 19
   6. Canceling a Job .................................................................................................................................. 21
   7. Refreshing Job Credential ................................................................................................................... 22

4. Advanced GRAM Client Scenarios ......................................................................................................... 24
   1. Non-blocking Job Submission ............................................................................................................ 24
   2. Custom Security Attributes .............................................................................................................. 27
   3. Modifying RSL .................................................................................................................................. 30
Chapter 3. Basic GRAM Client Scenarios

This chapter contains a series of examples demonstrating how to use different features of the GRAM APIs to interact with the GRAM service. These examples can be compiled by using GNU make with the makefile from Makefile.examples.

1. "Ping" a Job Manager

This example shows how to use a gatekeeper "ping" request to determine if a service is running and if the client is authorized to contact it. It takes a gatekeeper service contact as its only command-line option. The source to this example\(^1\) can be downloaded.

```c
/*
 * These headers contain declarations for the globus_module functions
 * and GRAM Client API functions
 */
#include "globus_common.h"
#include "globus_gram_client.h"
#include <stdio.h>

int
main(int argc, char *argv[])
{
    int rc;

    if (argc != 2)
    {
        fprintf(stderr, "Usage: %s RESOURCE-MANAGER-CONTACT\n", argv[0]);
        rc = 1;
        goto out;
    }

    printf("Pinging GRAM resource: %s\n", argv[1]);

    /*
     * Always activate the GLOBUS_GRAM_CLIENT_MODULE prior to using any
     * functions from the GRAM Client API or behavior is undefined.
     */
    rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
    if (rc != GLOBUS_SUCCESS)
    {
        fprintf(stderr, "Error activating %s because %s (Error %d)\n",
                GLOBUS_GRAM_CLIENT_MODULE->module_name,
                globus_gram_client_error_string(rc),
                rc);
        goto out;
    }

    gram_ping_example.c
```

\(^1\) gram_ping_example.c
/* Ping the service passed as our first command-line option. If successful, * this function will return GLOBUS_SUCCESS, otherwise an integer * error code. */
rc = globus_gram_client_ping(argv[1]);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Unable to ping service at %s because %s (Error %d)\n", 
            argv[1], globus_gram_client_error_string(rc), rc);
}
else
{
    printf("Ping successful\n");
}
/* Deactivating the module allows it to free memory and close network * connections. */
rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
out:
return rc;

2. Check a Job Manager Version

This example shows how to use the "version" command to determine what software version a gatekeeper service is running. The source to this example\(^2\) can be downloaded.

\(^2\) gram_version_example.c
rc = 1;
goto out;
}

printf("Checking version of GRAM resource: %s\n", argv[1]);

/*@ Always activate the GLOBUS_GRAM_CLIENT_MODULE prior to using any
* functions from the GRAM Client API or behavior is undefined. */
rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Error activating %s because %s (Error %d)\n",
            GLOBUS_GRAM_CLIENT_MODULE->module_name,
            globus_gram_client_error_string(rc),
            rc);
    goto out;
}

/*@ Contact the service passed as our first command-line option and perform
* a version check. If successful,
* this function will return GLOBUS_SUCCESS, otherwise an integer
* error code. Old versions of the job manager will return
* GLOBUS_GRAM_PROTOCOL_ERROR_HTTP_UNPACK_FAILED as they do not support
* the version operation. */
rc = globus_gram_client_get_jobmanager_version(argv[1], &extensions);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Unable to get service version from %s because %s "
            "(Error %d)\n",
            argv[1], globus_gram_client_error_string(rc),
            rc);
}
else
{
    /* The version information is returned in the extensions hash table */
    extension_value = globus_hashtable_lookup(
            &extensions,
            "toolkit-version");

    if (extension_value == NULL)
    {
        printf("Unknown toolkit version\n");
    }
    else
    {
        printf("Toolkit Version: %s\n", extension_value->value);
    }
    extension_value = globus_hashtable_lookup(
            &extensions,
            "version");
if (extension_value == NULL) {
    printf("Unknown package version\n");
} else {
    printf("Package Version: %s\n", extension_value->value);
} /* Free the extensions hash and its values */
globus_gram_protocol_hash_destroy(&extensions);

/* Deactivating the module allows it to free memory and close network
* connections. */
rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
out:
    return rc;
} /* End of gram_version_example.c */

3. Submitting a Job

This example shows how to submit a job to a GRAM service. The source to this example\(^3\) can be downloaded.

/*
* These headers contain declarations for the globus_module functions
* and GRAM Client API functions
*/
#include "globus_common.h"
#include "globus_gram_client.h"
#include <stdio.h>

int
main(int argc, char *argv[]) {
    int rc;
    char * job_contact = NULL;
    if (argc != 3) {
        fprintf(stderr, "Usage: %s RESOURCE-MANAGER-CONTACT RSL\n", argv[0]);
        rc = 1;
        goto out;
    }
    printf("Submitting job to GRAM resource: %s\n", argv[1]);
    /*
    ^3 gram_submit_example.c
Always activate the GLOBUS_GRAM_CLIENT_MODULE prior to using any
functions from the GRAM Client API or behavior is undefined.

```
rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Error activating \%s because \%s (Error \%d)\n", 
            GLOBUS_GRAM_CLIENT_MODULE->module_name,
            globus_gram_client_error_string(rc),
            rc);
    goto out;
}
```

Submit the job request to the service passed as our first command-line
option. If successful, this function will return GLOBUS_SUCCESS,
otherwise an integer error code.

```
rc = globus_gram_client_job_request(
    argv[1], argv[2], 0, NULL, &job_contact);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Unable to submit job to \%s because \%s (Error \%d)\n", 
            argv[1], globus_gram_client_error_string(rc), rc);
    if (job_contact != NULL)
    {
        printf("Job Contact: \%s\n", job_contact);
    }
}
else
{
    /* Display job contact string */
    printf("Job submit successful: \%s\n", job_contact);
}
if (job_contact != NULL)
{
    free(job_contact);
}
```

Deactivating the module allows it to free memory and close network
connections.

```
rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
```

```
/* End of gram_submit_example.c */
4. Submitting a Job and Processing Job State Callbacks

This example shows how to submit a job to a GRAM service and then wait until the job reaches the FAILED or DONE state. The source to this example can be downloaded.

```c
/*
 * These headers contain declarations for the globus_module functions
 * and GRAM Client API functions
 */
#include "globus_common.h"
#include "globus_gram_client.h"

#include <stdio.h>

struct monitor_t
{
    globus_mutex_t mutex;
    globus_cond_t cond;
    globus_gram_protocol_job_state_t state;
};

/*
 * Job State Callback Function
 * This function is called when the job manager sends job states.
 */
static
void
example_callback(void * callback_arg, char * job_contact, int state, int errorcode)
{
    struct monitor_t * monitor = callback_arg;

    globus_mutex_lock(&monitor->mutex);

    printf("Old Job State: %d\nNew Job State: %d\n", monitor->state, state);
    monitor->state = state;

    if (state == GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED || 
        state == GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE)
    {
        globus_cond_signal(&monitor->cond);
    }
    globus_mutex_unlock(&monitor->mutex);
}

int
main(int argc, char *argv[])
{
    gram_submit_and_wait_example.c
```

{
    int rc;
    char * callback_contact = NULL;
    char * job_contact = NULL;
    struct monitor_t monitor;

    if (argc != 3)
    {
        fprintf(stderr, "Usage: %s RESOURCE-MANAGER-CONTACT RSL\n", argv[0]);
        rc = 1;
        goto out;
    }

    /*
    * Always activate the GLOBUS_GRAM_CLIENT_MODULE prior to using any
    * functions from the GRAM Client API or behavior is undefined.
    */
    rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
    if (rc != GLOBUS_SUCCESS)
    {
        fprintf(stderr, "Error activating %s because %s (Error %d)
",
            GLOBUS_GRAM_CLIENT_MODULE->module_name,
            globus_gram_client_error_string(rc),
            rc);
        goto out;
    }

    rc = globus_mutex_init(&monitor.mutex, NULL);
    if (rc != GLOBUS_SUCCESS)
    {
        fprintf(stderr, "Error initializing mutex
");  goto deactivate;
    }

    rc = globus_cond_init(&monitor.cond, NULL);
    if (rc != GLOBUS_SUCCESS)
    {
        fprintf(stderr, "Error initializing condition variable\n");
        goto destroy_mutex;
    }

    monitor.state = GLOBUS_GRAM_PROTOCOL_JOB_STATE_UNSUBMITTED;

    globus_mutex_lock(&monitor.mutex);

    /*
    * Allow GRAM state change callbacks
    */
    rc = globus_gram_client_callback_allow(
        example_callback, &monitor, &callback_contact);
    if (rc != GLOBUS_SUCCESS)
    {
        fprintf(stderr, "Error allowing callbacks because %s (Error %d)\n",
            globus_gram_client_error_string(rc),
            rc);
    }

    goto out;
}

out:

deactivate:

destroy_mutex:


goto destroy_cond;
}

/*
 * Submit the job request to the service passed as our first command-line
 * option.
 */

rc = globus_gram_client_job_request(
    argv[1], argv[2],
    GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED|
    GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE,
    callback_contact, &job_contact);

if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Unable to submit job to %s because %s (Error %d)\n", 
            argv[1], globus_gram_client_error_string(rc), rc);
    /* Job submit failed. Short circuit the while loop below by setting
     * the job state to failed
     */
    monitor.state = GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED;
}
else
{
    /* Display job contact string */
    printf("Job submit successful: %s\n", job_contact);
}

/* Wait for job state callback to let us know the job has completed */
while (monitor.state != GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE &&
    monitor.state != GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED)
{
    globus_cond_wait(&monitor.cond, &monitor.mutex);
}
rc = globus_gram_client_callback_disallow(callback_contact);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Error disabling callbacks because %s (Error %d)\n", 
            globus_gram_client_error_string(rc), rc);
}
globus_mutex_unlock(&monitor.mutex);

if (job_contact != NULL)
{
    free(job_contact);
}

destroy_cond:
    globus_cond_destroy(&monitor.cond);
destroy_mutex:
    globus_mutex_destroy(&monitor.mutex);
deactivate:
/*
 * Deactivating the module allows it to free memory and close network
 * connections.
Basic GRAM Client Scenarios

5. Polling Job Status

This example shows how to submit a job to a GRAM service and then wait until the job reaches the FAILED or DONE state. The source to this example\(^5\) can be downloaded.

```
/*
 * These headers contain declarations for the globus_module functions
 * and GRAM Client API functions
 */
#include "globus_common.h"
#include "globus_gram_client.h"
#include <stdio.h>

int main(int argc, char *argv[]) {
    int rc;
    int status = 0;
    int failure_code = 0;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s JOB-CONTACT\n", argv[0]);
        rc = 1;
        goto out;
    }

    /*
     * Always activate the GLOBUS_GRAM_CLIENT_MODULE prior to using any
     * functions from the GRAM Client API or behavior is undefined.
     */
    rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
    if (rc != GLOBUS_SUCCESS) {
        fprintf(stderr, "Error activating %s because %s (Error %d)\n",
                GLOBUS_GRAM_CLIENT_MODULE->module_name,
                globus_gram_client_error_string(rc),
                rc);
        goto out;
    }

    /*
     * Check the job status of the job named by the first argument to
     * this program.
     */
```

\(^5\) gram_poll_example.c
/
rc = globus_gram_client_job_status(argv[1], &status, &failure_code);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Unable to check job status because %s (Error %d)\n",
            globus_gram_client_error_string(rc), rc);
}
else
{
    switch (status)
    {
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_UNSUBMITTED:
        printf("Unsubmitted\n");
        break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_IN:
        printf("StageIn\n");
        break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING:
        printf("Pending\n");
        break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE:
        printf("Active\n");
        break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_SUSPENDED:
        printf("Suspended\n");
        break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_OUT:
        printf("StageOut\n");
        break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
        printf("Done\n");
        break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
        printf("Failed (%d)\n", failure_code);
        break;
    default:
        printf("Unknown job state\n");
        break;
    }
}
/*
 * Deactivating the module allows it to free memory and close network
 * connections.
 */
rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
out:
    return rc;
} /* End of gram_poll_example.c */
6. Canceling a Job

This example shows how to cancel a job being run by a GRAM service. The source to this example\(^6\) can be downloaded.

```c
#include "globus_common.h"
#include "globus_gram_client.h"
#include <stdio.h>

int main(int argc, char *argv[]) {
    int rc;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s JOB-CONTACT\n", argv[0]);
        rc = 1;
        goto out;
    }

    rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
    if (rc != GLOBUS_SUCCESS)
        goto out;

    rc = globus_gram_client_job_cancel(argv[1]);
    if (rc != GLOBUS_SUCCESS)
        fprintf(stderr, "Unable to cancel job because %s (Error %d)\n",
                globus_gram_client_error_string(rc), rc);

out:
    return rc;
}
```

\(^6\)gram_cancel_example.c
Deactivating the module allows it to free memory and close network connections.
rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);

7. Refreshing Job Credential

This example shows how to refresh a GRAM job's credential after the job has been submitted by some other means. The source to this example can be downloaded.

#include "globus_common.h"
#include "globus_gram_client.h"
#include <stdio.h>

int
main(int argc, char *argv[])
{
    int rc;
    if (argc != 2)
    {
        fprintf(stderr, "Usage: %s JOB-CONTACT\n", argv[0]);
        rc = 1;
        goto out;
    }

    printf("Refreshing Credential for GRAM Job: %s\n", argv[1]);

    rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
    if (rc != GLOBUS_SUCCESS)
    {
        fprintf(stderr, "Error activating %s because %s (Error %d)\n", GLOBUS_GRAM_CLIENT_MODULE->module_name, globus_gram_client_error_string(rc), rc);
        goto out;
    }

    /*
     * Always activate the GLOBUS_GRAM_CLIENT_MODULE prior to using any functions from the GRAM Client API or behavior is undefined.
     */
}

7 gram_refresh_example.c
Basic GRAM Client Scenarios

* Refresh the credential of the job running at the contact named
  * by the first command-line argument to this program. We'll use the
  * process's default credential by passing in GSS_C_NO_CREDENTIAL.
  */
rc = globus_gram_client_job_refresh_credentials(
    argv[1], GSS_C_NO_CREDENTIAL);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Unable to refresh credential for job %s because %s (Error %d)\n",
            argv[1], globus_gram_client_error_string(rc), rc);
}
else
{
    printf("Refresh successful\n");
}
/*
* Deactivating the module allows it to free memory and close network
* connections.
*/
rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
out:
    return rc;
}
/* End of gram_refresh_example.c */
Chapter 4. Advanced GRAM Client Scenarios

1. Non-blocking Job Submission

This example shows how to submit a series of GRAM jobs using the non-blocking function `globus_gram_client_register_job_request` and wait until all submissions have completed. This example throttles the number of concurrent job submissions to reduce the load on the service node. The source to this example\(^1\) can be downloaded.

```c
/*
 * These headers contain declarations for the globus_module functions
 * and GRAM Client API functions
*/
#include "globus_common.h"
#include "globus_gram_client.h"
#include <stdio.h>

struct monitor_t {
    globus_mutex_t mutex;
    globus_cond_t cond;
    int submit_pending;
    int successful_submits;
};

#define CONCURRENT_SUBMITS 5

static void example_submit_callback(
    void * user_callback_arg,
    globus_gram_protocol_error_t operation_failure_code,
    const char * job_contact,
    globus_gram_protocol_job_state_t job_state,
    globus_gram_protocol_error_t job_failure_code)
{
    struct monitor_t * monitor = user_callback_arg;

    globus_mutex_lock(&monitor->mutex);
    monitor->submit_pending--;
    if (monitor->submit_pending < CONCURRENT_SUBMITS)
    {
        globus_cond_signal(&monitor->cond);
    }
    printf("Submitted job %s\n",
           job_contact ? job_contact : "UNKNOWN");
    if (operation_failure_code == GLOBUS_SUCCESS)
    {
        // Further processing...
    }
}

\(^1\) gram_nonblocking_submit_example.c
```
Advanced GRAM Client Scenarios

```c
{    monitor->successful_submits++;
} else
{    printf("submit failed because %s (Error %d)\n",
            globus_gram_client_error_string(operation_failure_code),
            operation_failure_code);
}    globus_mutex_unlock(&monitor->mutex);
}

int main(int argc, char *argv[])
{
    int rc;
    int i;
    struct monitor_t monitor;

    if (argc < 3)
    {        fprintf(stderr, "Usage: %s RESOURCE-MANAGER-CONTACT RSL-SPEC...\n", 
                        argv[0]);
        rc = 1;
        goto out;
    }
    printf("Submiting %d jobs to %s\n", argc-2, argv[1]);

    /*
    * Always activate the GLOBUS_GRAM_CLIENT_MODULE prior to using any
    * functions from the GRAM Client API or behavior is undefined.
    */
    rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
    if (rc != GLOBUS_SUCCESS)
    {        fprintf(stderr, "Error activating %s because %s (Error %d)\n",
                        GLOBUS_GRAM_CLIENT_MODULE->module_name,
                        globus_gram_client_error_string(rc),
                        rc);
            goto out;
        }
    rc = globus_mutex_init(&monitor.mutex, NULL);
    if (rc != GLOBUS_SUCCESS)
    {        fprintf(stderr, "Error initializing mutex %d\n", rc);
            goto deactivate;
        }
    rc = globus_cond_init(&monitor.cond, NULL);
    if (rc != GLOBUS_SUCCESS)
```
monitor.submit_pending = 0;

/* Submits jobs from argv[2] until end of the argv array. At most
 * CONCURRENT_SUBMITS will be pending at any given time.
 */
globus_mutex_lock(&monitor.mutex);
for (i = 2; i < argc; i++)
{
    /* This throttles the number of concurrent job submissions */
    while (monitor.submit_pending >= CONCURRENT_SUBMITS)
    {
        globus_cond_wait(&monitor.cond, &monitor.mutex);
    }
    /* When the job has been submitted, the example_submit_callback
     * will be called, either from another thread or from a
     * globus_cond_wait in a nonthreaded build
     */
    rc = globus_gram_client_register_job_request(
        argv[1], argv[i], 0, NULL, NULL, example_submit_callback,
        &monitor);
    if (rc != GLOBUS_SUCCESS)
    {
        fprintf(stderr, "Unable to submit job %s because %s (Error %d)\n",
            argv[i], globus_gram_client_error_string(rc), rc);
    }
    else
    {
        monitor.submit_pending++;
    }
}

/* Wait until the example_submit_callback function has been called for
 * each job submission
 */
while (monitor.submit_pending > 0)
{
    globus_cond_wait(&monitor.cond, &monitor.mutex);
}

printf("Submitted %d jobs (%d successfully)\n", argc-2, monitor.successful_submits);

globus_cond_destroy(&monitor.cond);
deactivate:
    globus_mutex_destroy(&monitor.mutex);
deactivate:
Deactivating the module allows it to free memory and close network connections.

```c
rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
```

/* End of gram_nonblocking_submit_example.c */

### 2. Custom Security Attributes

This example shows how to submit a job and delegate a full credential to the job. The source to this example\(^2\) can be downloaded.

```c
/*
 * These headers contain declarations for the globus_module functions
 * and GRAM Client API functions
 */
#include "globus_common.h"
#include "globus_gram_client.h"

#include <stdio.h>

struct monitor_t
{
    globus_mutex_t mutex;
    globus_cond_t cond;
    globus_bool_t done;
};

static
void
example_submit_callback(
    void * user_callback_arg,
    globus_gram_protocol_error_t operation_failure_code,
    const char * job_contact,
    globus_gram_protocol_job_state_t job_state,
    globus_gram_protocol_error_t job_failure_code)
{
    struct monitor_t * monitor = user_callback_arg;

    globus_mutex_lock(&monitor->mutex);
    monitor->done = GLOBUS_TRUE;
    globus_cond_signal(&monitor->cond);
    if (operation_failure_code == GLOBUS_SUCCESS)
    {
        printf("Submitted job %s\n",
               job_contact ? job_contact : "UNKNOWN");
    }
    else
    {
        printf("submit failed because %s (Error %d)\n",\n               "Unknown", job_failure_code);
    }
}
```

\(^2\) gram_attr_example.c
globus_gram_client_error_string(operation_failure_code),
operation_failure_code);
}
globus_mutex_unlock(&monitor->mutex);
}

int
main(int argc, char *argv[])
{
  int rc;
globus_gram_client_attr_t attr;
struct monitor_t monitor;

  if (argc < 3)
  {
    fprintf(stderr, "Usage: %s RESOURCE-MANAGER-CONTACT RSL-SPEC...\n", argv[0]);
    rc = 1;
    goto out;
  }

  printf("Submiting job to %s with full proxy\n", argv[1]);

  /*
   * Always activate the GLOBUS_GRAM_CLIENT_MODULE prior to using any
   * functions from the GRAM Client API or behavior is undefined.
   */
  rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
  if (rc != GLOBUS_SUCCESS)
  {
    fprintf(stderr, "Error activating %s because %s (Error %d)\n", 
      GLOBUS_GRAM_CLIENT_MODULE->module_name,
      globus_gram_client_error_string(rc), rc);
    goto out;
  }

  rc = globus_mutex_init(&monitor.mutex, NULL);
  if (rc != GLOBUS_SUCCESS)
  {
    fprintf(stderr, "Error initializing mutex %d\n", rc);
    goto deactivate;
  }

  rc = globus_cond_init(&monitor.cond, NULL);
  if (rc != GLOBUS_SUCCESS)
  {
    fprintf(stderr, "Error initializing condition variable %d\n", rc);
    goto destroy_mutex;
  }

  monitor.done = GLOBUS_FALSE;
/ * Initialize attribute so that we can set the delegation attribute */
rc = globus_gram_client_attr_init(&attr);

/* Set the proxy attribute */
rc = globus_gram_client_attr_set_delegation_mode(
    attr,
    GLOBUS_IO_SECURE_DELEGATION_MODE_FULL_PROXY);

/* Submit the job rsl from argv[2] */
globus_mutex_lock(&monitor.mutex);
/* When the job has been submitted, the example_submit_callback * will be called, either from another thread or from a * globus_cond_wait in a nonthreaded build */
rc = globus_gram_client_register_job_request(
    argv[1], argv[2], 0, NULL, attr, example_submit_callback,
    &monitor);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Unable to submit job %s because %s (Error %d)\n",
        argv[2], globus_gram_client_error_string(rc), rc);
}

/* Wait until the example_submit_callback function has been called for * the job submission */
while (!monitor.done)
{
    globus_cond_wait(&monitor.cond, &monitor.mutex);
}
globus_mutex_unlock(&monitor.mutex);

globus_cond_destroy(&monitor.cond);

destroy_mutex:
    globus_mutex_destroy(&monitor.mutex);

deactivate:
    /*
    * Deactivating the module allows it to free memory and close network
    * connections.
    */
    rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);

out:
    return rc;

/* End of gram_attr_example.c */
3. Modifying RSL

This example shows how to programmatically add environment variable definitions to an RSL prior to submitting a job. The source to this example\(^3\) can be downloaded.

```c
#include "globus_common.h"
#include "globus_gram_client.h"
#include "globus_rsl.h"
#include "globus_gram_protocol.h"

#include <stdio.h>
#include <strings.h>

static int example_rsl_attribute_match(void * datum, void * arg)
{
    const char * relation_attribute = globus_rsl_relation_get_attribute(datum);
    const char * attribute = arg;
    
    /* RSL attributes are case-insensitive */
    return (relation_attribute &&
            strcasecmp(relation_attribute, attribute) == 0);
}

int main(int argc, char *argv[])
{
    int rc;
    globus_rsl_t *rsl, *environment_relation;
    globus_rsl_value_t *new_env_pair = NULL;
    globus_list_t *environment_relation_node;
    char * rsl_string;
    char * job_contact;

    if (argc != 3)
    {
        fprintf(stderr, "Usage: %s RESOURCE-MANAGER-CONTACT RSL\n", argv[0]);
        rc = 1;
        goto out;
    }

    /*
    * Always activate the GLOBUS_GRAM_CLIENT_MODULE prior to using any
    * functions from the GRAM Client API or behavior is undefined.
    */
```

\(^3\) gram_rsl_example.c
rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Error activating %s because %s (Error %d)\n",
            GLOBUS_GRAM_CLIENT_MODULE->module_name,
            globus_gram_client_error_string(rc),
            rc);
    goto out;
}

/* Parse the RSL string into a syntax tree */
rsl = globus_rsl_parse(argv[2]);
if (rsl == NULL)
{
    rc = 1;
    fprintf(stderr, "Error parsing RSL string\n");
    goto deactivate;
}

/* Create the new environment variable pair that we'll insert */
new_env_pair = globus_rsl_value_make_sequence(NULL);
if (new_env_pair == NULL)
{
    fprintf(stderr, "Error creating value sequence\n");
    rc = 1;
    goto free_rsl;
}

/* Then insert the name-value pair in reverse order */
rc = globus_list_insert(
    globus_rsl_value_sequence_get_list_ref(new_env_pair),
    globus_rsl_value_make_literal(
            strdup("itsvalue")));
if (rc != GLOBUS_SUCCESS)
{
    goto free_env_pair;
}

rc = globus_list_insert(
    globus_rsl_value_sequence_get_list_ref(new_env_pair),
    globus_rsl_value_make_literal(
            strdup("EXAMPLE_ENVIRONMENT_VARIABLE")));
if (rc != GLOBUS_SUCCESS)
{
    goto free_env_pair;
}

/* Now, check to see if the RSL already contains an environment */
environment_relation_node = globus_list_search_pred(
    globus_rsl_boolean_get_operand_list(rsl),
    example_rsl_attribute_match,
GLOBUS_GRAM_PROTOCOL_ENVIRONMENT_PARAM);

if (environment_relation_node == NULL)
{
    /* Not present yet, create a new relation and insert it into
     * the RSL.
     */
    environment_relation = globus_rsl_make_relation(
        GLOBUS_RSL_EQ,
        strdup(GLOBUS_GRAM_PROTOCOL_ENVIRONMENT_PARAM),
        globus_rsl_value_make_sequence(NULL));
    rc = globus_list_insert(
        globus_rsl_boolean_get_operand_list_ref(rsl),
        environment_relation);
    if (rc != GLOBUS_SUCCESS)
    {
        globus_rsl_free_recursive(environment_relation);
        goto free_env_pair;
    }
}
else
{
    /* Pull the environment relation out of the node returned from the
     * search function
     */
    environment_relation = globus_list_first(environment_relation_node);
}
/* Add the new environment binding to the value sequence associated with
 * the environment relation
 */
rc = globus_list_insert(
    globus_rsl_value_sequence_get_list_ref(
        globus_rsl_relation_get_value_sequence(environment_relation)),
    new_env_pair);
if (rc != GLOBUS_SUCCESS)
{
    goto free_env_pair;
}
new_env_pair = NULL;

/* Convert the RSL parse tree to a string */
rsl_string = globus_rsl_unparse(rsl);

/*
 * Submit the augmented RSL to the service passed as our first command-line
 * option. If successful, this function will return GLOBUS_SUCCESS,
 * otherwise an integer error code.
 */
rc = globus_gram_client_job_request(
    argv[1],
    rsl_string,
    0,
    NULL,
&job_contact);
if (rc != GLOBUS_SUCCESS)
{
    fprintf(stderr, "Unable to submit job to %s because %s (Error %d)\n",
            argv[1], globus_gram_client_error_string(rc), rc);
} else
{
    printf("Job submitted successfully: %s\n", job_contact);
}
free(rsl_string);
if (job_contact)
{
    free(job_contact);
}
free_env_pair:
    if (new_env_pair != NULL)
    {
        globus_rsl_value_free_recursive(new_env_pair);
    }
free_rsl:
    globus_rsl_free_recursive(rsl);
deactivate:
    /*
     * Deactivating the module allows it to free memory and close network
     * connections.
     */
    rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
out:
    return rc;
}
/* End of gram_rsl_example.c */
Part III. GRAM Server
Developer's Guide
Table of Contents

5. LRM Adapter Tutorial ................................................................. 36
   1. Introduction .............................................................................. 36
   2. Parts of a GRAM5 LRM Adapter .............................................. 36
   3. RSL Validation File ................................................................ 36
   4. Configuration File ................................................................. 40
   5. LRM Adapter Perl Module ...................................................... 40
   6. LRM SEG Module ................................................................. 46
   7. Changes from Previous Versions ........................................... 59
Chapter 5. LRM Adapter Tutorial

1. Introduction

GRAM5 provides a resource-independent abstraction to remote job management. The resource abstraction contains methods for job submission and cancelling, and a method for monitoring job state changes. This set of tutorials describes how to implement and bundle all packages needed for a complete LRM Adapter interface for GRAM5.

For purposes of this tutorial, we will create a fake LRM adapter that pretends to run jobs, but in fact just keeps track of jobs and expires them after the job's max_wall_time expires. We'll call this LRM the fake LRM adapter.

2. Parts of a GRAM5 LRM Adapter

A GRAM5 LRM Adapter consists of a few parts which work together to provide a full interface between the GRAM5 Job Manager and the Local Resource Manager. These parts include:

- **RSL Validation File**: An option file which defines any custom RSL attributes which the LRM Adapter implements, or sets any custom defaults for RSL attributes that the LRM processes. Defining new RSL attributes in this file allows the GRAM5 service to detect some sets of RSL errors without invoking the Perl LRM Adapter Module. For this example, the file will be called fake.rvf.

- **Perl LRM Adapter Module**: A Perl module which implements the execution interface to the LRM. This module translates the Resource Specification Language description of a job's requirements to a concrete way of starting the job on a particular LRM. For this example, this file will be called fake.pm.

- **Configuration File**: The GRAM5 service implements a simple configuration file parser which can be used to provide a way to add site customizations to LRM Adapters. These files are usually shared between the Perl LRM Adapter Module and the Scheduler Event Generator Module. For this example, this file will be called fake.conf.

- **Gatekeeper Service File**: The Gatekeeper is a privileged service which authenticates and authorizes clients and then starts a Job Manager process on their behalf. The Gatekeeper Service File contains the LRM-specific command-line options to the job manager. For this example, this file will be called jobmanager-fork.

- **Scheduler Event Generator Module**: A dynamic object which parses LRM state and generates job state change events in a generic format for GRAM5 to consume. For this example, the SEG module will be called libglobus_seg_fake.so.

3. RSL Validation File

Each LRM Adapter can have a custom RSL validation file (RVF) which indicates which RSL attributes are valid for that LRM, what their default values are, and when they can be used during a job lifecycle.

The RVF entries consist of a set of records containing attribute-value pairs, with a blank line separating records. Each attribute-value pair is separated by the colon character. The value may be quoted with the double-quote character, in which case, the value continues until a second quote character is found; otherwise, the value terminates at end of line.
3.1. RVF Attributes

The attribute names understood by the GRAM5 RVF parser are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>The name of an RSL attribute.</td>
</tr>
<tr>
<td>Description</td>
<td>A textual description of the attribute.</td>
</tr>
<tr>
<td>RequiredWhen</td>
<td>A sequence of WHEN-VALUES describing when this RSL attribute must be present.</td>
</tr>
<tr>
<td>DefaultWhen</td>
<td>A sequence of WHEN-VALUES describing when the default RSL value will be applied if it's not present in the RSL.</td>
</tr>
<tr>
<td>ValidWhen</td>
<td>A sequence of WHEN-VALUES describing when the RSL attribute may be present.</td>
</tr>
<tr>
<td>Default</td>
<td>A literal RSL value sequence containing the default value of the attribute, applied to the RSL when the attribute is not present, but the RSL use matches the DefaultWhen value.</td>
</tr>
<tr>
<td>Values</td>
<td>A sequence of strings enumerating the legal values for the RSL attribute.</td>
</tr>
<tr>
<td>Publish</td>
<td>When set to true, the RSL attribute will be added to the documentation for the LRM Adapter if the RVF is processed by the create_rsl_documentation.pl script. Otherwise, it will not be mentioned.</td>
</tr>
</tbody>
</table>

3.1.1. RVF When Values

The WHEN-VALUES used by the RVF parser are described in this list:

- **GLOBUS_GRAM_JOB_SUBMIT**: RSL Attribute used in a GRAM5 job request to submit a job to an LRM Adapter.
- **GLOBUS_GRAM_JOB_RESTART**: RSL Attribute used in a GRAM5 job request to restart a job which was stopped due to a two-phase commit timeout.
- **GLOBUS_GRAM_JOB_STDIO_UPDATE**: RSL Attribute used in a GRAM5 STDIO_UPDATE signal, which may be sent to a job during the two-phase end state.

3.1.2. Common RSL Attributes

The GRAM5 service by default implements a common set of RSL attributes for all jobs. Not all of these may be relevant to all LRM types, but are included in the common set so that the same concept will be processed by the same attribute for each LRM. LRM Adapters can disable particular RSL attributes if they want by adding the attribute to their RVF file with

```
Attribute: AttributeName
ValidWhen: ""
```

The common list of attributes is described in Section 4, “RSL Attribute Summary”.

3.2. Creating a RSL Validation File for the Fake LRM

 Normally, the RVF for a new LRM Adapter will add any LRM-specific RSL attributes and perhaps change the DefaultValue for some. For the fake LRM, we'll be a bit more complicated and disable most of the GRAM common
RSL attributes and reduce things to indicate the queue and execution time for the fake jobs. The `fake.rvf` will do the following:

- **Remove** executable, arguments, directory, environment, file_clean_up, file_stage_in, file_stage_out, file_stage_in_shared, gass_cache, gram_my_job, host_count, library_path, max_cpu_time, min_memory, project, queue, remote_io_url, scratch_dir, stdin, stdout, and stderr attributes.

- **Add a** max_queue_time attribute, which will be the maximum time a particular fake job will be in the PENDING state. This will have a default of 20 minutes.

- **Add a default value to the** max_wall_time attribute of 5 minutes.
**Example 5.1. fake.rvf**

Attribute: max_cpu_time
ValidWhen: ""
RequiredWhen: ""

Attribute: min_memory
ValidWhen: ""
RequiredWhen: ""

Attribute: project
ValidWhen: ""
RequiredWhen: ""

Attribute: queue
ValidWhen: ""
RequiredWhen: ""

Attribute: remote_io_url
ValidWhen: ""
RequiredWhen: ""

Attribute: scratch_dir
ValidWhen: ""
RequiredWhen: ""

Attribute: stdin
ValidWhen: ""
RequiredWhen: ""

Attribute: stdout
ValidWhen: ""
RequiredWhen: ""

Attribute: stderr
ValidWhen: ""
RequiredWhen: ""

# Add a new attribute max_queue_time
Attribute: max_queue_time
ValidWhen: GLOBUS_GRAM_JOB_SUBMIT
DefaultWhen: GLOBUS_GRAM_JOB_SUBMIT
RequiredWhen: GLOBUS_GRAM_JOB_SUBMIT
Description: "Maximum time a fake job will be in pending, in seconds. The default value is 1200 seconds (20 minutes)"
Default: 1200

# Add a default value and requirement for max_wall_time
Attribute: max_wall_time
DefaultWhen: GLOBUS_GRAM_JOB_SUBMIT
RequiredWhen: GLOBUS_GRAM_JOB_SUBMIT
Default: 300
Description: "Maximum time a fake job will be in the ACTIVE state"
4. Configuration File

For the fake LRM, there's not much to configure: a path to a file where the LRM should write its job files. For real LRMds, there are other things which might belong there: paths to LRM-specific executables such as `qsub`, tuning parameters for the LRM adapter script such as the number of available cores per execution node, or the host to contact when using a remote submit protocol between GRAM and the LRM. The configuration parameters used by the LRM adapters included in GRAM5 are described in Section 4, “LRM Adapter Configuration”.

The LRM adapter configuration files consist of attribute="value" pairs, which comment lines beginning with the # character. For the example fake LRM, the configuration file looks like this:

```plaintext
# log_path is the path to log file that the fake LRM generates. This file is updated each time a job is submitted or cancelled. The default if it is not present is ${localstatedir}/fake, which is typically /var/fake
log_path="/tmp"
```

4.1. Parsing the Configuration File

The Globus Toolkit contains API functions for parsing files in the format used by the LRM configuration files. In Perl, use the `Globus::Core::Config` class. In C/C++, use the `globus_common_get_attribute_from_config_file()` function.

4.1.1. Perl API

The `Globus::Core::Config` API is quite simple. The `new()` constructor parses the configuration file and returns an object containing the attribute=value pairs. The `get_attribute()` method returns the value of the named attribute. These functions are used in the fake LRM Perl Module.

4.1.2. C/C++ API

The `globus_common_get_attribute_from_config_file()` function will load the configuration file and return the value of the attribute passed to it. This function is used in the SEG module below. Note that this function returns a pointer to a copy of the string value in the location pointed to by the `value` parameter. The caller must free this value.

5. LRM Adapter Perl Module

The Perl-language LRM module provides the job submission and cancelling interface between GRAM5 and the underlying scheduler. Very little has been added to this part of the scheduler interface since Globus Toolkit 2---if you have a version for an older Globus Toolkit release, you can ignore most of this tutorial and jump to the Section 7, “Changes from Previous Versions” section of this tutorial. The module annotated below is available from `fake.pm`.

5.1. Perl LRM Adapter Module

The LRM Adapter interface is implemented as a Perl module which is a subclass of the `Globus::GRAM::JobManager` module. Its name must match the type string used when the job manager is started, but in all lower case: for the fake LRM, the module name is `Globus::GRAM::JobManager::fake` and it is stored in the file `fake.pm`. Though there are several methods in the `Globus::GRAM::JobManager interface`, the only ones which absolutely need to be implemented in a scheduler module are `submit` and `cancel`. The `poll` can be used if there is no SEG module for your LRM Adapter, but using polling can be resource intensive and slow. We'll present the methods in the module one by one, but the entire module can be downloaded from here: `fake.pm`.
We'll begin by looking at the start of the `fake.pm` source module. To begin the script, we import the GRAM support modules into the LRM adapter module's namespace, declare the module's package, and declare this module as a subclass of the `Globus::GRAM::JobManager` module. All LRM adapter packages will need to do this, substituting the name of the LRM type being implemented where we see `fake` below.

```perl
use Globus::GRAM::Error;
use Globus::GRAM::JobState;
use Globus::GRAM::JobManager;
use Globus::Core::Paths;
use Globus::Core::Config;
use File::Path;
use strict;
use warnings;

package Globus::GRAM::JobManager::fake;

our @ISA = ('Globus::GRAM::JobManager');

Next, we declare any system-specific values which will be read from the configuration file. In the fake case, we will declare a module-global directory for job information and for SEG log entries. In real LRM Adapters, there are often variables which are loaded from the configuration file for such things as the list of available queues, paths to executables such as `mpiexec`, and any other site-specific configuration.

```perl
our($job_dir, $fake_seg_dir);
BEGIN
{
    my $config = new Globus::Core::Config( 
        '${sysconfdir}/globus/globus-fake.conf');

    $job_dir = $fake_seg_dir = "";

    if ($config)
    {
        $job_dir = $config->get_attribute("log_path") || "";
    }
    if ($job_dir eq '')
    {
        $job_dir = Globus::Core::Paths::eval_path('${localstatedir}/fake');
    }
}

5.1.1. Writing a Constructor

For LRM Adapter interfaces which need to setup some data before calling their other methods, they can overload the `new` method which acts as a constructor. Scheduler scripts which don't need any per-instance initialization will not need to provide a constructor, the default `Globus::GRAM::JobManager::new` constructor will do the job.

If you do need to overloaded this method, be sure to call the parent module's constructor to allow it to do its initialization. In this example, we create an object which includes a sequence number to ensure that the job ids returned from the LRM script is unique.

```perl
sub new
{
    my $proto = shift;

```
my $class = ref($proto) || $proto;
my $self = $class->SUPER::new(@_);
$self->{sequence} = 0;
return $self;
}

The job interface methods are called with only one argument: the LRM Adapter object itself. That object contains a Globus::GRAM::JobDescription object ($self->{JobDescription}) which includes the values from the RSL associated with the request, as well as a few extra values:

- **job_id** The string returned as the value of JOB_ID in the return hash from submit. This won't be present for methods called before the job is submitted.
- **uniq_id** A string associated with this job request by the job manager program. It will be unique for all jobs on a host for all time and might be useful in creating temporary files or LRM-specific processing.

Now, let's look at the methods which will interface to the LRM.

### 5.1.2. Submitting Jobs

All LRM adapter modules must implement the `submit` method. This method is called when the job manager wishes to submit the job to the LRM. The information in the original job request RSL string is available to the LRM adapter interface through the `JobDescription` data member of its hash.

For most LRM Adapters, this is the longest method to be implemented, as it must decide what to do with the job description, and convert RSL elements to something which the LRM can understand.

For our fake adapter, we will validate that the two RSL attributes we process are integers, and if so generate a new unique LRM ID and return it and the state Globus::GRAM::JobState::PENDING. Note the call to respond with GT3_FAILURE_MESSAGE. This allows the GRAM5 client application to see the context-sensitive error message along with the general failure code from GRAM5.

```perl
sub submit {
    my $self = shift;
    my $description = $self->{JobDescription};
    my $now = time();
    my $jobid;
    my $fh;
    my $pending_time;
    my $active_time;
    my $done_time;
    my $failed_time = 0;

    if ($description->max_wall_time() != int($description->max_wall_time()))
    {
        return Globus::GRAM::Error::INVALID_MAX_WALL_TIME;
    }
    elsif ($description->max_queue_time() != int($description->max_queue_time()))
    {
        $self->respond({GT3_FAILURE_MESSAGE => "Invalid max_queue_time");
        return Globus::GRAM::Error::INVALID_ATTR;
    }

    my $id = int($now);
    $jobid = int($id);
    my $pending = $jobid;
    my $active = $jobid;
    my $done = $jobid;
    my $failed = 0;

    return [Globus::GRAM::JobState::PENDING, $id];
}
```
5.1.3. Polling Job State

GRAM5 requires some way to determine the state of a job. In most systems, writing a Scheduler Event Generator module will provide the best performance and lowest resource overhead. However, when developing an LRM adapter, it is helpful to implement the polling interface so that the submission and cancel mechanism can be tested independent of having the SEG module completed. For the fake LRM Adapter, we'll write a simple poll method which will compare the current time with the time when the job was originally submitted.

```perl
sub poll {
    my $self = shift;
    my $description = $self->{JobDescription};
    my $state;
    my $pid;
    my $now;
    my $fh;
    my $pending_time = 0;
    my $active_time;
    my $done_time;
    my $failed_time;
    my $seqno;
    my $jobid = $description->jobid();

    if(!defined $jobid)
    {
        $self->log("poll: job id undefined!");
        return { JOB_STATE => Globus::GRAM::JobState::FAILED };
    }

    open($fh, "<$job_dir/fakejob.log");

    # Multiple matches might occur if the job is cancelled, so we keep looping
```
# until EOF
while (<$fh>)
{
    chomp;
    my @fields = split(/;/);
    if ($fields[0] ne $jobid)
    {
        next;
    }
    $pending_time = $fields[1];
    $active_time = $fields[2];
    $done_time = $fields[3];
    $failed_time = $fields[4];
}
close($fh);

$now = time();

if ($pending_time == 0)
{
    # not found
    $state = Globus::GRAM::JobState::FAILED;
}
elseif (int($failed_time) != 0)
{
    $state = Globus::GRAM::JobState::FAILED;
}
elseif ($now < int($active_time))
{
    $state = Globus::GRAM::JobState::PENDING;
    return
}
elseif ($now < int($done_time))
{
    $state = Globus::GRAM::JobState::ACTIVE;
}
else
{
    $state = Globus::GRAM::JobState::DONE;
}
return { JOB_STATE => $state };}

5.1.4. Cancelling Jobs

All LRM Adapter modules must also implement the cancel method. The purpose of this method is to cancel a job, whether it's already running or waiting in a queue.

This method will be given the job ID as part of the JobDescription object in the manager object. If the LRM interface provides feedback that the job was cancelled successfully, then we can return a JOB_STATE change to the
FAILED state. Otherwise we can return an empty hash reference, and let either the Scheduler Event Generator or a subsequent call to poll determine when the state change occurs.

For the fake LRM adapter, we will update the job file with a cancellation time and return the Globus::GRAM::JobState::FAILED state change.

```perl
sub cancel {
    my $self = shift;
    my $description = $self->{JobDescription};
    my $pgid;
    my $jobid = $description->jobid();
    my $fh;
    my $pending_time = 0;
    my $active_time;
    my $done_time;
    my $failed_time;
    my $now = time();

    if(!defined $jobid)
    {   $self->log("cancel: no jobid defined!");
        return { JOB_STATE => Globus::GRAM::JobState::FAILED ];
    }

    open($fh, "<$job_dir/fakejob.log");
    
    # Multiple matches might occur if the job is cancelled, so we keep looping
    # until EOF
    while (<$fh>)
    {
        chomp;

        my @fields = split(/;/);

        if ($fields[0] ne $jobid)
        {
            next;
        }

        $pending_time = $fields[1];
        $active_time = $fields[2];
        $done_time = $fields[3];
        $failed_time = $fields[4];
    }
    close($fh);

    $self->log("cancel job ". $jobid);

    if ($now < int($done_time) && int($failed_time) == 0)
    {
        $failed_time = $now;
        $done_time = 0;
        if (!open($fh, ">>$job_dir/fakejob.log"))
        {
            # Handle file opening failure
        }
    }
}
```

45
$self->respond({GT3_FAILURE_MESSAGE => "Unable to write job file"});
return Globus::GRAM::Error::INVALID_SCRIPT_STATUS;
}
print $fh "$jobid;$pending_time;$active_time;$done_time;$failed_time\n";
close($fh);
}
return { JOB_STATE => Globus::GRAM::JobState::FAILED };
}

5.1.5. End of the script

It is required that all perl modules return a non-zero value when they are parsed. To do this, make sure the last line of your module consists of:

1;

6. LRM SEG Module

6.1. Intro

The Scheduler Event Generator (SEG) module provides an efficient job monitoring interface between GRAM5 and the underlying local resource manager. In most cases, the SEG module parses a log generated by the local resource manager which contains information about job state changes and then uses the SEG API\(^1\) to signal job state changes as they occur.

A SEG module is implemented as a shared library which is loaded as a globus extension. This means that the only entry point to the library is a globus_module_descriptor, which defines activation and deactivation functions for the library. For this tutorial, we will build up the SEG module piecewise, but the entire fake SEG module source\(^3\) can be downloaded as well.

6.2. Outline

The outline for our SEG module is:

Include Headers
Module Specific Data
Module Specific Prototypes
Extension Module Descriptor
Module Activation
Module Deactivation
Process Events
Utility Functions

From this outline, we'll explain the various sections of the source file below.

6.3. LRM Module Dependencies

The LRM module uses the globus_common API from Globus for its linked list, mutual exclusion, timed events, and module dependency tracking. It also uses the Scheduler Event Generator APIs, which provide functions for defining and emitting LRM events.

\(^1\) [http://www.globus.org/api/c-globus-5.2.2/globus_scheduler_event_generator/html/group__seg__api.html](http://www.globus.org/api/c-globus-5.2.2/globus_scheduler_event_generator/html/group__seg__api.html)

\(^3\) seg_fake_module.c
Include Headers.  For our implementation, we'll need to include the headers for the Globus modules we'll be using. In this case we'll be using `globus_common.h`, `globus_scheduler_event_generator.h` (which includes the API for emitting SEG events), and `globus_scheduler_event_generator_app.h` (which includes the SEG event type definitions).

```c
#include "globus_common.h"
#include "globus_scheduler_event_generator.h"
#include "globus_scheduler_event_generator_app.h"
```

### 6.4. Module Specific Data

For the fake LRM, we need to keep some global state to keep track of what we've parsed from our LRM's log file, and what events are we should be sending in the future. To do this, we define two data structures, a `fake_job_info_t` which defines the set of event timestamps associated with a job, and `fake_state_t` which contains the state of the fake SEG parser.

#### Fake Job Info

LRM Parser State

**Fake Job Info.** For the `fake_job_info_t` structure, we will want to keep track of the LRM id (an up to 64-character long string), and the timestamps for the pending, active, failed, and done events for the job. We use the timestamp value of 0 to indicate an event which will not happen or has already been processed.

```c
typedef struct
{
    char   jobid[64];
    time_t pending;
    time_t active;
    time_t failed;
    time_t done;
} fake_job_info_t;
```

In addition, we will keep a null initializer for the `fake_job_info_t` structure so that we can simply initialize dynamically allocated data.

```c
/* A statically-initialized empty job info which is used to initialize *
 * dynamically allocated fake_job_info_t structs *
 */
static fake_job_info_t fake_job_info_initializer;
```

#### LRM Parser State

LRM Parser State.  For the LRM parser state, we will keep track of the start time for which we will emit events, the path to the fake job log, a file pointer open to that log, and a list of `fake_job_info_t` structs for each job we have data for. We also use a mutex/condition variable combination to block deactivation until all callback functions have completed. The data in this struct is initialized in the module's activation function below.

```c
/**
 * State of the FAKE log file parser.
 * /
static struct
{
    /** Timestamp of when to start generating events from */
    time_t start_timestamp;
```
/** Log file path */
char *                              log_path;
/** Log file pointer */
FILE *                              log;
/** List of job info containing future info we might need to
* turn into job state changes */
globus_list_t *                     jobs;
/**
* shutdown mutex
*/
globus_mutex_t                      mutex;
/**
* shutdown condition
*/
globus_cond_t                       cond;
/**
* shutdown flag
*/
globus_bool_t                       shutdown_called;
/**
* callback count
*/
int                                 callback_count;
} globus_l_fake_state;

6.5. Module Specific Prototypes

For our SEG, we define a small number of static functions to process the fake job log. These include our activation and deactivation functions, and our event callback which is called periodically to process the fake job log. We also have a couple of utility functions to look up entries in the job list and a predicate used to sort a list of SEG events by timestamp and jobid.

static int
globus_l_fake_module_activate(void);

static int
globus_l_fake_module_deactivate(void);

static void
globus_l_fake_read_callback(void *user_arg);

static int
globus_l_fake_find_by_job_id(void *datum, void *arg);

static int
globus_l_fake_compare_events(void *low_datum, void *high_datum, void *relation_args);
6.6. Extension Module Descriptor

The SEG dynamically loads our code using the Globus Extension API. To implement the interface it needs, we must define an extension descriptor so that it can find the entry point to our library. This module descriptor contains pointers to the activation and deactivation functions we prototyped above. It can contain other pointers but they aren't needed for our module implementation so we leave them as NULL.

```c
GlobusExtensionDefineModule(globus_seg_fake) =
{
    "globus_seg_fake",
    globus_l_fake_module_activate,
    globus_l_fake_module_deactivate,
    NULL,
    NULL,
    NULL
};
```

6.7. Module Activation

The entry point to our LRM-specific module is the activation function. This function is invoked by the `globus-scheduler-event-generator` program when it starts and dynamically loads the LRM-specific module. It is not passed any parameters, and is expected to return GLOBUS_SUCCESS if it is able to activate itself. Typically the activation function will do the following:

```c
static int
globus_l_fake_module_activate(void)
{
    Declare Variables
    Activate Dependencies
    Prepare Shutdown Handler
    Read Configuration
    Register Event
    Cleanup on Failure
    return result;
} /* globus_l_fake_module_activate() */
```

**Declare Variables.** For our activation function, we'll use variables to store the path to the configuration file as well as return values from functions we call.

```c
char * config_path = NULL;
char * log_dir;
int rc;
globus_result_t result = GLOBUS_SUCCESS;
```

**Activate Dependencies.** The headers we've just included contain the module descriptors which we will activate in our LRM-specific activation function, so that we are able to use the APIs in those modules. Our module is only ever activated by the SEG module, so we shouldn't activate it. In the activation function for our module, we'll include this fragment

```c
rc = globus_module_activate(GLOBUS_COMMON_MODULE);
if (rc != GLOBUS_SUCCESS)
```
{  
    fprintf(stderr, "Fatal error activating GLOBUS_COMMON_MODULE\n");  

    result = GLOBUS_FAILURE;  
    goto activation_failure;  
}

**Prepare Shutdown Handler.** To handling deactivation safely, we'll create a mutex and condition variable to handle the case of when a shutdown is called while our event handler is running. In that case, we'll set the `shutdown_called` variable to GLOBUS_TRUE and then wait until the callback has terminated. Here we just set the variables to their non-shutdown values.

```c
rc = globus_mutex_init(&globus_l_fake_state.mutex, NULL);  
if (rc != GLOBUS_SUCCESS)  
{  
    result = GLOBUS_FAILURE;  
    goto mutex_init_failed;  
}

rc = globus_cond_init(&globus_l_fake_state.cond, NULL);  
if (rc != GLOBUS_SUCCESS)  
{  
    result = GLOBUS_FAILURE;  
    goto cond_init_failed;  
}

globus_l_fake_state.shutdown_called = GLOBUS_FALSE;  
globus_l_fake_state.callback_count = 0;
```

### 6.7.1. LRM SEG Module Configuration

**Read Configuration.** There are two main pieces of configuration information we'll need to process SEG events: the earliest timestamp we care about (which we get from the SEG module) and the path to our fake job log file (which we get from our configuration file as in the perl module).

So first, to get the timestamp, we'll use the `globus_scheduler_event_generator_get_timestamp()` function.

```c
result = globus_scheduler_event_generator_get_timestamp(
    &globus_l_fake_state.start_timestamp);  
if (result != GLOBUS_SUCCESS)  
{  
    goto get_timestamp_failed;  
}
```

Then, to get the configuration file data, we first construct the path to the configuration file and then pull out the configuration attribute log_path, falling back to the default (`${localstatedir}/fake` if it is not found).

```c
result = globus_eval_path(
    "${sysconfdir}/globus/globus-fake.conf",
    &config_path);  
if (result != GLOBUS_SUCCESS || config_path == NULL)  
{  
    goto get_config_path_failed;  
}  
```
result = globus_common_get_attribute_from_config_file("", config_path, "log_path", &log_dir);

/* This default must match fake.pm's default for things to work */
if (result != GLOBUS_SUCCESS)
{
    result = globus_eval_path("${localstatedir}/fake", &log_dir);
}

if (result != GLOBUS_SUCCESS)
{
    goto get_log_dir_failed;
}

globus_l_fake_state.log_path = globus_common_create_string("%s/fakejob.log", log_dir);
if (globus_l_fake_state.log_path == NULL)
{
    result = GLOBUS_FAILURE;
    goto get_log_path_failed;
}

6.7.2. Register Event

The next main action the activation function does is to register an event to happen later to process the events in the
LRM log. For this, we use the globus_callback_register_oneshot() function to register an event han-
dler to execute as soon as possible within the globus-scheduduler-event-generator program. The callback function
in this case is the globus_l_fake_read_callback() function defined later.

result = globus_callback_register_oneshot(NULL, NULL, globus_l_fake_read_callback, &globus_l_fake_state);
if (result != GLOBUS_SUCCESS)
{
    goto register_oneshot_failed;
}
globus_l_fake_state.callback_count++;

6.7.3. Cleanup on Failure

Here we handle the errors that might have occurred above and free temporarily used memory. In case of a failure,
result is set to something other than GLOBUS_SUCCESS.

register_oneshot_failed:
get_log_path_failed:
    if (result != GLOBUS_SUCCESS)
    {
        free(globus_l_fake_state.log_path);
free(log_dir);
get_log_dir_failed:
    free(config_path);
get_config_path_failed:
get_timestamp_failed:
    if (result != GLOBUS_SUCCESS)
        {
            malloc_state_failed:
                globus_cond_destroy(&globus_l_fake_state.cond);
            cond_init_failed:
                globus_mutex_destroy(&globus_l_fake_state.mutex);
            mutex_init_failed:
                globus_module_deactivate(GLOBUS_COMMON_MODULE);
        }
activation_failure:

6.8. Module Deactivation

For our deactivation function, we will wait use the shutdown handling variables in the state structure to wait until all outstanding callback have terminated and then free memory associated with the state.

static
int
globus_l_fake_module_deactivate(void)
{
    Shutdown Handling
    Cleanup State
} /* globus_l_fake_module_deactivate() */

Shutdown Handling. To handle shutdown safely, we must wait until pending callbacks have terminated. To do this, we set the shutdown-called field in the state structure and wait until the callback_count field is 0. Inside the callback function, if we see that the shutdown-called field is GLOBUS_TRUE then it will not reregister itself and will signal when it terminates.

globus_mutex_lock(&globus_l_fake_state.mutex);
globus_l_fake_state.shutdown_called = GLOBUS_TRUE;
while (globus_l_fake_state.callback_count > 0)
{
    globus_cond_wait(&globus_l_fake_state.cond, &globus_l_fake_state.mutex);
}
globus_mutex_unlock(&globus_l_fake_state.mutex);

Cleanup State. Finally, we'll free data we allocated in the activation function.

globus_mutex_destroy(&globus_l_fake_state.mutex);
globus_cond_destroy(&globus_l_fake_state.cond);
if (globus_l_fake_state.log_path)
    {
        free(globus_l_fake_state.log_path);
    }
if (globus_l_fake_state.log)
    {
        fclose(globus_l_fake_state.log);
    }
while (!globus_list_empty(globus_l_fake_state.jobs))
{
fake_job_info_t *info;

info = globus_list_remove(
    &globus_l_fake_state.jobs,
    globus_l_fake_state.jobs);
free(info);
}
globus_module_deactivate(GLOBUS_COMMON_MODULE);

return GLOBUS_SUCCESS;

6.9. Process Events

The main activity of our LRM module is to generate SEG events so that a job manager will be able to efficient manage its jobs. In this code, we will parse our log file periodically, and fire off any events which are to have occurred for the jobs in the fake job log. The structure of the processing function is this

static
void
globus_l_fake_read_callback(void * arg)
{
Declarate Variables
Check for Shutdown
Open Log
Read Log
Create Events
Emit Events
Reregister Callback
Error Handling
} /* globus_l_fake_read_callback() */

Declare Variables.

char                                jobid[64];
globus_list_t                       *l, *events;
fake_job_info_t                     *info;
globus_reltime_t                    delay = {0};
time_t                              now;
unsigned long                       pending_time, active_time, done_time, failed_time;
globus_scheduler_event_t            *e;
time_t                              last_timestamp;
globus_result_t                     result = GLOBUS_SUCCESS;

Check for Shutdown. To check for shutdown, we'll first lock the mutex associated with the state structure and check if the shutdown_called field is set to true. If so, we'll jump to our error handling code.

globus_mutex_lock(&globus_l_fake_state.mutex);
if (globus_l_fake_state.shutdown_called)
{
result = GLOBUS_FAILURE;
goto error;
}

Open Log. In general, we'll keep a file open to parse the log, but the first time around, or before any events have been written, the log file might not exist. So we'll check to see if we have a NULL file pointer, and if so, try to open the file. Once opened, we'll use line buffering while we process the file.

if (globus_l_fake_state.log == NULL)
{
    globus_l_fake_state.log = fopen(globus_l_fake_state.log_path, "r");
    if (globus_l_fake_state.log != NULL)
    {
        /* Enable line buffering */
        setvbuf(globus_l_fake_state.log, NULL, _IOLBF, 0);
    }
}
if (globus_l_fake_state.log == NULL)
{
    result = GLOBUS_FAILURE;
    GlobusTimeReltimeSet(delay, 30, 0);
goto reregister;
}

Read Log. Now we will read all of the log entries from our current position until the end of file. If we've already parsed an entry for a particular job, we will zero out its timestamps and replace with the new timestamps to handle cancel events in the fake job log.

/* previous read might have hit EOF, so clear it before trying to read */
clearerr(globus_l_fake_state.log);

/* Read any new job info from the log */
while (fscanf(globus_l_fake_state.log, "%[^;];%ld;%ld;%ld;%ld\n", 
    jobid, 
    &pending_time, 
    &active_time, 
    &done_time, 
    &failed_time) == 5)
{
    l = globus_list_search_pred(globus_l_fake_state.jobs, globus_l_fake_find_by_job_id, jobid);
    if (l != NULL)
    {
        info = globus_list_first(l);
        /* If there's a second entry for the same job, it was cancelled, so 
         * clear done/failed timestamps and copy them below 
         */
        info->done = info->failed = 0;
    }
    else
    {
        /* First time we've seen this job, set jobid and insert*/
        info = malloc(sizeof(fake_job_info_t));
        *info = fake_job_info_initializer;
        strcpy(info->jobid, jobid);
Create Events. Now, we'll walk our list of jobs and create SEG events for each state transition which has occurred since our last timestamp and the current time. These events will be out of order in our events list, because they are created in order of job IDs in the jobs list, and not in timestamp list. We'll deal with this later. Note that we set the timestamp values in the job info to 0 after we create an event. This keeps us from generating an event multiple times.

```c
now = time(NULL);

events = NULL;
for (l = globus_l_fake_state.jobs; l != NULL; l = globus_list_rest(l))
{
    info = globus_list_first(l);
    if (info->pending >= globus_l_fake_state.start_timestamp &&
        info->pending < now)
    {
        e = malloc(sizeof(globus_scheduler_event_t));
        e->event_type = GLOBUS_SCHEDULER_EVENT_PENDING;
        e->job_id = info->jobid;
        e->timestamp = info->pending;
        e->exit_code = 0;
        e->failure_code = 0;
        e->raw_event = NULL;

        info->pending = 0;

        globus_list_insert(&events, e);
    }
    if (info->active >= globus_l_fake_state.start_timestamp &&
        info->active < now)
    {
        e = malloc(sizeof(globus_scheduler_event_t));
        e->event_type = GLOBUS_SCHEDULER_EVENT_ACTIVE;
        e->job_id = info->jobid;
        e->timestamp = info->active;
        e->exit_code = 0;
        e->failure_code = 0;
        e->raw_event = NULL;

        info->active = 0;

        globus_list_insert(&events, e);
    }
}
```
if (info->done != 0 &&
    info->done >= globus_l_fake_state.start_timestamp &&
    info->done < now)
{
    e = malloc(sizeof(globus_scheduler_event_t));
    e->event_type = GLOBUS_SCHEDULER_EVENT_DONE;
    e->job_id = info->jobid;
    e->timestamp = info->done;
    e->exit_code = 0;
    e->failure_code = 0;
    e->raw_event = NULL;

    info->done = 0;
    globus_list_insert(&events, e);
}

if (info->failed != 0 &&
    info->failed >= globus_l_fake_state.start_timestamp &&
    info->failed < now)
{
    e = malloc(sizeof(globus_scheduler_event_t));
    e->event_type = GLOBUS_SCHEDULER_EVENT_FAILED;
    e->job_id = info->jobid;
    e->timestamp = info->failed;
    e->exit_code = 0;
    e->failure_code = GLOBUS_GRAM_PROTOCOL_ERROR_USER_CANCELLED;
    e->raw_event = NULL;

    info->failed = 0;
    globus_list_insert(&events, e);
}

Emit Events. Now we have a set of events, we will sort them by timestamp and then use the SEG API to emit them. After we’ve emitted an event, we have to free it. If the event is a terminal one (DONE or FAILED) we’ll remove the job from the list of jobs in the state structure.

/* Sort the events so that they're in timestamp order */
events = globus_list_sort_destructive(events, globus_l_fake_compare_events, NULL);

/* Emit events in proper order */
while (! globus_list_empty(events))
{
    e = globus_list_remove(&events, events);
    last_timestamp = e->timestamp;

    switch (e->event_type)
    {
    case GLOBUS_SCHEDULER_EVENT_PENDING:
        globus_scheduler_event_pending(e->timestamp, e->job_id);
        break;
    case GLOBUS_SCHEDULER_EVENT_ACTIVE:
LRM Adapter Tutorial

globus_scheduler_event_active(e->timestamp, e->job_id);
break;
case GLOBUS_SCHEDULER_EVENT_FAILED:
globus_scheduler_event_failed(e->timestamp, e->job_id, e->failure_code);
break;
case GLOBUS_SCHEDULER_EVENT_DONE:
globus_scheduler_event_done(e->timestamp, e->job_id, e->exit_code);
break;

}
/* If this is a terminal event, we can remove the job from the list */
if (e->event_type == GLOBUS_SCHEDULER_EVENT_FAILED ||
e->event_type == GLOBUS_SCHEDULER_EVENT_DONE)
{
l = globus_list_search_pred(globus_l_fake_state.jobs, globus_l_fake_find_by_job_id,
info = globus_list_remove(&globus_l_fake_state.jobs, l);
free(info);
}
free(e);
}
globus_l_fake_state.start_timestamp = last_timestamp;
Reregister Callback.
We'll register a new callback instance now (provided we haven't had an error occur) so that
we can continue to process jobs later.
GlobusTimeReltimeSet(delay, 1, 0);
reregister:
result = globus_callback_register_oneshot(
NULL,
&delay,
globus_l_fake_read_callback,
&globus_l_fake_state);
if (result != GLOBUS_SUCCESS)
{
goto error;
}
globus_mutex_unlock(&globus_l_fake_state.mutex);
return;
Error Handling.
If an error occurred registering the event or the shutdown handler is invoked, we'll exit this
function without registering a new event. In the case the shutdown handling is in place, we'll signal that as well
error:
if (globus_l_fake_state.shutdown_called)
{
globus_l_fake_state.callback_count--;
if (globus_l_fake_state.callback_count == 0)
{
globus_cond_signal(&globus_l_fake_state.cond);
}
}
else
{
fprintf(stderr,

57


"FATAL: Unable to register callback. FAKE SEG exiting\n"; 
    exit(EXIT_FAILURE); 
} 
globus_mutex_unlock(&globus_l_fake_state.mutex); 
return; 

6.10. Utility Functions

We have two utility functions to implement for this module to manage our lists of pending events and jobs.

Find By Job ID

Sort Events

Find By Job ID. The globus_l_fake_find_by_job_id() function is used to search the jobs field of the state structure for a fake_job_info_t containing info about a particular job. This predicate returns a non-zero value if the datum passed to the function has the same job ID as the arg parameter.

static 
int 
globus_l_fake_find_by_job_id(void * datum, void * arg) 
{ 
    fake_job_info_t * info = datum; 
    
    return (strcmp(info->jobid, arg) == 0); 
} /* globus_l_fake_find_by_job_id() */

Sort Events. The globus_l_fake_compare_events() function is used as a predicate to compare the timestamps and job ids of a pair of events. If the log_datum points to an event which happens earlier in the job lifecycle than the high_datum, this function returns GLOBUS_TRUE; otherwise it returns GLOBUS_FALSE.

static 
int 
globus_l_fake_compare_events(void * low_datum, void * high_datum, void * relation_args) 
{ 
    globus_scheduler_event_t *low_event = low_datum, *high_event = high_datum; 

    if (low_event->timestamp < high_event->timestamp) 
    { 
        return GLOBUS_TRUE; 
    } 
    else if (low_event->timestamp == high_event->timestamp) 
    { 
        if (low_event->event_type == GLOBUS_SCHEDULER_EVENT_PENDING) 
        { 
            return GLOBUS_TRUE; 
        } 
        else if (low_event->event_type == GLOBUS_SCHEDULER_EVENT_ACTIVE && 
                 high_event->event_type != GLOBUS_SCHEDULER_EVENT_PENDING) 
        { 
            return GLOBUS_TRUE; 
        } 
        else if (high_event->event_type != GLOBUS_SCHEDULER_EVENT_PENDING &&
high_event->event_type != GLOBUS_SCHEDULER_EVENT_ACTIVE) {
    return GLOBUS_TRUE;
}
} /* globus_l_fake_compare_events() */

7. Changes from Previous Versions

7.1. Changes in GT 5.2

GRAM5 is now designed to work as a native debian or RPM package, with default configuration being done at configuration time, so the setup script description has been removed.

7.2. Changes in GT 5.0

GRAM5 is based again on the C code base used for GRAM2 (also known as Pre-WS GRAM). The SEG module interface from GRAM4 is retained and optionally used by GRAM5. The GRAM job manager will avoid reloading the GRAM LRM Adapter script for each operation, so all variables not intended to be global state in the Perl LRM Adapter module must be lexically scoped, or state will leak between jobs and cause potentially cause problems.

7.3. Changes in GT 4.0

7.3.1. Module Methods

The GT-4.0 ws-GRAM service only calls a subset of the Perl methods which were used by the pre-ws GRAM services. Most importantly for script implementors, the polling method is no longer used. Instead, the scheduler-event-generator monitors jobs to signal the service when job change changes occur. Staging is now done via the Reliable File Transfer service, so the file_stage_in and file_stage_out methods are no longer called. Schedulers typically did not implement the staging methods, so this shouldn't affect most scheduler modules.

That being said, scheduler implementers which would like to have their scheduler both with pre-ws GRAM and WS-GRAM should definitely implement the poll() method described in the pre-WS version of this tutorial.

7.3.2. GASS Cache

The GT-4.0 ws-GRAM service does not use the GASS cache for storing temporary files or for staging files.

7.4. Changes in GT 3.2

In GT 3.2, additional error message context info was added. Scripts can optionally add one of these fields to the return hash from an operation to provide extra error information to the client:

- **GT3_FAILURE_MESSAGE**  Error message from underlying script processing indicating what caused a job request to fail
- **GT3_FAILURE_TYPE**  One of filestagein, filestageout, filestageinshared, executable, or stdin indicating what job request element caused a staging fault.
- **GT3_FAILURE_SOURCE**  Source URL or file for a failed staging operation
GT3_FAILURE_DESTINATION  Destination URL or file for a failed staging operation
Part IV. GRAM5 Developer's Reference
# Table of Contents

6. APIs ............................................................................................................................................. 63  
   1. Programming Model Overview ................................................................................................. 63  
7. RSL Specification v1.1 .................................................................................................................. 83  
   1. RSL Syntax Overview ................................................................................................................- 83  
   2. RSL Tokenization Overview .................................................................................................... 84  
   3. RSL Substitution Semantics ..................................................................................................... 85  
   4. RSL Attribute Summary ........................................................................................................... 85  
   5. Simple RSL Examples ............................................................................................................... 89  
   6. RSL grammar and tokenization rules ....................................................................................... 90  
8. Debugging ................................................................................................................................... 92  
   1. Basic Debugging Methods ........................................................................................................ 92  
   2. Advanced Debugging Methods ............................................................................................... 93  
9. Troubleshooting .......................................................................................................................... 94  
   1. GRAM Client Troubleshooting ............................................................................................... 94  
   2. Admin Troubleshooting .......................................................................................................... 97  
   3. Errors ...................................................................................................................................... 99  
10. Semantics and syntax of protocols ............................................................................................ 113  
    1. GRAM5 Protocol .................................................................................................................... 113
Chapter 6. APIs

1. Programming Model Overview

1.1. C API Documentation Links

Table 6.1. GRAM Client APIs

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAM Protocol(^1)</td>
<td>Low-level functions for processing GRAM protocol messages. Symbolic constants for RSL attributes, signals, and job states.</td>
</tr>
<tr>
<td>GRAM Client(^1)</td>
<td>Functions for submitting job requests, sending signals, and listening for job state updates.</td>
</tr>
<tr>
<td>RSL (^1)</td>
<td>Functions for parsing and manipulating job specifications in the RSL language.</td>
</tr>
<tr>
<td>Scheduler Event Generator(^1)</td>
<td>Functions for generating and parsing LRM-independent job state change events.</td>
</tr>
</tbody>
</table>

1.2. GRAM5 Perl API Reference

\(^1\) [http://www.globus.org/api/c-globus-5.2.2/globus_gram_protocol/html/](http://www.globus.org/api/c-globus-5.2.2/globus_gram_protocol/html/)
\(^1\) [http://www.globus.org/api/c-globus-5.2.2/globus_gram_client/html/](http://www.globus.org/api/c-globus-5.2.2/globus_gram_client/html/)
\(^1\) [http://www.globus.org/api/c-globus-5.2.2/globus_rsl/html/](http://www.globus.org/api/c-globus-5.2.2/globus_rsl/html/)
\(^1\) [http://www.globus.org/api/c-globus-5.2.2/globus_scheduler_event_generator/html/](http://www.globus.org/api/c-globus-5.2.2/globus_scheduler_event_generator/html/)
Name
Globus::GRAM::Error — GRAM Protocol Error Constants

DESCRIPTION

The Globus::GRAM::Error module defines symbolic names for the Error constants in the GRAM Protocol.

The Globus::GRAM::Error module methods return an object consisting of an integer error code, and (optionally) a string explaining the error.

Methods

$error = new Globus::GRAM::Error($number, $string);
Create a new error object with the given error number and string description.
This is called by the error-specific factory methods described below.

$error->string()
Return the error string associated with a Globus::GRAM::Error object.

$error->value()
Return the integer error code associated with a Globus::GRAM::Error object.

$error = Globus::GRAM::Error::PARAMETER_NOT_SUPPORTED()
Create a new PARAMETER_NOT_SUPPORTED GRAM error.

$error = Globus::GRAM::Error::INVALID_REQUEST()
Create a new INVALID_REQUEST GRAM error.

$error = Globus::GRAM::Error::NO_RESOURCES()
Create a new NO_RESOURCES GRAM error.

$error = Globus::GRAM::Error::BAD_DIRECTORY()
Create a new BAD_DIRECTORY GRAM error.

$error = Globus::GRAM::Error::EXECUTABLE_NOT_FOUND()
Create a new EXECUTABLE_NOT_FOUND GRAM error.

$error = Globus::GRAM::Error::INSUFFICIENT_FUNDS()
Create a new INSUFFICIENT_FUNDS GRAM error.

$error = Globus::GRAM::Error::AUTHORIZATION()
Create a new AUTHORIZATION GRAM error.

$error = Globus::GRAM::Error::USER_CANCELLED()
Create a new USER_CANCELLED GRAM error.

$error = Globus::GRAM::Error::SYSTEM_CANCELLED()
Create a new SYSTEM_CANCELLED GRAM error.

$error = Globus::GRAM::Error::PROTOCOL_FAILED()
Create a new PROTOCOL_FAILED GRAM error.

$error = Globus::GRAM::Error::STDIN_NOT_FOUND()
Create a new STDIN_NOT_FOUND GRAM error.

$error = Globus::GRAM::Error::CONNECTION_FAILED()
Create a new CONNECTION_FAILED GRAM error.
$error = Create a new INVALID_MAXTIME GRAM error.
Globus::GRAM::Error::INVALID_MAXTIME()

$error = Create a new INVALID_COUNT GRAM error.
Globus::GRAM::Error::INVALID_COUNT()

$error = Create a new NULL_SPECIFICATION_TREE GRAM error.
Globus::GRAM::Error::NULL_SPECIFICATION_TREE()

$error = Create a new JM_FAILED_ALLOW_ATTACH GRAM error.
Globus::GRAM::Error::JM_FAILED_ALLOW_ATTACH()

$error = Create a new JOB_EXECUTION_FAILED GRAM error.
Globus::GRAM::Error::JOB_EXECUTION_FAILED()

$error = Create a new INVALID_PARADYN GRAM error.
Globus::GRAM::Error::INVALID_PARADYN()

$error = Create a new INVALID_JOBTYPE GRAM error.
Globus::GRAM::Error::INVALID_JOBTYPE()

$error = Create a new INVALID_GRAM_MYJOB GRAM error.
Globus::GRAM::Error::INVALID_GRAM_MYJOB()

$error = Create a new BAD_SCRIPT_ARG_FILE GRAM error.
Globus::GRAM::Error::BAD_SCRIPT_ARG_FILE()

$error = Create a new ARG_FILE_CREATION_FAILED GRAM error.
Globus::GRAM::Error::ARG_FILE_CREATION_FAILED()

$error = Create a new INVALID_JOBSTATE GRAM error.
Globus::GRAM::Error::INVALID_JOBSTATE()

$error = Create a new INVALID_SCRIPT_REPLY GRAM error.
Globus::GRAM::Error::INVALID_SCRIPT_REPLY()

$error = Create a new INVALID_SCRIPT_STATUS GRAM error.
Globus::GRAM::Error::INVALID_SCRIPT_STATUS()

$error = Create a new JOBTYPE_NOT_SUPPORTED GRAM error.
Globus::GRAM::Error::JOBTYPE_NOT_SUPPORTED()

$error = Create a new UNIMPLEMENTED GRAM error.
Globus::GRAM::Error::UNIMPLEMENTED()

$error = Create a new TEMP_SCRIPT_FILE_FAILED GRAM error.
Globus::GRAM::Error::TEMP_SCRIPT_FILE_FAILED()

$error = Create a new USER_PROXY_NOT_FOUND GRAM error.
Globus::GRAM::Error::USER_PROXY_NOT_FOUND()

$error = Create a new OPENING_USER_PROXY GRAM error.
Globus::GRAM::Error::OPENING_USER_PROXY()

$error = Create a new JOB_CANCEL_FAILED GRAM error.
Globus::GRAM::Error::JOB_CANCEL_FAILED()
$error = Create a new MALLOC_FAILED GRAM error.
Globus::GRAM::Error::MALLOC_FAILED()

$error = Create a new DUCT_INIT_FAILED GRAM error.
Globus::GRAM::Error::DUCT_INIT_FAILED()

$error = Create a new DUCT_LSP_FAILED GRAM error.
Globus::GRAM::Error::DUCT_LSP_FAILED()

$error = Create a new INVALID_HOST_COUNT GRAM error.
Globus::GRAM::Error::INVALID_HOST_COUNT()

$error = Create a new UNSUPPORTED_PARAMETER GRAM error.
Globus::GRAM::Error::UNSUPPORTED_PARAMETER()

$error = Create a new INVALID_QUEUE GRAM error.
Globus::GRAM::Error::INVALID_QUEUE()

$error = Create a new INVALID_PROJECT GRAM error.
Globus::GRAM::Error::INVALID_PROJECT()

$error = Create a new RSL_EVALUATION_FAILED GRAM error.
Globus::GRAM::Error::RSL_EVALUATION_FAILED()

$error = Create a new BAD_RSL_ENVIRONMENT GRAM error.
Globus::GRAM::Error::BAD_RSL_ENVIRONMENT()

$error = Create a new DRYRUN GRAM error.
Globus::GRAM::Error::DRYRUN()

$error = Create a new ZERO_LENGTH_RSL GRAM error.
Globus::GRAM::Error::ZERO_LENGTH_RSL()

$error = Create a new STAGING_EXECUTABLE GRAM error.
Globus::GRAM::Error::STAGING_EXECUTABLE()

$error = Create a new STAGING_STDIN GRAM error.
Globus::GRAM::Error::STAGING_STDIN()

$error = Create a new INVALID_JOB_MANAGER_TYPE GRAM error.
Globus::GRAM::Error::INVALID_JOB_MANAGER_TYPE()

$error = Create a new BAD_ARGUMENTS GRAM error.
Globus::GRAM::Error::BAD_ARGUMENTS()

$error = Create a new GATEKEEPER_MISCONFIGURED GRAM error.
Globus::GRAM::Error::GATEKEEPER_MISCONFIGURED()

$error = Create a new BAD_RSL GRAM error.
Globus::GRAM::Error::BAD_RSL()

$error = Create a new VERSION_MISMATCH GRAM error.
Globus::GRAM::Error::VERSION_MISMATCH()

$error = Create a new RSL_ARGUMENTS GRAM error.
Globus::GRAM::Error::RSL_ARGUMENTS()
$error = Globus::GRAM::Error::RSL_COUNT() Create a new RSL_COUNT GRAM error.

$error = Globus::GRAM::Error::RSL_DIRECTORY() Create a new RSL_DIRECTORY GRAM error.

$error = Globus::GRAM::Error::RSL_DRYRUN() Create a new RSL_DRYRUN GRAM error.

$error = Globus::GRAM::Error::RSL_ENVIRONMENT() Create a new RSL_ENVIRONMENT GRAM error.

$error = Globus::GRAM::Error::RSL_EXECUTABLE() Create a new RSL_EXECUTABLE GRAM error.

$error = Globus::GRAM::Error::RSL_HOST_COUNT() Create a new RSL_HOST_COUNT GRAM error.

$error = Globus::GRAM::Error::RSL_JOBTYPE() Create a new RSL_JOBTYPE GRAM error.

$error = Globus::GRAM::Error::RSL_MAXTIME() Create a new RSL_MAXTIME GRAM error.

$error = Globus::GRAM::Error::RSL_MYJOB() Create a new RSL_MYJOB GRAM error.

$error = Globus::GRAM::Error::RSL_PARADYN() Create a new RSL_PARADYN GRAM error.

$error = Globus::GRAM::Error::RSL_PROJECT() Create a new RSL_PROJECT GRAM error.

$error = Globus::GRAM::Error::RSL_QUEUE() Create a new RSL_QUEUE GRAM error.

$error = Globus::GRAM::Error::RSL_STDERR() Create a new RSL_STDERR GRAM error.

$error = Globus::GRAM::Error::RSL_STDIN() Create a new RSL_STDIN GRAM error.

$error = Globus::GRAM::Error::RSL_STDOUT() Create a new RSL_STDOUT GRAM error.

$error = Globus::GRAM::Error::OPENING_JOBMANAGER_SCRIPT() Create a new OPENING_JOBMANAGER_SCRIPT GRAM error.

$error = Globus::GRAM::Error::CREATINGPIPE() Create a new CREATING_PIPE GRAM error.

$error = Globus::GRAM::Error::FCNTL_FAILED() Create a new FCNTL_FAILED GRAM error.

$error = Globus::GRAM::Error::STDOUT_FILENAME_FAILED() Create a new STDOUT_FILENAME_FAILED GRAM error.
$error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = $error = Create a new STDERR_FILENAME_FAILED GRAM error. Globus::GRAM::Error::STDERR_FILENAME_FAILED()

Create a new FORKING_EXECUTABLE GRAM error. Globus::GRAM::Error::FORKING_EXECUTABLE()

Create a new EXECUTABLE_PERMISSIONS GRAM error. Globus::GRAM::Error::EXECUTABLE_PERMISSIONS()

Create a new OPENING_STDOUT GRAM error. Globus::GRAM::Error::OPENING_STDOUT()

Create a new OPENING_STDERR GRAM error. Globus::GRAM::Error::OPENING_STDERR()

Create a new OPENING_CACHE_USER_PROXY GRAM error. Globus::GRAM::Error::OPENING_CACHE_USER_PROXY()

Create a new OPENING_CACHE GRAM error. Globus::GRAM::Error::OPENING_CACHE()

Create a new INSERTING_CLIENT_CONTACT GRAM error. Globus::GRAM::Error::INSERTING_CLIENT_CONTACT()

Create a new CLIENT_CONTACT_NOT_FOUND GRAM error. Globus::GRAM::Error::CLIENT_CONTACT_NOT_FOUND()

Create a new CONTACTING_JOB_MANAGER GRAM error. Globus::GRAM::Error::CONTACTING_JOB_MANAGER()

Create a new INVALID_JOB_CONTACT GRAM error. Globus::GRAM::Error::INVALID_JOB_CONTACT()

Create a new UNDEFINED_EXE GRAM error. Globus::GRAM::Error::UNDEFINED_EXE()

Create a new CONDOR_ARCH GRAM error. Globus::GRAM::Error::CONDOR_ARCH()

Create a new CONDOR_OS GRAM error. Globus::GRAM::Error::CONDOR_OS()

Create a new RSL_MIN_MEMORY GRAM error. Globus::GRAM::Error::RSL_MIN_MEMORY()

Create a new RSL_MAX_MEMORY GRAM error. Globus::GRAM::Error::RSL_MAX_MEMORY()

Create a new INVALID_MIN_MEMORY GRAM error. Globus::GRAM::Error::INVALID_MIN_MEMORY()

Create a new INVALID_MAX_MEMORY GRAM error. Globus::GRAM::Error::INVALID_MAX_MEMORY()

Create a new HTTP_FRAME_FAILED GRAM error. Globus::GRAM::Error::HTTP_FRAME_FAILED()
APIs

$error = GD::GRAM::Error::HTTP_UNFRAME_FAILED() Create a new HTTP_UNFRAME_FAILED GRAM error. 

$error = GD::GRAM::Error::HTTP_PACK_FAILED() Create a new HTTP_PACK_FAILED GRAM error. 

$error = GD::GRAM::Error::HTTP_UNPACK_FAILED() Create a new HTTP_UNPACK_FAILED GRAM error. 

$error = GD::GRAM::Error::INVALID_JOB_QUERY() Create a new INVALID_JOB_QUERY GRAM error. 

$error = GD::GRAM::Error::SERVICE_NOT_FOUND() Create a new SERVICE_NOT_FOUND GRAM error. 

$error = GD::GRAM::Error::JOB_QUERY_DENIAL() Create a new JOB_QUERY_DENIAL GRAM error. 

$error = GD::GRAM::Error::CALLBACK_NOT_FOUND() Create a new CALLBACK_NOT_FOUND GRAM error. 

$error = GD::GRAM::Error::BAD_GATEKEEPER_CONTACT() Create a new BAD_GATEKEEPER_CONTACT GRAM error. 

$error = GD::GRAM::Error::POE_NOT_FOUND() Create a new POE_NOT_FOUND GRAM error. 

$error = GD::GRAM::Error::MPIRUN_NOT_FOUND() Create a new MPIRUN_NOT_FOUND GRAM error. 

$error = GD::GRAM::Error::RSL_START_TIME() Create a new RSL_START_TIME GRAM error. 

$error = GD::GRAM::Error::RSL_RESERVATION_HANDLE() Create a new RSL_RESERVATION_HANDLE GRAM error. 

$error = GD::GRAM::Error::RSL_MAX_WALL_TIME() Create a new RSL_MAX_WALL_TIME GRAM error. 

$error = GD::GRAM::Error::INVALID_MAX_WALL_TIME() Create a new INVALID_MAX_WALL_TIME GRAM error. 

$error = GD::GRAM::Error::RSL_MAX_CPU_TIME() Create a new RSL_MAX_CPU_TIME GRAM error. 

$error = GD::GRAM::Error::INVALID_MAX_CPU_TIME() Create a new INVALID_MAX_CPU_TIME GRAM error. 

$error = GD::GRAM::Error::JM_SCRIPT_NOT_FOUND() Create a new JM_SCRIPT_NOT_FOUND GRAM error. 

$error = GD::GRAM::Error::JM_SCRIPT_PERMISSIONS() Create a new JM_SCRIPT_PERMISSIONS GRAM error. 

$error = GD::GRAM::Error::SIGNALING_JOB() Create a new SIGNALING_JOB GRAM error.
$error = Globus::GRAM::Error::UNKNOWN_SIGNAL_TYPE() Create a new UNKNOWN_SIGNAL_TYPE GRAM error.

$error = Globus::GRAM::Error::GETTING_JOBID() Create a new GETTING_JOBID GRAM error.

$error = Globus::GRAM::Error::WAITING_FOR_COMMIT() Create a new WAITING_FOR_COMMIT GRAM error.

$error = Globus::GRAM::Error::COMMIT_TIMED_OUT() Create a new COMMIT_TIMED_OUT GRAM error.

$error = Globus::GRAM::Error::RSL_SAVE_STATE() Create a new RSL_SAVE_STATE GRAM error.

$error = Globus::GRAM::Error::RSL_RESTART() Create a new RSL_RESTART GRAM error.

$error = Globus::GRAM::Error::RSL_TWO_PHASE_COMMIT() Create a new RSL_TWO_PHASE_COMMIT GRAM error.

$error = Globus::GRAM::Error::INVALID_TWO_PHASE_COMMIT() Create a new INVALID_TWO_PHASE_COMMIT GRAM error.

$error = Globus::GRAM::Error::RSL_STDOUT_POSITION() Create a new RSL_STDOUT_POSITION GRAM error.

$error = Globus::GRAM::Error::INVALID_STDOUT_POSITION() Create a new INVALID_STDOUT_POSITION GRAM error.

$error = Globus::GRAM::Error::RSL_STDERR_POSITION() Create a new RSL_STDERR_POSITION GRAM error.

$error = Globus::GRAM::Error::INVALID_STDERR_POSITION() Create a new INVALID_STDERR_POSITION GRAM error.

$error = Globus::GRAM::Error::RESTART_FAILED() Create a new RESTART_FAILED GRAM error.

$error = Globus::GRAM::Error::NO_STATE_FILE() Create a new NO_STATE_FILE GRAM error.

$error = Globus::GRAM::Error::READING_STATE_FILE() Create a new READING_STATE_FILE GRAM error.

$error = Globus::GRAM::Error::WRITING_STATE_FILE() Create a new WRITING_STATE_FILE GRAM error.

$error = Globus::GRAM::Error::OLD_JM_ALIVE() Create a new OLD_JM_ALIVE GRAM error.

$error = Globus::GRAM::Error::TTL_EXPIRED() Create a new TTL_EXPIRED GRAM error.

$error = Globus::GRAM::Error::SUBMIT_UNKNOWN() Create a new SUBMIT_UNKNOWN GRAM error.
$error = Create a new RSL_REMOTE_IO_URL GRAM error.
Globus::GRAM::Error::RSL_REMOTE_IO_URL()

$error = Create a new WRITING_REMOTE_IO_URL GRAM error.
Globus::GRAM::Error::WRITING_REMOTE_IO_URL()

$error = Create a new STDIO_SIZE GRAM error.
Globus::GRAM::Error::STDIO_SIZE()

$error = Create a new JM_STOPPED GRAM error.
Globus::GRAM::Error::JM_STOPPED()

$error = Create a new USER_PROXY_EXPIRED GRAM error.
Globus::GRAM::Error::USER_PROXY_EXPIRED()

$error = Create a new JOB_UNSUBMITTED GRAM error.
Globus::GRAM::Error::JOB_UNSUBMITTED()

$error = Create a new INVALID_COMMIT GRAM error.
Globus::GRAM::Error::INVALID_COMMIT()

$error = Create a new RSL_SCHEDULER_SPECIFIC GRAM error.
Globus::GRAM::Error::RSL_SCHEDULER_SPECIFIC()

$error = Create a new STAGE_IN_FAILED GRAM error.
Globus::GRAM::Error::STAGE_IN_FAILED()

$error = Create a new INVALID_SCRATCH GRAM error.
Globus::GRAM::Error::INVALID_SCRATCH()

$error = Create a new RSL_CACHE GRAM error.
Globus::GRAM::Error::RSL_CACHE()

$error = Create a new INVALID_SUBMIT_ATTRIBUTE GRAM error.
Globus::GRAM::Error::INVALID_SUBMIT_ATTRIBUTE()

$error = Create a new INVALID_STDIO_UPDATE_ATTRIBUTE GRAM error.
Globus::GRAM::Error::INVALID_STDIO_UPDATE_ATTRIBUTE()

$error = Create a new INVALID_RESTART_ATTRIBUTE GRAM error.
Globus::GRAM::Error::INVALID_RESTART_ATTRIBUTE()

$error = Create a new RSL_FILE_STAGE_IN GRAM error.
Globus::GRAM::Error::RSL_FILE_STAGE_IN()

$error = Create a new RSL_FILE_STAGE_IN_SHARED GRAM error.
Globus::GRAM::Error::RSL_FILE_STAGE_IN_SHARED()

$error = Create a new RSL_FILE_STAGE_OUT GRAM error.
Globus::GRAM::Error::RSL_FILE_STAGE_OUT()

$error = Create a new RSL_GASS_CACHE GRAM error.
Globus::GRAM::Error::RSL_GASS_CACHE()

$error = Create a new RSL_FILE_CLEANUP GRAM error.
Globus::GRAM::Error::RSL_FILE_CLEANUP()
APIs

$error = $error = Globus::GRAM::Error::RSL_SCRATCH()
Create a new RSL_SCRATCH GRAM error.

$error = $error = Globus::GRAM::Error::INVALID_SCHEDULER_SPECIFIC()
Create a new INVALID_SCHEDULER_SPECIFIC GRAM error.

$error = $error = Globus::GRAM::Error::UNDEFINED_ATTRIBUTE()
Create a new UNDEFINED_ATTRIBUTE GRAM error.

$error = $error = Globus::GRAM::Error::INVALID_CACHE()
Create a new INVALID_CACHE GRAM error.

$error = $error = Globus::GRAM::Error::INVALID_SAVE_STATE()
Create a new INVALID_SAVE_STATE GRAM error.

$error = $error = Globus::GRAM::Error::OPENING_VALIDATION_FILE()
Create a new OPENING_VALIDATION_FILE GRAM error.

$error = $error = Globus::GRAM::Error::READING_VALIDATION_FILE()
Create a new READING_VALIDATION_FILE GRAM error.

$error = $error = Globus::GRAM::Error::RSL_PROXY_TIMEOUT()
Create a new RSL_PROXY_TIMEOUT GRAM error.

$error = $error = Globus::GRAM::Error::INVALID_PROXY_TIMEOUT()
Create a new INVALID_PROXY_TIMEOUT GRAM error.

$error = $error = Globus::GRAM::Error::STAGE_OUT_FAILED()
Create a new STAGE_OUT_FAILED GRAM error.

$error = $error = Globus::GRAM::Error::JOB_CONTACT_NOT_FOUND()
Create a new JOB_CONTACT_NOT_FOUND GRAM error.

$error = $error = Globus::GRAM::Error::DELEGATION_FAILED()
Create a new DELEGATION_FAILED GRAM error.

$error = $error = Globus::GRAM::Error::LOCKING_STATE_LOCK_FILE()
Create a new LOCKING_STATE_LOCK_FILE GRAM error.

$error = $error = Globus::GRAM::Error::INVALID_ATTR()
Create a new INVALID_ATTR GRAM error.

$error = $error = Globus::GRAM::Error::NULL_PARAMETER()
Create a new NULL_PARAMETER GRAM error.

$error = $error = Globus::GRAM::Error::STILL_STREAMING()
Create a new STILL_STREAMING GRAM error.

$error = $error = Globus::GRAM::Error::AUTHORIZATION_DENIED()
Create a new AUTHORIZATION_DENIED GRAM error.

$error = $error = Globus::GRAM::Error::AUTHORIZATION_SYSTEM_FAILURE()
Create a new AUTHORIZATION_SYSTEM_FAILURE GRAM error.

$error = $error = Globus::GRAM::Error::AUTHORIZATION_DENIED_JOB_ID()
Create a new AUTHORIZATION_DENIED_JOB_ID GRAM error.
$error = Gentoo::GRAM::Error::AUTHORIZATION_DENIED_EXECUTABLE()
Create a new AUTHORIZATION_DENIED_EXECUTABLE GRAM error.

$error = Gentle::GRAM::Error::RSL_USER_NAME()
Create a new RSL_USER_NAME GRAM error.

$error = Gentle::GRAM::Error::INVALID_USER_NAME()
Create a new INVALID_USER_NAME GRAM error.

$error = Gentle::GRAM::Error::LAST()
Create a new LAST GRAM error.
Name
Globus::GRAM::JobDescription — GRAM Job Description

Synopsis

use Globus::GRAM::JobDescription;

$hash = { executable => [ '/bin/echo' ], arguments => [ 'hello' ] };
$description = new Globus::GRAM::JobDescription($filename);
$description = new Globus::GRAM::JobDescription($hash);
$executable = $description->executable();
$description->add($new_attribute, $new_value);
$description->save();
$description->save($filename);
$description->print_recursive($file_handle);

DESCRIPTION

This object contains the parameters of a job request in a simple object wrapper. The object may be queried to determine the value of any RSL parameter, may be updated with new parameters, and may be saved in the filesystem for later use.

Methods

new

A JobDescription is constructed from a file consisting of a Perl hash of parameter => array mappings. Every value in the Job Description is stored internally as an array, even single literals, similar to the way an RSL tree is parsed in C. An example of such a file is

$description =
{
    executable => [ '/bin/echo' ],
    arguments => [ 'hello', 'world' ],
    environment => [
        [
            'GLOBUS_GRAM_JOB_CONTACT',
            'https://globus.org:1234/2345/4332'
        ]
    ]
};

which corresponds to the rsl fragment

&{(executable = /bin/echo)
 (arguments = hello world)
 (environment =
     (GLOBUS_GRAM_JOB_CONTACT 'https://globus.org:1234/2345/4332')
 )
}

When the library_path RSL attribute is specified, this object modifies the environment RSL attribute value to append its value to any system specific variables.
$description->add('name', $value); Add a parameter to a job description. The parameter will be normalized internally so that the access methods described below will work with this new parameter. As an example,

$description->add('new_attribute', $new_value)

will create a new attribute in the JobDescription, which can be accessed by calling the $description->new_attribute() method.

$value $description->get('name'); Get a parameter from a job description. As an example,

$description->get('attribute')

will return the appropriate attribute in the JobDescription by name.

$description->save([$filename]) Save the JobDescription, including any added parameters, to the file named by $filename if present, or replacing the file used in constructing the object.

$description->print_recursive($file_handle) Write the value of the job description object to the file handle specified in the argument list.

$description->parameter() For any parameter defined in the JobDescription can be accessed by calling the method named by the parameter. The method names are automatically created when the JobDescription is created, and may be invoked with arbitrary SillyCaps or underscores. That is, the parameter gram_myjob may be accessed by the GramMyJob, grammyjob, or gram_my_job method names (and others).

If the attributes does not in this object, then undef will be returned.

In a list context, this returns the list of values associated with an attribute.

In a scalar context, if the attribute's value consist of a single literal, then that literal will be returned, otherwise undef will be returned.

For example, from a JobDescription called $d constructed from a description file containing

```
{  executable => [ '/bin/echo' ],
    arguments => [ 'hello', 'world' ]
}
```

The following will hold:

```
$executable = $d->executable()    # '/bin/echo'
$arguments = $d->arguments()      # undef
@executable = $d->executable()    # ('/bin/echo')
@arguments = $d->arguments()      # ('hello', 'world')
$not_present = $d->not_present()  # undef
@not_present = $d->not_present()  # ()
```

To test for existence of a value:

```
@not_present = $d->not_present()
print "Not defined\n" if(!defined($not_present[0]));
```
Name
Globus::GRAM::JobManager — Base class for all Job Manager scripts

Synopsis

$manager = new Globus::GRAM::JobManager($job_description);

$manager->log("Starting new operation");
$manager->nfssync($fileobj,$createflag);
$manager->respond($hashref);
$hashref = $manager->submit();
$hashref = $manager->poll();
$hashref = $manager->cancel();
$hashref = $manager->signal();
$hashref = $manager->make_scratchdir();
$hashref = $manager->remove_scratchdir();
$hashref = $manager->rewrite_urls();
$hashref = $manager->stage_in();
$hashref = $manager->stage_out();
$hashref = $manager->cache_cleanup();
$hashref = $manager->remote_io_file_create();
$hashref = $manager->proxy_relocate();
$hashref = $manager->proxy_update();
$scalar  = $manager->pipe_out_cmd(@arglist);
($stderr, $rc) = $manager->pipe_err_cmd(@arglist);
$status  = $manager->fork_and_exec_cmd(@arglist);
$manager->append_path($hash, $variable, $path);
$scalar = $manager->setup_softenv();

DESCRIPTION

The Globus::GRAM::JobManager module implements the base behavior for a Job Manager script interface. Scheduler-specific job manager scripts must inherit from this module in order to be used by the job manager.

Methods

$manager =
Globus::GRAM::JobManager->new($JobDescription)

Each Globus::GRAM::JobManager object is created by calling the constructor with a single argument, a Globus::GRAM::JobDescription object containing the information about the job request which the script will be modifying. Modules which subclass Globus::GRAM::JobManager MUST call the super-class's constructor, as in this code fragment:

my $proto = shift;
my $class = ref($proto) || $proto;
my $self = $class->SUPER::new(@_);

bless $self, $class;

$manager->log($string)

Log a message to the job manager log file. The message is preceded by a timestamp.

$manager->nfssync($object,$create)

Send an NFS update by touching the file (or directory) in question. If the $create is true, a file will be created. If it is false, the $object will not be created.
Send a response to the job manager program. The response may either be a hash reference consisting of a hash of (variable, value) pairs, which will be returned to the job manager, or an already formatted string. This only needs to be directly called by a job manager implementation when the script wants to send a partial response while processing one of the scheduler interface methods (for example, to indicate that a file has been staged).

The valid keys for a response are defined in the RESPONSES section.

Submit a job request to the scheduler. The default implementation returns with the Globus::GRAM::Error::UNIMPLEMENTED error. Scheduler specific subclasses should reimplement this method to submit the job to the scheduler.

A scheduler which implements this method should return a hash reference containing a scheduler-specific job identifier as the value of the hash's JOB_ID key, and optionally, the a GRAM job state as the value of the hash's JOB_STATE key if the job submission was successful; otherwise a Globus::GRAM::Error value should be returned. The job state values are defined in the Globus::GRAM::JobState module. The job parameters (as found in the job rsl) are defined in Globus::GRAM::Jobdescription object in $self->{JobDescription}.

For example:

```
return {JOB_STATE => Globus::GRAM::JobState::PENDING,
        JOB_ID => $job_id};
```

Poll a job's status. The default implementation returns with the Globus::GRAM::Error::UNIMPLEMENTED error. Scheduler specific subclasses should reimplement this method to poll the scheduler.

A scheduler which implements this method should return a hash reference containing the JOB_STATE value. The job's ID can be accessed by calling the $self->{JobDescription}->jobid() method.

Cancel a job. The default implementation returns with the Globus::GRAM::Error::UNIMPLEMENTED error. Scheduler specific subclasses should reimplement this method to remove the job from the scheduler.

A scheduler which implements this method should return a hash reference containing the JOB_STATE value. The job's ID can be accessed by calling the $self->{JobDescription}->jobid() method.

Signal a job. The default implementation returns with the Globus::GRAM::Error::UNIMPLEMENTED error. Scheduler specific subclasses should reimplement this method to remove the job from the scheduler. The JobManager module can determine the job's ID, the signal number, and the (optional) signal arguments from the Job Description by calling it's job_id(), signal(), and and signal_arg() methods, respectively.

Depending on the signal, it may be appropriate for the JobManager object to return a hash reference containing a JOB_STATE update.

Create a scratch directory for a job. The scratch directory location is based on the JobDescription's scratch_dir_base() and scratch_dir() methods.
If the scratch_dir() value is a relative path, then a directory will be created as a subdirectory of scratch_dir_base() / scratch_dir(), otherwise, it will be created as a subdirectory of scratch_dir(). This method will return a hash reference containing mapping SCRATCH_DIR to the absolute path of newly created scratch directory if successful.

$manager->remove_scratchdir() Delete a job's scratch directory. All files and subdirectories of the JobDescription's scratch_directory() will be deleted.

$manager->file_cleanup() Delete some job-related files. All files listed in the JobDescription's file_cleanup() array will be deleted.

$manager->rewrite_urls() Looks up URLs listed in the JobDescription's stdin() and executable(), and replaces them with paths to locally cached copies.

$manager->stage_in() Stage input files need for the job from remote storage. The files to be staged are defined by the array of [URL, path] pairs in the job description's file_stage_in() and file_stage_in_shared() methods. The Globus::GRAM::JobManager module provides an implementation of this functionality using the globus-url-copy and globus-gass-cache programs. Files which are staged in are not automatically removed when the job terminates.

This function returns intermediate responses using the Globus::GRAM::JobManager::response() method to let the job manager know when each individual file has been staged.

$manager->stage_out() Stage output files generated by this job to remote storage. The files to be staged are defined by the array of [URL, destination] pairs in the job description's file_stage_out() method. The Globus::GRAM::JobManager module provides an implementation of this functionality using the globus-url-copy program. Files which are staged out are not removed by this method.

$manager->cache_cleanup() Clean up cache references in the GASS which match this job's cache tag.

$manager->remote_io_file_create() Create the remote I/O file in the job dir which will contain the remote_io_url RSL attribute's value.

$manager->proxy_relocate() Relocate the delegated proxy for job execution. Job Managers need to override the default if they intend to relocate the proxy into some common file system other than the cache. The job manager program does not depend on the new location of the proxy. Job Manager modules must not remove the default proxy.

$hashref = $manager->proxy_update();

$manager->append_path($ref, $var, $path) Append $path to the value of $ref->{$var}, dealing with the case where $ref->{$var} is not yet defined.

$manager->pipe_out_cmd(@arg) Create a new process to run the first argument application with the remaining arguments (which may be empty). No shell metacharacter will be evaluated, avoiding a shell invocation. Stderr is redirected to /dev/null and stdout is being captured by the parent process, which is also the result returned. In list mode, all lines are returned, in scalar mode, only the first line is being re-
turned. The line termination character is already cut off. Use this function as more efficient backticks, if you do not need shell metacharacter evaluation.

Caution: This function deviates in two manners from regular backticks. Firstly, it chomps the line terminator from the output. Secondly, it returns only the first line in scalar context instead of a multiline concatenated string. As with regular backticks, the result may be undefined in scalar context, if no result exists.

A child error code with an exit code of 127 indicates that the application could not be run. The scalar result returned by this function is usually undef'ed in this case.

```perl
($stderr, $rc) = $manager->pipe_err_cmd(@arg)
```

Create a new process to run the first argument application with the remaining arguments (which may be empty). No shell metacharacter will be evaluated, avoiding a shell invocation.

This method returns a list of two items, the standard error of the program, and the exit code of the program. If the error code is 127, then the application could not be run. Standard output is discarded.

```perl
$manager->fork_and_exec_cmd(@arg)
```

Fork off a child to run the first argument in the list. Remaining arguments will be passed, but shell interpolation is avoided. Signals SIGINT and SIGQUIT are ignored in the child process. Stdout is appended to /dev/null, and stderr is dup2 from stdout. The parent waits for the child to finish, and returns the value for the CHILD_ERROR variable as result. Use this function as more efficient system() call, if you can do not need shell metacharacter evaluation.

Note that the inability to execute the program will result in a status code of 127.

```perl
$manager->job_dir()
```

Return the temporary directory to store job-related files, which have no need for file caching.

```perl
$manager->setup_softenv()
```

Either add a line to the specified command script file handle to load the user's default SoftEnv configuration, or create a custom SoftEnv script and add commands to the specified command script file handle to load it.

**RESPONSES**

When returning from a job interface method, or when sending an intermediate response via the `response()` method, the following hash keys are valid:

* **JOB_STATE** An integer job state value. These are enumerated in the Globus::GRAM::JobState module.

* **ERROR** An integer error code. These are enumerated in the Globus::GRAM::Error module.

* **JOB_ID** A string containing a job identifier, which can be used to poll, cancel, or signal a job in progress. This response should only be returned by the `submit` method.

* **SCRATCH_DIR** A string containing the path to a newly-created scratch directory. This response should only be returned by the `make_scratchdir` method.

* **STAGED_IN** A string containing the (URL, path) pair for a file which has now been staged in. This response should only be returned by the `stage_in` method.
<table>
<thead>
<tr>
<th>APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STAGED_IN.Shared</strong></td>
<td>A string containing the (URL, path) pair for a file which has now been staged in and symlinked from the cache. This response should only be returned by the stage_in_shared method.</td>
</tr>
<tr>
<td><strong>STAGED_OUT</strong></td>
<td>A string containing the (path, URL) pair for a file which has now been staged out by the script. This response should only be returned by the stage_out method.</td>
</tr>
</tbody>
</table>
Name
Globus::GRAM::JobSignal — GRAM Protocol JobSignal Constants

DESCRIPTION
The Globus::GRAM::JobSignal module defines symbolic names for the JobSignal constants in the GRAM Protocol.

Methods

$value =
Globus::GRAM::CANCEL()
Return the value of the CANCEL constant.

$value =
Globus::GRAM::SUSPEND()
Return the value of the SUSPEND constant.

$value =
Globus::GRAM::RESUME()
Return the value of the RESUME constant.

$value =
Globus::GRAM::PRIORITY()
Return the value of the PRIORITY constant.

$value =
Globus::GRAM::COMMIT_REQUEST()
Return the value of the COMMIT_REQUEST constant.

$value =
Globus::GRAM::COMMIT_EXTEND()
Return the value of the COMMIT_EXTEND constant.

$value =
Globus::GRAM::STDIO_UPDATE()
Return the value of the STDIO_UPDATE constant.

$value =
Globus::GRAM::STDIO_SIZE()
Return the value of the STDIO_SIZE constant.

$value =
Globus::GRAM::STOP_MANAGER()
Return the value of the STOP_MANAGER constant.

$value =
Globus::GRAM::COMMIT_END()
Return the value of the COMMIT_END constant.
Name
Globus::GRAM::JobState — GRAM Protocol JobState Constants

DESCRIPTION
The Globus::GRAM::JobState module defines symbolic names for the JobState constants in the GRAM Protocol.

Methods

$value =$
Globus::GRAM::PENDING()  Return the value of the PENDING constant.

$value =$
Globus::GRAM::ACTIVE()  Return the value of the ACTIVE constant.

$value =$
Globus::GRAM::FAILED()  Return the value of the FAILED constant.

$value =$ Globus::GRAM::DONE()  Return the value of the DONE constant.

$value =$
Globus::GRAM::SUSPENDED()  Return the value of the SUSPENDED constant.

$value =$
Globus::GRAM::UNSUBMITTED()  Return the value of the UNSUBMITTED constant.

$value =$
Globus::GRAM::STAGE_IN()  Return the value of the STAGE_IN constant.

$value =$
Globus::GRAM::STAGE_OUT()  Return the value of the STAGE_OUT constant.

$value =$ Globus::GRAM::ALL()  Return the value of the ALL constant.
Chapter 7. RSL Specification v1.1

This is a document to specify the existing RSL v1.0 implementation and interfaces, as they are provided in the GT 5.2.2 release. This document serves as a reference, and more introductory text.

The Globus Resource Specification Language (RSL) provides a common interchange language to describe resources. The various components of the Globus Resource Management architecture manipulate RSL strings to perform their management functions in cooperation with the other components in the system. The RSL provides the skeletal syntax used to compose complicated resource descriptions, and the various resource management components introduce specific ATTRIBUTE, VALUE> pairings into this common structure. Each attribute in a resource description serves as a parameter to control the behavior of one or more components in the resource management system.

1. RSL Syntax Overview

The core syntax of the RSL syntax is the relation. Relations associate an attribute name with a value, eg the relation executable=a.out provides the name of an executable in a resource request. There are two generative syntactic structures in the RSL that are used to build more complicated resource descriptions out of the basic relations: compound requests and value sequences. In addition, the RSL syntax includes a facility to both introduce and dereference string substitution variables.

The simplest form of compound request, utilized by all resource management components, is the conjunct-request. The conjunct-request expresses a conjunction of simple relations or compound requests (like a boolean AND). The most common conjunct-request in Globus RSL strings is the combination of multiple relations such as executable name, node count, executable arguments, and output files for a basic GRAM job request. Similarly, the core RSL syntax includes a disjunct-request form to represent disjunctive relations (like a boolean OR). Currently, however, no resource management component utilizes the disjunct-request form.

The last form of compound request is the multi-request. The multi-request expresses multiple parallel resources that make up a resource description. The multi-request form differs from the conjunction and disjunction in two ways: multi-requests introduce new variable scope, meaning variables defined in one clause of a multi-request are not visible to the other clauses, and multi-requests introduce a non-reducible hierarchy to the resource description. Whereas relations within a conjunct-request can be thought of as constraints on the resource being described, the subclauses of a multi-request are best thought of as individual resource descriptions that together constitute an abstract resource collection; the same attributes may be constrained in different ways in each subclause without causing a logical contradiction. An example of a contradiction would be to constrain the executable attribute to be two conflicting values within a conjunction. Currently, however, no resource management component utilizes the disjunct-request form.

The simplest form of value in the RSL syntax is the string literal. When explicitly quoted, literals can contain any character, and many common literals that don't contain special characters can appear without quotes. Values can also be variable references, in which case the variable reference is in essence replaced with the string value defined for that variable. RSL descriptions can also express string-concatenation of values, especially useful to construct long strings out of several variable references. String concatenation is supported with both an explicit concatenation operator and implicit concatenation for many idiomatic constructions involving variable references and literals.

In addition to the simple value forms given above, the RSL syntax includes the value sequence to express ordered sets of values. The value sequence syntax is used primarily for defining variables and for providing the argument list for a program.
2. RSL Tokenization Overview

Each RSL string consists of a sequence of RSL tokens, whitespace, and comments. The RSL tokens are either special syntax or regular unquoted literals, where special syntax contains one or more of the following listed special characters and unquoted literals are made of sequences of characters excluding the special characters.

The complete set of special characters that cannot appear as part of an unquoted literal is:

- + (plus)
- & (ampersand)
- | (pipe)
- ( (left paren)
- ) (right paren)
- = (equal)
- < (left angle)
- > (right angle)
- ! (exclamation)
- " (double quote)
- ' (apostrophe)
- ^ (carat)
- # (pound)
- $ (dollar)

These characters can only be used for the special syntactic forms described in the section and in the section or as within quoted literals.

Quoted literals are introduced with the " (double quote) or ' (single quote/apostrophe) and consist of all the characters up to (but not including) the next solo double or single quote, respectively. To escape a quote character within a quoted literal, the appearance of the quote character twice in a row is converted to a single instance of the character and the literal continues until the next solo quote character. For any quoted literal, there is only one possible escape sequence, eg within a literal delimited by the single quote character only the single quote character uses the escape notation and the double quote character can appear without escape.

Quoted literals can also be introduced with an alternate user delimiter notation. User delimited literals are introduced with the ^ (carat) character followed immediately by a user-provided delimiter; the literal consists of all the characters after the user's delimiter up to (but not including) the next solo instance of the delimiter. The delimiter itself may be escaped within the literal by providing two instances in a row, just as the regular quote delimiters are escaped in regular quoted literals.

RSL string comments use a notation similar to comments in the C programming language. Comments are introduced by the prefix /*. Comments continue to the first terminating suffix */ and cannot be nested. Comments are stripped from the RSL string during processing and are syntactically equivalent to whitespace.
Example 7.1. Quoted Literal Examples

Assign the value Hello. Welcome to "The Grid" to the attribute arguments, using double-quote as the delimiter and the escaping sequence.

arguments = "Hello. Welcome to ""The Grid""

Assign the value Hello. Welcome to "The Grid" to the attribute arguments using the single-quote delimiter.

arguments = 'Hello. Welcome to "The Grid'

Assign the value Hello. Welcome to "The Grid" to the attribute arguments using a user-defined quoting character !.

arguments = ^!Hello. Welcome to "The Grid"!

3. RSL Substitution Semantics

RSL strings can introduce and reference string variables. String substitution variables are defined in a special relation using the rsl_substitution attribute, and the definitions affect variable references made in the same conjunct-request (or disjunct-request), as well as references made within any multi-request nested inside one of the clauses of the conjunction (or disjunction). Each multi-request introduces a new variable scope for each subrequest, and variable definitions do not escape the closest enclosing scope.

Within any given scope, variable definitions are processed left-to-right in the resource description. Outermost scopes are processed before inner scopes, and the definitions in inner scopes augment the inherited definitions with new and/or updated variable definitions.

Variable definitions and variable references are processed in a single pass, with each definition updating the environment prior to processing the next definition. The value provided in a variable definition may include a reference to a previously-defined variable. References to variables that are not yet provided with definitions in the standard RSL variable processing order are replaced with an empty literal string.

4. RSL Attribute Summary

The RSL syntax is extensible because it defines structure without too many keywords. Each Globus resource management component introduces additional attributes to the set recognized by RSL-aware components, so it is difficult to provide a complete listing of attributes which might appear in a resource description. Resource management components are designed to utilize attributes they recognize and pass unrecognized relations through unchanged. This allows powerful compositions of different resource management functions.

The following listing summarizes the attribute names utilized by existing resource management components in the standard Globus release. Please see the individual component documentation for discussion of the attribute semantics.
Name
rsl — GRAM5 RSL Attributes

Description

arguments
The command line arguments for the executable. Use quotes, if a space is required in a single argument.

count
The number of executions of the executable. [Default: 1]

directory
Specifies the path of the directory the jobmanager will use as the default directory for the requested job. [Default: $ (HOME) ]

dry_run
If dryrun = yes then the jobmanager will not submit the job for execution and will return success. [Default: no]

environment
The environment variables that will be defined for the executable in addition to default set that is given to the job by the jobmanager.

executable
The name of the executable file to run on the remote machine. If the value is a GASS URL, the file is transferred to the remote gass cache before executing the job and removed after the job has terminated.

expiration
Time (in seconds) after a a job fails to receive a two-phase commit end signal before it is cleaned up. [Default: 14400]

file_clean_up
Specifies a list of files which will be removed after the job is completed.

file_stage_in
Specifies a list of ("remote URL" "local file") pairs which indicate files to be staged to the nodes which will run the job.

file_stage_in_shared
Specifies a list of ("remote URL" "local file") pairs which indicate files to be staged into the cache. A symlink from the cache to the "local file" path will be made.

file_stage_out
Specifies a list of ("local file" "remote URL") pairs which indicate files to be staged from the job to a GASS-compatible file server.

gass_cache
Specifies location to override the GASS cache location.

ggram_my_job
Obsolete and ignored. [Default: collective]

host_count
Only applies to clusters of SMP computers, such as newer IBM SP systems. Defines the number of nodes ("pizza boxes") to distribute the "count" processes across.

job_type
This specifies how the jobmanager should start the job. Possible values are single (even if the count > 1, only start 1 process or thread), multiple (start count processes or threads), mpi (use the appropriate method (e.g. mpirun) to start a program compiled with a vendor-provided MPI library. Program is started with count nodes), and condor (starts condor jobs in the "condor" universe.) [Default: multiple]

library_path
Specifies a list of paths to be appended to the system-specific library path environment variables. [Default: $ (GLOBUS_LOCATION) /lib]

logfile
Override the default log level for this job. The value of this attribute consists of a combination of the strings FATAL, ERROR, WARN, INFO, DEBUG, TRACE joined by the \ character
logpattern

Override the default log path pattern for this job. The value of this attribute is a string (potentially containing RSL substitutions) that is evaluated to the path to write the log to. If the resulting string contains the string $(DATE) (or any other RSL substitution), it will be reevaluated at log time.

max_cpu_time

Explicitly set the maximum cpu time for a single execution of the executable. The units is in minutes. The value will go through an atoi() conversion in order to get an integer. If the GRAM scheduler cannot set cpu time, then an error will be returned.

max_memory

Explicitly set the maximum amount of memory for a single execution of the executable. The units is in Megabytes. The value will go through an atoi() conversion in order to get an integer. If the GRAM scheduler cannot set maxMemory, then an error will be returned.

max_time

The maximum walltime or cpu time for a single execution of the executable. Walltime or cpu time is selected by the GRAM scheduler being interfaced. The units is in minutes. The value will go through an atoi() conversion in order to get an integer.

max_wall_time

Explicitly set the maximum walltime for a single execution of the executable. The units is in minutes. The value will go through an atoi() conversion in order to get an integer. If the GRAM scheduler cannot set walltime, then an error will be returned.

min_memory

Explicitly set the minimum amount of memory for a single execution of the executable. The units is in Megabytes. The value will go through an atoi() conversion in order to get an integer. If the GRAM scheduler cannot set minMemory, then an error will be returned.

project

Target the job to be allocated to a project account as defined by the scheduler at the defined (remote) resource.

proxy_timeout

Obsolete and ignored. Now a job-manager-wide setting.

queue

Target the job to a queue (class) name as defined by the scheduler at the defined (remote) resource.

remote_io_url

Writes the given value (a URL base string) to a file, and adds the path to that file to the environment through the GLOBUS_REMOTE_IO_URL environment variable. If this is specified as part of a job restart RSL, the job manager will update the file's contents. This is intended for jobs that want to access files via GASS, but the URL of the GASS server has changed due to a GASS server restart.

restart

Start a new job manager, but instead of submitting a new job, start managing an existing job. The job manager will search for the job state file created by the original job manager. If it finds the file and successfully reads it, it will become the new manager of the job, sending callbacks on status and streaming stdout/err if appropriate. It will fail if it detects that the old job manager is still alive (via a timestamp in the state file). If stdout or stderr was being streamed over the network, new stdout and stderr attributes can be specified in the restart RSL and the job manager will stream to the new locations (useful when output is going to a GASS server started by the client that's listening on a dynamic port, and the client was restarted). The new job manager will return a new contact string that should be used to communicate with it. If a job manager is restarted multiple times, any of the previous contact strings can be given for the restart attribute.

rsl_substitution

Specifies a list of values which can be substituted into other rsl attributes' values through the $(SUBSTITUTION) mechanism.
save_state

Causes the jobmanager to save it's job state information to a persistent file on disk. If the job manager exits or is suspended, the client can later start up a new job manager which can continue monitoring the job.

savejobdescription

Save a copy of the job description to $HOME [Default: no]

scratch_dir

Specifies the location to create a scratch subdirectory in. A SCRATCH_DIRECTORY RSL substitution will be filled with the name of the directory which is created.

stderr

The name of the remote file to store the standard error from the job. If the value is a GASS URL, the standard error from the job is transferred dynamically during the execution of the job. There are two accepted forms of this value. It can consist of a single destination: stderr = URL, or a sequence of destinations: stderr = (DESTINATION) (DESTINATION). In the latter case, the DESTINATION may itself be a URL or a sequence of an x-gass-cache URL followed by a cache tag. [Default: /dev/null]

stderr_position

Specifies where in the file remote standard error streaming should be restarted from. Must be 0.

stdin

The name of the file to be used as standard input for the executable on the remote machine. If the value is a GASS URL, the file is transferred to the remote gass cache before executing the job and removed after the job has terminated. [Default: /dev/null]

stdout

The name of the remote file to store the standard output from the job. If the value is a GASS URL, the standard output from the job is transferred dynamically during the execution of the job. There are two accepted forms of this value. It can consist of a single destination: stdout = URL, or a sequence of destinations: stdout = (DESTINATION) (DESTINATION). In the latter case, the DESTINATION may itself be a URL or a sequence of an x-gass-cache URL followed by a cache tag. [Default: /dev/null]

stdout_position

Specifies where in the file remote output streaming should be restarted from. Must be 0.

two_phase

Use a two-phase commit for job submission and completion. The job manager will respond to the initial job request with a WAITING_FOR_COMMIT error. It will then wait for a signal from the client before doing the actual job submission. The integer supplied is the number of seconds the job manager should wait before timing out. If the job manager times out before receiving the commit signal, or if a client issues a cancel signal, the job manager will clean up the job's files and exit, sending a callback with the job status as GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED. After the job manager sends a DONE or FAILED callback, it will wait for a commit signal from the client. If it receives one, it cleans up and exits as usual. If it times out and save_state was enabled, it will leave all of the job's files in place and exit (assuming the client is down and will attempt a job restart later). The timeoutvalue can be extended via a signal. When one of the following errors occurs, the job manager does not delete the job state file when it exits: GLOBUS_GRAM_PROTOCOL_ERROR_COMMIT_TIMED_OUT, GLOBUS_GRAM_PROTOCOL_ERROR_TTL_EXPIRED, GLOBUS_GRAM_PROTOCOL_ERROR_JM_STOPPED, GLOBUS_GRAM_PROTOCOL_ERROR_USER_PROXY_EXPIRED. In these
cases, it can not be restarted, so the job manager will not wait for the commit signal after sending the FAILED callback

username Verify that the job is running as this user.

5. Simple RSL Examples

The following are some simple example RSL strings to illustrate idiomatic usage with existing tools and to make concrete some of the more interesting cases of tokenization, concatenation, and variable semantics. These are meant to illustrate the use of the RSL notation without much regard for the specific details of a particular resource management component.

Typical GRAM5 resource descriptions contain at least a few relations in a conjunction:
Example 7.2. GRAM5 Job Request Examples

This example shows a conjunct request containing values that are unquoted literals and ordered sequences of a mix of quoted and unquoted literals.

```plaintext
(* this is a comment *)
& (executable = a.out (* <-- that is an unquoted literal *))
  (directory = /home/nobody )
  (arguments = arg1 "arg 2")
  (count = 1)
```

This example demonstrates RSL substitutions, which can be used to make sure a string is used consistently multiple times in a resource description:

```plaintext
& (rsl_substitution = (TOPDIR "/home/nobody")
    (DATADIR $(TOPDIR)"/data")
    (EXECDIR $(TOPDIR)/bin) )
  (executable = $(EXECDIR)/a.out
    (* ^-- implicit concatenation *))
  (directory = $(TOPDIR) )
  (arguments = $(DATADIR)/file1
    (* ^-- implicit concatenation *)
    $(DATADIR) # /file2
    (* ^-- explicit concatenation *)
    "$(FOO)"
    (* <-- a quoted literal *))
  (environment = (DATADIR $(DATADIR))
  (count = 1)
```

Performing all variable substitution and removing comments yields an equivalent RSL string:

```plaintext
& (rsl_substitution = (TOPDIR "/home/nobody")
    (DATADIR "/home/nobody/data")
    (EXECDIR "/home/nobody/bin") )
  (executable = "/home/nobody/bin/a.out" )
  (directory = "/home/nobody" )
  (arguments = "/home/nobody/data/file1"
    "/home/nobody/data/file2"
    "$(FOO)"
  )
  (environment = (DATADIR "/home/nobody/data")
  (count = 1)
```

Note in the above variable-substitution example, the variable substitution definitions are not automatically made a part of the job's environment. And explicit environment attribute must be used to add environment variables for the job. Also note that the third value in the arguments clause is not a variable reference but only quoted literal that happens to contain one of the special characters.

6. RSL grammar and tokenization rules

The following is a modified BNF grammar for the Resource Specification Language. Lexical rules are provided for the implicit concatenation sequences in the form of conventional regular expressions; for the implicit-concat non-terminal rules, whitespace is not allowed between juxtaposed non-terminals. Grammar comments are provided in square brackets in a column to the right of the productions, eg [comment] to help relate productions in the grammar to the terminology used in the above discussion.
Regular expressions are provided for the terminal class `string-literal` and for RSL comments. These regular expressions make use of a common inverted character-class notation, as popularized by the various lex tools. Comments are syntactically equivalent to whitespace and can only appear where the comment prefix cannot be mistaken for the trailing part of a multi-character unquoted literal.

### RSL Grammar

```plaintext
[1]   specification := relation
         | '+' spec-list
         | '&' spec-list
         | '|' spec-list
 /* relation */ /* multi-request */ /* conjunct-request */ /* disjunct-request */
[2]   spec-list := '(' specification ')' spec-list
         | '(' specification ')
 /* Substitution variable definition */ /* Attribute binding relation */
[3]   relation := 'rsl_substitution' '=' binding-sequence
         | attribute op value-sequence
 /* Substitution variable definition */ /* Attribute */
[4]   binding-sequence := binding binding-sequence
         | binding
 /* String */ /* Concatenation */
[5]   binding := '(' string-literal simple-value ')
 /* Variable Reference */
 [7]   op := '=' | '!=' | '>' | '>=' | '<' | '<='
 /* Attribute */
[8]   value-sequence := value value-sequence
         | value
 /* String */ /* Concatenation */
[9]   value := '(' value-sequence ')
         | simple-value
 /* User defined delimiter */
[10]  simple-value := string-literal
         | simple-value '# simple-value
         | implicit-concat | variable-reference
 /* Implicit concatenation */ /* Variable Reference */
 /* Implicit concatenation */
[12]  implicit-concat := (unquoted-literal)? (implicit-concat-core)+)
 /* Implicit concatenation */
         | (variable-reference) (unquoted-literal)
 /* User defined delimiter */
[14]  string-literal := quoted-literal
         | unquoted-literal
 /* Single-quote delimiter */
         | '' ((""')) (""')+''
         | '' c((("c")|cc))'" c
 /* Double-quote delimiter with escaping */ /* User defined delimiter c with escaping */
[16]  unquoted-literal := (^[\t\v\n+&()=<>!"'#$]+)
 /* Non-special characters */
[17]  comment := '(* (("*"|("|"*)))* "')
 /* Comment */
```
Chapter 8. Debugging

Log output from GRAM5 is a useful tool for debugging issues. GRAM5 can log to either local files or syslog. See the Admin Guide for information about how to configure logging.

In most cases, logging at the INFO level will produce enough information to show progress of most operations. Adding DEBUG will also allow log information from the GRAM LRM scripts.

1. Basic Debugging Methods

The first thing to determine when debugging unexpected failures is to determine whether the gatekeeper service is running, reachable from the client, and properly configured.

First, determine that the gatekeeper is running by using a tool such as telnet to connect to the TCP/IP port that the gatekeeper is listening on. From the GRAM service node, using a default configuration, use a command like:

```
% telnet localhost 2119
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'
```

An error message like the following indicates that the gatekeeper service is not starting:

```
telnet: connect to address 127.0.0.1: Connection refused
telnet: Unable to connect to remote host
```

If the telnet command exits immediately, then the gatekeeper service is being started but not running. Check the gatekeeper log (by default $GLOBUS_LOCATION/var/globus-gatekeeper.log) to see if there is an error message. A common error is having a missing library path environment variable in the gatekeeper's environment or having a malformed configuration file. See the globus-gatekeeper for information on the configuration options.

The next recommended diagnostic is to run the same telnet command from the machine which is acting as the GRAM client if it is distinct from the GRAM service node. Be sure to replace localhost with the actual host name of the GRAM service. Again, check for log entries in the case of immediate exit or refused connection. If the connection does not work, then there may be some network connectivity or firewall issues preventing access.

Next use a tool like globusrun to diagnose whether the client is authorized to contact the gatekeeper service. This is done by using the -a command-line option. For example:

```
% globusrun -a -r grid.example.org
GRAM Authentication test successful
```

If you do not get the success message above, then check the gatekeeper log to see if there is a diagnostic message. A common problem is that the identity of the client is not in the grid mapfile used by the gatekeeper.

The next test is to use the -dryrun option to globusrun to verify that the job manager service is properly configured. To do so, try the following:

```
% globusrun -dryrun -r grid.example.org "&{executable=/bin/sh}"
globus_gram_client_callback_allow successful
Dryrun successful
```

If you do not get the success message above, first check the error number in the GRAM5 Error codes table to determine how to proceed. If the result is unclear, check the job manager log (default $HOME/gram_DATE.log) to see if there are any further details of the error.
The final test is to submit a test job to the GRAM5 service and wait for it to terminate, such as this example shows:

```bash
% globus-job-run grid.example.org /bin/sh -c 'echo "hello, grid"'
hello, grid
```

If the process appears to hang, it might be that the job manager is unable to send state callbacks to the client. Check that there are no firewalls or network issues that would prevent the job manager process from connecting from the GRAM service node to the client node.

## 2. Advanced Debugging Methods

The methods described in this section are intended for debugging problems in the GRAM code, not in the user environment.

### 2.1. Debugging the Job Manager

To debug the GRAM5 job manager, run the command located in `$GLOBUS_LOCATION/etc/grid-services/jobmanager-LRM`(ignoring the first 3 fields). For example:

```bash
% $GLOBUS_LOCATION/libexec/globus-job-manager \
  -conf $GLOBUS_LOCATION/etc/globus-job-manager.conf -type fork
```

When the job manager is started in this way, it will log messages to standard error and will terminate 60 seconds after its last job has completed. This only works if there are no job managers running for this particular user. The job manager can be started in a debugger such as `gdb` or `valgrind` using a similar command-line.
Chapter 9. Troubleshooting

For a list of error codes generated by GRAM5, see Section 3, “Errors”.

1. GRAM Client Troubleshooting

1.1. Credential Problems

GRAM requires a client certificate and private key in order to authenticate with the GRAM service. If these are not available, the GRAM client will fail. In typical use, a user will create a temporary proxy certificate either derived from their identity certificate issued by some certificate authority, or from a service such as myproxy. If a GRAM client command returns any error containing the string `GSS Major Status` you've hit a credential problem. Look at the Troubleshooting Section of the GSI manual for details about how to diagnose and correct these errors. The `grid-cert-diagnostics` tool with the `-p` command-line option is especially helpful for diagnosing some of these types of problems.

1.2. Connection Problems

There are a few things which can go wrong when trying to contact a GRAM service. These have slightly different error types which can help diagnose which problem is occurring.

1.2.1. Invalid Resource Name

If the hostname or TCP port you are using for a GRAM resource name is not correct, then the GRAM client will be unable to access the service. Errors of this type will look like this:

```
% globus-job-run grid.example.org/jobmanager-fork /bin/hostname
GRAM Job submission failed because the connection to the server failed (check host and port) (error code 12)
```

When this occurs, check with the resource administrator for correct resource naming so that you can contact the service.

1.2.2. Mutual Authentication Failure

GRAM performs mutual authentication, that is, both the client and service provide certificates indicating who they are. The service uses the client's identity to map the user to a local unix account. The client uses the server's identity to verify that the service is running with a host credential. The failure of the client to trust the server's certificate will generate an error message that looks like this: `globus_gsi_gssapi: Authorization denied: The expected name for the remote host (host@alias.example.org) does not match the authenticated name of the remote host (host@grid.example.org). This happens when the name in the host certificate does not match the information obtained from DNS and is often a DNS configuration problem.`

This mismatch can happen for a number of reasons: a site administrator has multiple hosts sharing a certificate, a host has multiple DNS aliases, and the client is not aware of which name the server is using for its certificate, or a host's name has changed since the certificate was issued. The remedy for the client, after confirming with the GRAM administrator that the name after "authenticated name of the remote host" is the correct certificate name is to use a form of the GRAM resource name which includes this name. For example, explicitly adding a name to the abbreviated GRAM contact so that instead of `alias.example.org`, you would use `alias.example.org::host@grid.example.org`. 
1.2.3. Certificate Trust Issues

Because of the mutual authentication, both GRAM users and services can hit problems if they do not trust their peer's certificate or the Certificate Authority which issued it. If the client doesn't trust the server's certificate, it is easier to diagnose, because the GRAM service doesn't send much information back to the client if it doesn't trust it. However, working with the system administrator to get information from the GRAM logs will usually fix these problems fairly easily.

If the service's certificate is not trusted, the client will receive a message like this:

```
% globus-job-run grid.example.org /bin/hostname
GRAM Job submission failed because an authentication operation failed
OpenSSL Error: s3_clnt.c:915: in library: SSL routines, function SSL3_GET_SERVER_CERTIFICATE
globus_gsi_callback_module: Could not verify credential
globus_gsi_callback_module: Can't get the local trusted CA certificate: Untrusted self-signed certificate: Untrusted self-signed certificate (bbfccedf in this case).
```

This error indicates that certificate chain from the service certificate to the client contained a self-signed certificate (usually an indication that it's a CA certificate), which the client doesn't trust, and includes the hash of the certificate name (bbfccedf in this case). If you hit this particular type of error, you should send the information to the GRAM administrator and determine which CA should be trusted and what its signing policy is, to determine if you want to add it to your local set of trust roots.

**Note**

Different versions of OpenSSL produce different hashes for the same certificate names. If you upgrade a system (or transfer CA certificates between systems) to a different version of OpenSSL, you may hit this problem even if you think you have the CA certificate in your trusted certificate directory. If so, run the `globus-update-certificate-dir` program to update your hashes.

There are other reasons why a certificate might not be trusted (it's in a revoked list, it has expired or was issued in the future, etc). For more details look at the troubleshooting information in the GSI user's guide.

If for some reason the service does not trust your certificate, you'll get a rather cryptic message from GRAM that looks like this:

```
% globus-job-run grid.example.org /bin/hostname
GRAM Job submission failed because an authentication operation failed
globus_gsi_gssapi: Unable to verify remote side's credentials
globus_gsi_gssapi: Unable to verify remote side's credentials: Couldn't verify the remote certificate: Untrusted self-signed certificate: Untrusted self-signed certificate (error code 7)
```

To remedy this, consult the GRAM administrator to get information from the `/var/log/globus-gatekeeper.log` file to determine the reason why the gatekeeper didn't like your certificate. Again it could be CA trust issues, clock skew, or a revoked certificate. The error in the gatekeeper log would typically look like the client-side trust issue above.

1.2.4. Authentication with the Remote Server Failed

Once the GRAM service has authenticated the client, it maps the client's identity to a local user account using a grid-mapfile or other mapping service. If this fails, the client will receive a message that looks like this:

```
% globus-job-run grid.example.org /bin/hostname
GRAM Job submission failed because authentication with the remote server failed (error code 7)
```

Note

Different versions of OpenSSL produce different hashes for the same certificate names. If you upgrade a system (or transfer CA certificates between systems) to a different version of OpenSSL, you may hit this problem even if you think you have the CA certificate in your trusted certificate directory. If so, run the `globus-update-certificate-dir` program to update your hashes.

There are other reasons why a certificate might not be trusted (it's in a revoked list, it has expired or was issued in the future, etc). For more details look at the troubleshooting information in the GSI user's guide.

If for some reason the service does not trust your certificate, you'll get a rather cryptic message from GRAM that looks like this:

```
% globus-job-run grid.example.org /bin/hostname
GRAM Job submission failed because an authentication operation failed
globus_gsi_gssapi: Unable to verify remote side's credentials
globus_gsi_gssapi: Unable to verify remote side's credentials: Couldn't verify the remote certificate: Untrusted self-signed certificate: Untrusted self-signed certificate (error code 7)
```

To remedy this, consult the GRAM administrator to get information from the `/var/log/globus-gatekeeper.log` file to determine the reason why the gatekeeper didn't like your certificate. Again it could be CA trust issues, clock skew, or a revoked certificate. The error in the gatekeeper log would typically look like the client-side trust issue above.
To remedy this, consult the system administrator of the GRAM resource to be added to the authorized user's list. Be sure to send your credential subject name to make it easier for them. To get that information, run the command `grid-cert-info -s`.

### 1.2.5. Unable to Find the Requested Service

Recall that a GRAM resource name includes a component called the service name. The default if not specified is `jobmanager`, but some sites may not use that name, or have a different LRM name than you expect. If you specify an incorrect service name, or the default is not present, you'll get an error that looks like this:

```
% globus-job-run grid.example.org /bin/hostname
GRAM Job submission failed because the gatekeeper failed to find the requested service (error code 93)
```

If you get this error, you'll need to determine which services are available on that GRAM resource, either by asking the admin or by looking at the entries in `/etc/grid-services`.

### 1.2.6. Failed to Run the Job Manager

The GRAM service is split between a privileged process called the `globus-gatekeeper` and a non-privileged process called the `globus-job-manager` which runs as a user process. If the `globus-gatekeeper` is unable to locate the `globus-job-manager` process, then this misconfiguration will show up like this:

```
% globus-job-run grid.example.org /bin/hostname
GRAM Job submission failed because the gatekeeper failed to run the job manager (error code 47)
```

This is an installation mistake, and the administrator of the GRAM resource must fix this.

### 1.3. Jobs are Hanging

One problem GRAM users sometimes encounter is that it looks like jobs submitted to GRAM are not making any progress, even though the local resource manager thinks they've run. There are a couple of reasons why this might occur: GRAM is not getting the information it needs from the local resource manager or the GRAM client is not getting the information it needs. We'll cover diagnosing and handling the latter case in this document, as the other is an system administrator issue.

The way `globus-job-run` and `globusrun` determine that jobs have completed is via GRAM job state callbacks. These are messages sent by the GRAM service to the client node indicating that something significant has happened in the lifecycle of the job. If for some reason the GRAM service can not get those messages to the client, the client will not be able to detect job state changes.

In order to determine if this is the case, submit a job using `globus-job-submit`, and then use the `globus-job-status` command to see if the job state changes. If it does not, then consult the GRAM administrator—there might be some problem with the installation. If it does, then for some reason the callbacks are not happening. This might be firewall issues or host naming issues.

The GRAM client sends a "callback contact" to the GRAM service when it submits a job, in order that it can receive notifications. This contact is a reference to a https server embedded in the GRAM client which only handles GRAM state callbacks. As with all web servers, it has a URL which defines how to contact it, which in this case consists of the client host name and the service port number. If the host name that is used is not resolvable (such as a for a laptop with a dynamic address), then the GRAM service will not be able to contact it. If that's the case, you can set the `GLOBUS_HOSTNAME` environment variable to the IP address that your client can be reached at, and then submit your jobs. This will cause GRAM to publish that address instead of what it thinks the client's host name is.

Another way that the GRAM service would be unable to send job state updates to a client would be if there's a firewall between the service and the client. If that's the case, you might need to set the `GLOBUS_TCP_PORT_RANGE`
environment variable to a comma-separated list of numbers which represent a range of minimum and maximum TCP port numbers to listen on. You might have to contact your site administrator to determine what TCP ports are allowed. If there are none, you can still use `globus-job-submit` and `globus-job-status` to track your job's state changes, or use another tool like those mentioned in the section about client tools.

### 1.4. Logs and Debugging

The GRAM service has a log file which contains information about the job as it is processed. These logs are located by default in `/var/log/globus/gram_USERNAME.log`. There are some different logging levels available, as described in the GRAM Administrator's Guide. These can be controlled on a per-job basis by adding the `loglevel` RSL attribute to your job description. The default is to log only FATAL and ERROR messages, but other levels can sometimes help understand what is going on.

### 1.5. Diagnosing LRM Errors

Sometimes, bugs creep into the LRM adapter scripts. When that occurs, the GRAM job will usually fail with an error like this:

```
GRAM Job failed because the job manager detected an invalid script status (error code 25)
```

If this occurs, you may have to work with a GRAM administrator to help debug this problem. One helpful thing you can do when reporting it is to save the GRAM internal script data so that it can be used outside of the GRAM service to see what the low-level error looks like. To do this, add the RSL fragment `(savejobdescription = yes)` to your job request. This will cause GRAM to leave a file called something like `$HOME/gram_[0-9]*.pl` in your home directory. You can use this with the internal tool `/usr/share/globus/globus-job-manager-script.pl` to try to submit the job to the LRM without using the GRAM service. The command line `/usr/share/globus/globus-job-manager-script.pl -m LRM -c submit -f GRAM-PL-FILE` will attempt to submit the job to the LRM. It will show all the information the LRM script sends to the GRAM service, which might include some perl-language error or badly formatted output from the script (which must only output lines which begin with `GRAM_SCRIPT_`.

In some extreme cases, the savejobdescription option will not generate a file. If that's the case, pass `/dev/null` as the argument to the `-f` command-line option. The problem is likely a perl syntax error which will be reached before the job description is loaded.

### 1.6. Email Support

If all else fails, please send information about your problem to `<gram-user@globus.org>`. You'll have to subscribe to a list before you can send an e-mail to it. See [here](http://dev.globus.org/wiki/Mailing_Lists) for general e-mail lists and information on how to subscribe to a list and [here](http://dev.globus.org/wiki/GRAM#Mailing_Lists) for GRAM specific lists. Depending on the problem, you may be requested to file a bug report to the globus project's [Issue Tracker](http://jira.globus.org).

### 2. Admin Troubleshooting

#### 2.1. Security

GRAM requires a host certificate and private key in order for the `globus-gatekeeper` service to run. These are typically located in `/etc/grid-security/hostcert.pem` and `/etc/grid-security/hostkey.pem`, but

---

1 [http://dev.globus.org/wiki/Mailing_Lists](http://dev.globus.org/wiki/Mailing_Lists)
2 [http://dev.globus.org/wiki/GRAM#Mailing_Lists](http://dev.globus.org/wiki/GRAM#Mailing_Lists)
3 [http://jira.globus.org](http://jira.globus.org)
the path is configurable in the gatekeeper configuration file. The key must be protected by file permissions allowing only the root user to read it.

GRAM also (by default) uses a grid-mapfile to authorize Grid users as local users. This file is typically located in /etc/grid-security/grid-mapfile, but is configurable in the gatekeeper configuration file.

Problems in either of these configurations will show up in the gatekeeper log described below. See the GSI documentation for more detailed information about obtaining and installing host certificates and maintaining a grid-mapfile.

### 2.2. Verify that Services are Running

GRAM relies on the globus-gatekeeper program and (in some cases) the globus-scheduler-event-generator programs to process jobs. If the former is not running, jobs requests will fail with a "connection refused" error. If the latter is not running, GRAM jobs will appear to "hang" in the PENDING state.

The globus-gatekeeper is typically started via an init script installed in /etc/init.d/globus-gatekeeper. The command /etc/init.d/globus-gatekeeper status will indicate whether the service is running. See Section 2, “Starting and Stopping GRAM5 services” for more information about starting and stopping the globus-gatekeeper program.

If the globus-gatekeeper service fails to start, the output of the command globus-gatekeeper -test will output information describing some types of configuration problems.

The globus-scheduler-event-generator is typically started via an init script installed in /etc/init.d/globus-scheduler-event-generator. It is only needed when the LRM-specific “setup-seg” package is installed. The command /etc/init.d/globus-scheduler-event-generator status will indicate whether the service is running. See Section 2, “Starting and Stopping GRAM5 services” for more information about starting and stopping the globus-scheduler-event-generator program.

### 2.3. Verify that LRM packages are installed

The globus-gatekeeper program starts the globus-job-manager service with different command-line parameters depending on the LRM being used. Use the command globus-gatekeeper-admin -l to list which LRMs the gatekeeper is configured to use.

The globus-job-manager-script.pl is the interface between the GRAM job manager process and the LRM adapter. The command /usr/share/globus/globus-job-manager-script.pl -h will print the list of available adapters.

```bash
% /usr/share/globus/globus-job-manager-script.pl -h
USAGE: /usr/share/globus/globus-job-manager-script.pl -m MANAGER -f FILE -c COMMAND
Installed managers: condor fork
```

The globus-scheduler-event-generator also uses an LRM-specific module to generate scheduler events for GRAM to reduce the amount of resources GRAM uses on the machine where it runs. To determine which LRMs are installed and configured, use the command globus-scheduler-event-generator-admin -l.

```bash
% globus-scheduler-event-generator-admin -l
fork [DISABLED]
```

If any of these do not show the LRM you are trying to use, install the relevant packages related to that LRM and restart the GRAM services. See the GRAM Administrator's Guide for more information about starting and stopping the GRAM services.
2.4. Verify that the LRM packages are configured

All GRAM5 LRM adapters have a configuration file for site customizations, such as queue names, paths to executables needed to interface with the LRM, etc. Check that the values in these files are correct. These files are described in Section 4, “LRM Adapter Configuration”.

2.5. Check the Gatekeeper Log

The /var/log/globus-gatekeeper.log file contains information about service requests from clients, and will be useful when diagnosing service startup failures, authentication failures, and authorization failures.

2.5.1. Authorization failures

GRAM uses GSI to authenticate client job requests. If there is a problem with the GSI configuration for your host, or a client is trying to connect with a certificate signed by a CA your host does not trust, the job request will fail. This will show up in the log as a “GSS authentication failure”. See the GSI Administrator's Guide for information about diagnosing authentication failures.

2.5.2. Gridmap failures

After authentication is complete, GRAM maps the Grid identity to a local user prior to starting the globus-job-manager process. If this fails, an error will show up in the log as "globus_gss_assist_gridmap() failed authorization". See the GSI Administrator's Guide for information about managing gridmap files.

2.6. Job Manager Logs

A per-user job manager log is typically located in /var/log/globus/gram_\$USERNAME.log. This log contains information from the job manager as it attempts to execute GRAM jobs via a local resource manager. The logs can be fairly verbose. Sometimes looking for log entries near those containing the string level=ERROR will show more information about what caused a particular failure.

Once you've found an error in the log, it is generally useful to find log entries related to the job which hit that error. There are two job IDs associated with each job, one a GRAM-specific ID, and one an LRM-specific ID. To determine the GRAM ID associated with a job, look for the attribute gramid in the log message. Finding that, looking for all other log messages which contain that gramid value will give a better picture of what the job manager is doing. To determine the LRM-specific ID, look for a message at TRACE level with the matching GRAM ID found above with the response value matching GRAM_SCRIPT_JOB_ID:LRM-ID. You can then find follow the state of the LRM-ID as well as the GRAM ID in the log, and correlate the LRM-ID information with local resource manager logs and administrative tools.

2.7. Email Support

If all else fails, please send information about your problem to <gram-user@globus.org>. You'll have to subscribe to a list before you can send an e-mail to it. See here\(^4\) for general e-mail lists and information on how to subscribe to a list and here\(^5\) for GRAM-specific lists. Depending on the problem, you may be requested to file a bug report to the Globus project's Issue Tracker\(^6\).

3. Errors

\(^4\) [http://dev.globus.org/wiki/Mailing_Lists](http://dev.globus.org/wiki/Mailing_Lists)
\(^5\) [http://dev.globus.org/wiki/GRAM#Mailing_Lists](http://dev.globus.org/wiki/GRAM#Mailing_Lists)
\(^6\) [http://jira.globus.org](http://jira.globus.org)
## Table 9.1. GRAM5 Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Reason</th>
<th>Possible Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>one of the RSL parameters is not supported</td>
<td>Check RSL documentation</td>
</tr>
<tr>
<td>2</td>
<td>the RSL length is greater than the maximum allowed</td>
<td>Use RSL substitutions to reduce length of RSL strings</td>
</tr>
<tr>
<td>3</td>
<td>an I/O operation failed</td>
<td>Enable trace logging and report to <a href="mailto:gram-dev@globus.org">gram-dev@globus.org</a></td>
</tr>
<tr>
<td>4</td>
<td>jobmanager unable to set default to the directory requested</td>
<td>Check that RSL directory attribute refers to a directory that exists on the target system.</td>
</tr>
<tr>
<td>5</td>
<td>the executable does not exist</td>
<td>Check that the RSL executable attribute refers to an executable that exists on the target system.</td>
</tr>
<tr>
<td>6</td>
<td>of an unused INSUFFICIENT_FUNDS</td>
<td>Unimplemented feature.</td>
</tr>
<tr>
<td>7</td>
<td>authentication with the remote server failed</td>
<td>Check that the contact string contains the proper X.509 DN.</td>
</tr>
<tr>
<td>8</td>
<td>the user cancelled the job</td>
<td>Don't cancel jobs you want to complete.</td>
</tr>
<tr>
<td>9</td>
<td>the system cancelled the job</td>
<td>Check RSL requirements such as maximum time and memory are valid for the job.</td>
</tr>
<tr>
<td>10</td>
<td>data transfer to the server failed</td>
<td>Check gatekeeper and/or job manager logs to see why the process failed.</td>
</tr>
<tr>
<td>11</td>
<td>the stdin file does not exist</td>
<td>Check that the RSL stdin attribute refers to a file that exists on the target system or has a valid ftp, gsiftp, http, or https URL.</td>
</tr>
<tr>
<td>12</td>
<td>the connection to the server failed (check host and port)</td>
<td>Check that the service is running on the expected TCP/IP port. Check that no firewall prevents contacting that TCP/IP port. Check $GLOBUS_LOCATION/var/globus-gatekeeper.log for runtime configuration errors.</td>
</tr>
<tr>
<td>13</td>
<td>the provided RSL 'maxtime' value is not an integer</td>
<td>Check that the RSL maxtime value evaluates to an integer.</td>
</tr>
<tr>
<td>14</td>
<td>the provided RSL 'count' value is not an integer</td>
<td>Check that the RSL count value evaluates to an integer.</td>
</tr>
<tr>
<td>15</td>
<td>the job manager received an invalid RSL</td>
<td>Check that the RSL string can be parsed by using <code>globusrun -p RSL</code>.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td>16</td>
<td>the job manager failed in allowing others to make contact</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>17</td>
<td>the job failed when the job manager attempted to run it</td>
<td>Verify that the LRM is configured properly.</td>
</tr>
<tr>
<td>18</td>
<td>an invalid paradigm was specified</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>19</td>
<td>the provided RSL 'jobtype' value is invalid</td>
<td>The RSL jobtype attribute is not indicated as supported by the LRM. Valid jobtype values are single, multiple, mpi, and condor.</td>
</tr>
<tr>
<td>20</td>
<td>the provided RSL 'myjob' value is invalid</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>21</td>
<td>the job manager failed to locate an internal script argument file</td>
<td>Check that $GLOBUS_LOCATION/libexec/globus-job-manager-script.pl exists and is executable. Check that the LRM-specific perl module is located in $GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/ directory and is valid. The command perl -I $GLOBUS_LOCATION/lib/perl $GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/LRM.pm can be used to check if there are any syntax errors in the script.</td>
</tr>
<tr>
<td>22</td>
<td>the job manager failed to create an internal script argument file</td>
<td>Check that your home directory is writable and not full.</td>
</tr>
<tr>
<td>23</td>
<td>the job manager detected an invalid job state</td>
<td>Check job manager logs.</td>
</tr>
<tr>
<td>24</td>
<td>the job manager detected an invalid script response</td>
<td>Check job manager logs. This is likely a bug in the LRM script.</td>
</tr>
<tr>
<td>25</td>
<td>the job manager detected an invalid script status</td>
<td>Check job manager logs. This is likely a bug in the LRM script.</td>
</tr>
<tr>
<td>26</td>
<td>the provided RSL 'jobtype' value is not supported by this job manager</td>
<td>Check that the RSL jobtype attribute is implemented by the LRM script. Note that some job types require configuration</td>
</tr>
<tr>
<td>27</td>
<td>unused ERROR_UNIMPLEMENTED</td>
<td>LRM does not support some feature included in the job request.</td>
</tr>
<tr>
<td>28</td>
<td>the job manager failed to create an internal script submission file</td>
<td>Check that the user's home file system is not full. Check job manager log</td>
</tr>
<tr>
<td>29</td>
<td>the job manager cannot find the user proxy</td>
<td>Check that client is delegating a proxy when authenticating with</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td>30</td>
<td>the job manager failed to open the user proxy</td>
<td>Check that the user's home filesystem and the /tmp file system are not full.</td>
</tr>
<tr>
<td>31</td>
<td>the job manager failed to cancel the job as requested</td>
<td>Check that the user's home filesystem and the /tmp file system are not full.</td>
</tr>
<tr>
<td>32</td>
<td>system memory allocation failed</td>
<td>Check job manager log for details.</td>
</tr>
<tr>
<td>33</td>
<td>the interprocess job communication initialization failed</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>34</td>
<td>the interprocess job communication setup failed</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>35</td>
<td>the provided RSL 'host count' value is invalid</td>
<td>Check that the RSL host_count attribute evaluates to an integer.</td>
</tr>
<tr>
<td>36</td>
<td>one of the provided RSL parameters is unsupported</td>
<td>Check job manager log for details about invalid parameter.</td>
</tr>
<tr>
<td>37</td>
<td>the provided RSL 'queue' parameter is invalid</td>
<td>Check that the RSL queue attribute evaluates to a string that corresponds to an LRM-specific queue name.</td>
</tr>
<tr>
<td>38</td>
<td>the provided RSL 'project' parameter is invalid</td>
<td>Check that the RSL project attribute evaluates to a string that corresponds to an LRM-specific project name.</td>
</tr>
<tr>
<td>39</td>
<td>the provided RSL string includes variables that could not be identified</td>
<td>Check that all RSL substitutions are defined before being used in the job description.</td>
</tr>
<tr>
<td>40</td>
<td>the provided RSL 'environment' parameter is invalid</td>
<td>Check that the RSL environment attribute contains a sequence of VARIABLE VALUE pairs.</td>
</tr>
<tr>
<td>41</td>
<td>the provided RSL 'dryrun' parameter is invalid</td>
<td>Remove the RSL dryrun attribute from the job description.</td>
</tr>
<tr>
<td>42</td>
<td>the provided RSL is invalid (an empty string)</td>
<td>Include a non-empty RSL string in your job submission request.</td>
</tr>
<tr>
<td>43</td>
<td>the job manager failed to stage the executable</td>
<td>Check that the file service hosting the executable is reachable from the GRAM5 service node. Check that the executable exists on the file service node. Check that there is sufficient disk space in the user's home directory on the service node to store the executable.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td>44</td>
<td>the job manager failed to stage the stdin file</td>
<td>Check that the file service hosting the standard input file is reachable from the GRAM5 service node. Check that the standard input file exists on the file service node. Check that there is sufficient disk space in the user's home directory on the service node to store the standard input file.</td>
</tr>
<tr>
<td>45</td>
<td>the requested job manager type is invalid</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>46</td>
<td>the provided RSL 'arguments' parameter is invalid</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>47</td>
<td>the gatekeeper failed to run the job manager</td>
<td>Check the gatekeeper or job manager logs for more information.</td>
</tr>
<tr>
<td>48</td>
<td>the provided RSL could not be properly parsed</td>
<td>Check that the RSL string can be parsed by using <code>globusrun -p RSL</code></td>
</tr>
<tr>
<td>49</td>
<td>there is a version mismatch between GRAM components</td>
<td>Ask system administrator to upgrade GRAM service to GRAM2 or GRAM5</td>
</tr>
<tr>
<td>50</td>
<td>the provided RSL 'arguments' parameter is invalid</td>
<td>Check that the RSL arguments attribute evaluates to a sequence of strings.</td>
</tr>
<tr>
<td>51</td>
<td>the provided RSL 'count' parameter is invalid</td>
<td>Check that the RSL count attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>52</td>
<td>the provided RSL 'directory' parameter is invalid</td>
<td>Check that the RSL directory attribute evaluates to a string.</td>
</tr>
<tr>
<td>53</td>
<td>the provided RSL 'dryrun' parameter is invalid</td>
<td>Check that the RSL dryrun attribute evaluates to either yes or no.</td>
</tr>
<tr>
<td>54</td>
<td>the provided RSL 'environment' parameter is invalid</td>
<td>Check that the RSL environment attribute evaluates to a sequence of VARIABLE, VALUE pairs.</td>
</tr>
<tr>
<td>55</td>
<td>the provided RSL 'executable' parameter is invalid</td>
<td>Check that the RSL executable attribute evaluates to a string value.</td>
</tr>
<tr>
<td>56</td>
<td>the provided RSL 'host_count' parameter is invalid</td>
<td>Check that the RSL host_count attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>57</td>
<td>the provided RSL 'jobtype' parameter is invalid</td>
<td>Check that the RSL jobtype attribute evaluates to one of single, multiple, mpi, or condor</td>
</tr>
</tbody>
</table>
## Troubleshooting

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Reason</th>
<th>Possible Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>the provided RSL 'maxtime' parameter is invalid</td>
<td>Check that the RSL maxtime attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>59</td>
<td>the provided RSL 'myjob' parameter is invalid</td>
<td>OBSOLETE IN GRAM5.</td>
</tr>
<tr>
<td>60</td>
<td>the provided RSL 'paradyn' parameter is invalid</td>
<td>OBSOLETE IN GRAM2.</td>
</tr>
<tr>
<td>61</td>
<td>the provided RSL 'project' parameter is invalid</td>
<td>Check that the RSL project attribute evaluates to a string value.</td>
</tr>
<tr>
<td>62</td>
<td>the provided RSL 'queue' parameter is invalid</td>
<td>Check that the RSL queue attribute evaluates to a string value.</td>
</tr>
<tr>
<td>63</td>
<td>the provided RSL 'stderr' parameter is invalid</td>
<td>Check that the RSL stderr attribute evaluates to a string value or a sequence of DESTINATION URLs with optional CACHE_TAG string parameters.</td>
</tr>
<tr>
<td>64</td>
<td>the provided RSL 'stdin' parameter is invalid</td>
<td>Check that the RSL stdin attribute evaluates to a string value.</td>
</tr>
<tr>
<td>65</td>
<td>the provided RSL 'stdout' parameter is invalid</td>
<td>Check that the RSL stdout attribute evaluates to a string value or a sequence of DESTINATION URLs with optional CACHE_TAG string parameters.</td>
</tr>
<tr>
<td>66</td>
<td>the job manager failed to locate an internal script</td>
<td>Check job manager log for more details.</td>
</tr>
<tr>
<td>67</td>
<td>the job manager failed on the system call pipe()</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>68</td>
<td>the job manager failed on the system call fcntl()</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>69</td>
<td>the job manager failed to create the temporary stdout filename</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>70</td>
<td>the job manager failed to create the temporary stderr filename</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>71</td>
<td>the job manager failed on the system call fork()</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>72</td>
<td>the executable file permissions do not allow execution</td>
<td>Check that the RSL executable attribute refers to an executable program or script.</td>
</tr>
<tr>
<td>73</td>
<td>the job manager failed to open stdout</td>
<td>Check that the RSL stdout attribute refers to one or more valid destination files or URLs.</td>
</tr>
<tr>
<td>74</td>
<td>the job manager failed to open stderr</td>
<td>Check that the RSL stderr attribute refers to one or more valid destination files or URLs.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>75</td>
<td>the cache file could not be opened in order to relocate the user proxy</td>
<td>Check that the user's home directory is writable and not full on the GRAM5 service node.</td>
</tr>
<tr>
<td>76</td>
<td>cannot access cache files in ~/.globus/.gass_cache, check permissions, quota, and disk space</td>
<td>Check that the user's home directory is writable and not full on the GRAM5 service node.</td>
</tr>
<tr>
<td>77</td>
<td>the job manager failed to insert the contact in the client contact list</td>
<td>Check job manager log</td>
</tr>
<tr>
<td>78</td>
<td>the contact was not found in the job manager's client contact list</td>
<td>Don't attempt to unregister callback contacts that are not registered</td>
</tr>
<tr>
<td>79</td>
<td>connecting to the job manager failed. Possible reasons: job terminated, invalid job contact, network problems, ...</td>
<td>Check that the job manager process is running. Check that the job manager credential has not expired. Check that the job manager contact refers to the correct TCP/IP host and port. Check that the job manager contact is not blocked by a firewall.</td>
</tr>
<tr>
<td>80</td>
<td>the syntax of the job contact is invalid</td>
<td>Check the syntax of job contact string.</td>
</tr>
<tr>
<td>81</td>
<td>the executable parameter in the RSL is undefined</td>
<td>Include the RSL executable in all job requests.</td>
</tr>
<tr>
<td>82</td>
<td>the job manager service is misconfigured. condor arch undefined</td>
<td>Add the -condor-arch to the command-line or configuration file for a job manager configured to use the condor LRM.</td>
</tr>
<tr>
<td>83</td>
<td>the job manager service is misconfigured. condor os undefined</td>
<td>Add the -condor-os to the command-line or configuration file for a job manager configured to use the condor LRM.</td>
</tr>
<tr>
<td>84</td>
<td>the provided RSL 'min_memory' parameter is invalid</td>
<td>Check that the RSL min_memory attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>85</td>
<td>the provided RSL 'max_memory' parameter is invalid</td>
<td>Check that the RSL max_memory attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>86</td>
<td>the RSL 'min_memory' value is not zero or greater</td>
<td>Check that the RSL min_memory attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>87</td>
<td>the RSL 'max_memory' value is not zero or greater</td>
<td>Check that the RSL max_memory attribute evaluates to a positive integer value.</td>
</tr>
<tr>
<td>88</td>
<td>the creation of a HTTP message failed</td>
<td>Check job manager log</td>
</tr>
<tr>
<td>89</td>
<td>parsing incoming HTTP message failed</td>
<td>Check job manager log</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>90</td>
<td>the packing of information into a HTTP message failed</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>91</td>
<td>an incoming HTTP message did not contain the expected information</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>92</td>
<td>the job manager does not support the service that the client requested</td>
<td>Check that the client is talking to the correct service.</td>
</tr>
<tr>
<td>93</td>
<td>the gatekeeper failed to find the requested service</td>
<td>OBSOLET IN GRAM2</td>
</tr>
<tr>
<td>94</td>
<td>the job manager does not accept any new requests (shutting down)</td>
<td>Execute queries before the job has been cleaned up.</td>
</tr>
<tr>
<td>95</td>
<td>the client failed to close the listener associated with the callback URL</td>
<td>Call <code>globus_gram_client_callback_disallow()</code> with a valid the callback contact.</td>
</tr>
<tr>
<td>96</td>
<td>the gatekeeper contact cannot be parsed</td>
<td>Check the syntax of the gatekeeper contact string you are attempting to contact.</td>
</tr>
<tr>
<td>97</td>
<td>the job manager could not find the 'poe' command</td>
<td>OBSOLET IN GRAM2</td>
</tr>
<tr>
<td>98</td>
<td>the job manager could not find the 'mpirun' command</td>
<td>Configure the LRM script with <code>mpirun</code> in your path.</td>
</tr>
<tr>
<td>99</td>
<td>the provided RSL 'start_time' parameter is invalid</td>
<td>OBSOLET IN GRAM2</td>
</tr>
<tr>
<td>100</td>
<td>the provided RSL 'reservation_handle' parameter is invalid</td>
<td>OBSOLET IN GRAM2</td>
</tr>
<tr>
<td>101</td>
<td>the provided RSL 'max_wall_time' parameter is invalid</td>
<td>Check that the RSL <code>max_wall_time</code> attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>102</td>
<td>the RSL 'max_wall_time' value is not zero or greater</td>
<td>Check that the RSL <code>max_wall_time</code> attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>103</td>
<td>the provided RSL 'max_cpu_time' parameter is invalid</td>
<td>Check that the RSL <code>max_cpu_time</code> attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>104</td>
<td>the RSL 'max_cpu_time' value is not zero or greater</td>
<td>Check that the RSL <code>max_cpu_time</code> attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>105</td>
<td>the job manager is misconfigured, a scheduler script is missing</td>
<td>Check that the administrator has configured the LRM by running its setup script.</td>
</tr>
<tr>
<td>106</td>
<td>the job manager is misconfigured, a scheduler script has invalid permisions</td>
<td>Check that the administrator has installed the <code>GLLOBUS_LOCATION/libexec/globus-job-manag-</code></td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>-------------------</td>
</tr>
<tr>
<td>107</td>
<td>the job manager failed to signal the job</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>108</td>
<td>the job manager did not recognize/support the signal type</td>
<td>Check that your signal operation is using the correct signal constant.</td>
</tr>
<tr>
<td>109</td>
<td>the job manager failed to get the job id from the local scheduler</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>110</td>
<td>the job manager is waiting for a commit signal</td>
<td>Send a two-phase commit signal to the job manager to acknowledge receiving the job contact from the job manager.</td>
</tr>
<tr>
<td>111</td>
<td>the job manager timed out while waiting for a commit signal</td>
<td>Send a two-phase commit signal to the job manager to acknowledge receiving the job contact from the job manager. Increase the two-phase commit time out for your job. Check that the job manager contact TCP/IP port is reachable from your client.</td>
</tr>
<tr>
<td>112</td>
<td>the provided RSL 'save_state' parameter is invalid</td>
<td>Check that the RSL save_state attribute is set to yes or no.</td>
</tr>
<tr>
<td>113</td>
<td>the provided RSL 'restart' parameter is invalid</td>
<td>Check that the RSL restart attribute evaluates to a string containing a job contact string.</td>
</tr>
<tr>
<td>114</td>
<td>the provided RSL 'two_phase' parameter is invalid</td>
<td>Check that the RSL two_phase attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>115</td>
<td>the RSL 'two_phase' value is not zero or greater</td>
<td>Check that the RSL two_phase attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>116</td>
<td>the provided RSL 'stdout_position' parameter is invalid</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>117</td>
<td>the RSL 'stdout_position' value is not zero or greater</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>118</td>
<td>the provided RSL 'stderr_position' parameter is invalid</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>119</td>
<td>the RSL 'stderr_position' value is not zero or greater</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>120</td>
<td>the job manager restart attempt failed</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>121</td>
<td>the job state file doesn't exist</td>
<td>Check that the job contact you are trying to restart matches one that the job manager returned to you.</td>
</tr>
<tr>
<td>122</td>
<td>could not read the job state file</td>
<td>Check that the state file directory is not full.</td>
</tr>
<tr>
<td>123</td>
<td>could not write the job state file</td>
<td>Check that the state file directory is not full.</td>
</tr>
<tr>
<td>124</td>
<td>old job manager is still alive</td>
<td>Contact the returned job manager contact to manage the job you are trying to restart.</td>
</tr>
<tr>
<td>125</td>
<td>job manager state file TTL expired</td>
<td>OBSOLETE in GRAM2</td>
</tr>
<tr>
<td>126</td>
<td>it is unknown if the job was submitted</td>
<td>Check job manager log.</td>
</tr>
<tr>
<td>127</td>
<td>the provided RSL 'remote_io_url' parameter is invalid</td>
<td>Check that the RSL remote_io_url attribute evaluates to a string value.</td>
</tr>
<tr>
<td>128</td>
<td>could not write the remote io url file</td>
<td>Check that the user's home file system on the job manager service node is writable and not full.</td>
</tr>
<tr>
<td>129</td>
<td>the standard output/error size is different</td>
<td>Send a stdio update signal to redirect the job manager output to a new URL</td>
</tr>
<tr>
<td>130</td>
<td>the job manager was sent a stop signal (job is still running)</td>
<td>Submit a restart request to monitor the job.</td>
</tr>
<tr>
<td>131</td>
<td>the user proxy expired (job is still running)</td>
<td>Generate a new proxy and then submit a restart request to monitor the job.</td>
</tr>
<tr>
<td>132</td>
<td>the job was not submitted by original jobmanager</td>
<td>OBSOLETE IN GRAM2</td>
</tr>
<tr>
<td>133</td>
<td>the job manager is not waiting for that commit signal</td>
<td>Do not send a commit signal to a job that is not waiting for a commit signal.</td>
</tr>
<tr>
<td>134</td>
<td>the provided RSL scheduler specific parameter is invalid</td>
<td>Check the LRM-specific documentation to determine what values are legal for the RSL extensions implemented by the LRM.</td>
</tr>
<tr>
<td>135</td>
<td>the job manager could not stage in a file</td>
<td>Check that the file service hosting the file to stage is reachable from the GRAM5 service node. Check that the file to stage exists on the file service node. Check that there is sufficient disk space in the user's home directory on the service node to store the file to stage.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td>136</td>
<td>the scratch directory could not be created</td>
<td>Check that the directory named by the RSL <code>scratch_dir</code> attribute exists and is writable. Check that the directory named by the RSL <code>scratch_dir</code> attribute is not full.</td>
</tr>
<tr>
<td>137</td>
<td>the provided <code>gass_cache</code> parameter is invalid</td>
<td>Check that the RSL <code>gass_cache</code> attribute evaluates to a string.</td>
</tr>
<tr>
<td>138</td>
<td>the RSL contains attributes which are not valid for job submission</td>
<td>Do not use restart- or signal-only RSL attributes when submitting a job.</td>
</tr>
<tr>
<td>139</td>
<td>the RSL contains attributes which are not valid for stdio update</td>
<td>Do not use submit- or restart-only RSL attributes when sending a stdio update signal to a job.</td>
</tr>
<tr>
<td>140</td>
<td>the RSL contains attributes which are not valid for job restart</td>
<td>Do not use submit- or signal-only RSL attributes when restarting a job.</td>
</tr>
<tr>
<td>141</td>
<td>the provided RSL <code>file_stage_in</code> parameter is invalid</td>
<td>Check that the RSL <code>file_stage_in</code> attribute evaluates to a sequence of <code>SOURCE DESTINATION</code> pairs.</td>
</tr>
<tr>
<td>142</td>
<td>the provided RSL <code>file_stage_in_shared</code> parameter is invalid</td>
<td>Check that the RSL <code>file_stage_in_shared</code> attribute evaluates to a sequence of <code>SOURCE DESTINATION</code> pairs.</td>
</tr>
<tr>
<td>143</td>
<td>the provided RSL <code>file_stage_out</code> parameter is invalid</td>
<td>Check that the RSL <code>file_stage_out</code> attribute evaluates to a sequence of <code>SOURCE DESTINATION</code> pairs.</td>
</tr>
<tr>
<td>144</td>
<td>the provided RSL <code>gass_cache</code> parameter is invalid</td>
<td>Check that the RSL <code>gass_cache</code> attribute evaluates to a string.</td>
</tr>
<tr>
<td>145</td>
<td>the provided RSL <code>file_cleanup</code> parameter is invalid</td>
<td>Check that the RSL <code>file_clean_up</code> attribute evaluates to a sequence of strings.</td>
</tr>
<tr>
<td>146</td>
<td>the provided RSL <code>scratch_dir</code> parameter is invalid</td>
<td>Check that the RSL <code>scratch_dir</code> attribute evaluates to a string.</td>
</tr>
<tr>
<td>147</td>
<td>the provided scheduler-specific RSL parameter is invalid</td>
<td>Check the LRM-specific documentation to determine what values are legal for the RSL extensions implemented by the LRM.</td>
</tr>
<tr>
<td>148</td>
<td>a required RSL attribute was not defined in the RSL spec</td>
<td>Check that the RSL <code>executable</code> attribute is present in your job request RSL. Check that</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the RSL.restart attributes is present in your restart RSL.</td>
</tr>
<tr>
<td>149</td>
<td>the gass_cache attribute points to an invalid cache directory</td>
<td>Check that the RSL gass_cache attributes evaluates to a directory that exists or can be created. Check that the user's home file system is writable and not full.</td>
</tr>
<tr>
<td>150</td>
<td>the provided RSL 'save_state' parameter has an invalid value</td>
<td>Check that the RSL save_state attribute has a value of yes or no.</td>
</tr>
<tr>
<td>151</td>
<td>the job manager could not open the RSL attribute validation file</td>
<td>Check that $GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf is present and readable on the job manager service node. Check that $GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf is readable on the job manager service node if present.</td>
</tr>
<tr>
<td>152</td>
<td>the job manager could not read the RSL attribute validation file</td>
<td>Check that $GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf is valid. Check that $GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf is valid if present.</td>
</tr>
<tr>
<td>153</td>
<td>the provided RSL 'proxy_timeout' is invalid</td>
<td>Check that RSL proxy_timeout attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>154</td>
<td>the RSL 'proxy_timeout' value is not greater than zero</td>
<td>Check that RSL proxy_timeout attribute evaluates to a positive integer.</td>
</tr>
<tr>
<td>155</td>
<td>the job manager could not stage out a file</td>
<td>Check that the source file being staged exists on the job manager service node. Check that the directory of the destination file being staged exists on the file service node. Check that the directory of the destination file being staged is writable by the user. Check that the destination file service is reachable by the job manager service node.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>156</td>
<td>the job contact string does not match any which the job manager is handling</td>
<td>Check that the job contact string matches one returned from a job request.</td>
</tr>
<tr>
<td>157</td>
<td>proxy delegation failed</td>
<td>Check that the job manager service node trusts the signer of your credential.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check that you trust the signer of the job manager service node's credential.</td>
</tr>
<tr>
<td>158</td>
<td>the job manager could not lock the state lock file</td>
<td>Check that the file system holding the job state directory supports POSIX advisory locking.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check that the job state directory is writable by the user on the service node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check that the job state directory is not full.</td>
</tr>
<tr>
<td>159</td>
<td>an invalid globus_io_clientattr_t was used.</td>
<td>Check that you have initialized the globus_io_clientattr_t attribute prior to using it with the GRAM client API.</td>
</tr>
<tr>
<td>160</td>
<td>an null parameter was passed to the gram library</td>
<td>Check that you are passing legal values to all GRAM API calls.</td>
</tr>
<tr>
<td>161</td>
<td>the job manager is still streaming output</td>
<td>OBSOLETE IN GRAM5</td>
</tr>
<tr>
<td>162</td>
<td>the authorization system denied the request</td>
<td>Check with your GRAM system administrator to allow a particular certificate to be authorized.</td>
</tr>
<tr>
<td>163</td>
<td>the authorization system reported a failure</td>
<td>Check with your system administrator to verify that the authorization system is configured properly.</td>
</tr>
<tr>
<td>164</td>
<td>the authorization system denied the request - invalid job id</td>
<td>Check with your system administrator to verify that the authorization system is configured properly. Use a credential which is authorized to interact with a particular GRAM job.</td>
</tr>
<tr>
<td>165</td>
<td>the authorization system denied the request - not authorized to run the specified executable</td>
<td>Check with your system administrator to verify that the authorization system is configured properly. Use a credential which is authorized to interact with a particular GRAM job.</td>
</tr>
<tr>
<td>166</td>
<td>the provided RSL 'user_name' parameter is invalid.</td>
<td>Check that the RSL user_name attribute evaluates to a string.</td>
</tr>
<tr>
<td>167</td>
<td>the job is not running in the account named by the 'user_name' parameter.</td>
<td>Ask with the GRAM system administrator to add an authorization entry to allow your credential to</td>
</tr>
<tr>
<td>Error Code</td>
<td>Reason</td>
<td>Possible Solutions</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>run jobs as the specified user account.</td>
</tr>
</tbody>
</table>
Chapter 10. Semantics and syntax of protocols

1. GRAM5 Protocol

The GRAM Protocol is used to handle communication between the Gatekeeper, Job Manager, and GRAM Clients. The protocol is based on a subset of the HTTP/1.1 protocol, with a small set of message types and responses sent as the body of the HTTP requests and responses. This document describes GRAM Protocol version 2 as used by GRAM5. This is compatible with with the GRAM Protocol parsers in GRAM2 with extensions.

1.1. Framing

GRAM messages are framed in HTTP/1.1 messages. However, only a small subset of the HTTP specification is used or understood by the GRAM system. All GRAM requests are HTTP POST messages. Only the following HTTP headers are understood:

- Host
- Content-Type (set to "application/x-globus-gram" in all cases)
- Content-Length
- Connection (set to "close" in all HTTP responses)

Only the following status codes are supported in response's HTTP Status-Line:

- 200 OK
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error
- 400 Bad Request

1.2. Message Format

All messages use the carriage return (ASCII value 13) followed by line feed (ASCII value 10) sequence to delimit lines. In all cases, a blank line separates the HTTP header from the message body. All application/x-globus-gram message bodies consist of attribute names followed by a colon, a space, and then the value of the attribute. When the value may contain a newline or double-quote character, a special escaping rule is used to encapsulate the complete string. This encapsulation consists of surrounding the string with double-quotes, and escaping all double-quote and backslash characters within the string with a backslash. All other characters are sent without modification. For example, the string

```plaintext
rsl: &{ executable = "*/bin/echo" } ( arguments = "hello" )
```

becomes
Semantics and syntax of protocols

rsl: "& ( executable = "$bin/echo$" )
  (arguments = "$hello$")"

In GRAM5, protocol extensions are supported in the status update messages. These extensions are implemented as extra attribute names after all of the attributes defined in the messages below. Older GRAM protocol parsers will ignore those extensions that occur after the attributes in the messages defined below. In GRAM5, the following extensions are used:

- **exit-code**
  Job exit code. Sent in job state callbacks and in job status replies when the job completes.

- **gt3-failure-type**
  Failure detail type for staging errors. Sent in job state callbacks and in job status replies when a job fails.

- **gt3-failure-message**
  Failure detail message for more context for errors. Sent in job state callbacks and in job status replies when a job fails.

- **gt3-failure-source**
  Failure detail message for the source of a failed file transfer. Sent in job state callbacks and in job status replies when a job fails.

- **gt3-failure-destination**
  Failure detail message for the destination of a failed file transfer. Sent in job state callbacks and in job status replies when a job fails.

- **version**
  Job manager package version. Sent in all messages from the job manager.

- **toolkit-version**
  Toolkit release that the job manager is running. Sent in all messages from the job manager.

This is the only form of quoting which application/x-globus-gram messages support. Use of \%HEXHEX escapes (such as seen in URL encodings) is not meaningful for this protocol.

### 1.3. Message Types

#### 1.3.1. Ping Request

A ping request is used to verify that the gatekeeper is configured properly to handle a named service. The ping request consists of the following:

```
POST ping/job-manager-name HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size

protocol-version: version
```

The values of the message-specific strings are

- **job-manager-name**
  The name of the service to have the gatekeeper check. The service name corresponds to one of the gatekeeper's configured grid-services, and is usually of the form "jobmanager-LRM".

- **host-name**
  The name of the host on which the gatekeeper is running. This exists only for compatibility with the HTTP/1.1 protocol.

- **message-size**
  The length of the content of the message, not including the HTTP/1.1 header.
version  
The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string "2".

### 1.3.2. Job Request

A job request is used to scheduler a job remotely using GRAM. The ping request consists of the HTTP framing described above with the request-URI consisting of `job-manager-name`, where `job-manager name` is the name of the service to use to schedule the job. The format of a job request message consists of the following:

```plaintext
POST job-manager-name[@user-name] HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size

protocol-version: version
job-state-mask: mask
callback-url: callback-contact
rsl: rsl-description
```

The values of the emphasized text items are as below:

- **job-manager-name**  The name of the service to submit the job request to. The service name corresponds to one of the gatekeeper's configured grid-services, and is usually of the form jobmanager-LRM.
- **user-name**  Starting with GT4.0, a client may request that a certain account by used by the gatekeeper to start the job manager. This is done optionally by appending the @ symbol and the local user name that the job should be run as to the `job-manager-name`. If the @ and user-name are not present, then the first grid map entry will be used. If the client credential is not authorized in the grid map to use the specified account, an authorization error will occur in the gatekeeper.
- **host-name**  The name of the host on which the gatekeeper is running. This exists only for compatibility with the HTTP/1.1 protocol.
- **message-size**  The length of the content of the message, not including the HTTP/1.1 header.
- **version**  The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string 2.
- **mask**  An integer representation of the job state mask. This value is obtained from a bitwise-OR of the job state values which the client wishes to receive job status callbacks about. These meanings of the various job state values are defined in the GRAM Protocol API documentation.
- **callback-contact**  A https URL which defines a GRAM protocol listener which will receive job state updates. The from a bitwise-OR of the job state values which the client wishes to receive job status callbacks about. The job status update messages are defined below.
- **rsl-description**  A quoted string containing the RSL description of the job request.

### 1.3.3. Status Request

A status request is used by a GRAM client to get the current job state of a running job. This type of message can only be sent to a job manager's job-contact (as returned in the reply to a job request message). The format of a job request message consists of the following:
Semantics and syntax of protocols

POST job-contact HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size
protocol-version: version

"status"

The values of the emphasized text items are as below:

job-contact  The job contact string returned in a response to a job request message, or determined by querying the MDS system.
host-name    The name of the host on which the job manager is running. This exists only for compatibility with the HTTP/1.1 protocol.
message-size The length of the content of the message, not including the HTTP/1.1 header.
version      The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string 2.

1.3.4. Callback Register Request

A callback register request is used by a GRAM client to register a new callback contact to receive GRAM job state updates. This type of message can only be sent to a job manager's job-contact (as returned in the reply to a job request message). The format of a job request message consists of the following:

POST job-contact HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size

protocol-version: version
"register mask callback-contact"

The values of the emphasized text items are as below:

job-contact  The job contact string returned in a response to a job request message, or determined by querying the MDS system.
host-name    The name of the host on which the job manager is running. This exists only for compatibility with the HTTP/1.1 protocol.
message-size The length of the content of the message, not including the HTTP/1.1 header.
version      The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string 2.
mask         An integer representation of the job state mask. This value is obtained from a bitwise-OR of the job state values which the client wishes to receive job status callbacks about. These meanings of the various job state values are defined in the GRAM Protocol API documentation.
callback-contact A https URL which defines a GRAM protocol listener which will receive job state updates. The from a bitwise-OR of the job state values which the client wishes to receive job status callbacks about. The job status update messages are defined below.
1.3.5. Callback Unregister Request

A callback unregister request is used by a GRAM client to request that the job manager no longer send job state updates to the specified callback contact. This type of message can only be sent to a job manager’s job-contact (as returned in the reply to a job request message). The format of a job request message consists of the following:

```
POST job-contact HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size

protocol-version: version
"unregister callback-contact"
```

The values of the emphasized text items are as below:

- **job-contact**: The job contact string returned in a response to a job request message, or determined by querying the MDS system.
- **host-name**: The name of the host on which the job manager is running. This exists only for compatibility with the HTTP/1.1 protocol.
- **message-size**: The length of the content of the message, not including the HTTP/1.1 header.
- **version**: The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string "2".
- **callback-contact**: A https URL which defines a GRAM protocol listener which should no longer receive job state updates. The from a bitwise-OR of the job state values which the client wishes to receive job status callbacks about. The job status update messages are defined @ref globus_gram_protocol_job_state_updates "below".

1.3.6. Job Cancel Request

A job cancel request is used by a GRAM client to request that the job manager terminate a job. This type of message can only be sent to a job manager’s job-contact (as returned in the reply to a job request message). The format of a job request message consists of the following:

```
POST job-contact HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size

protocol-version: version
"cancel"
```

The values of the emphasized text items are as below:

- **job-contact**: The job contact string returned in a response to a job request message, or determined by querying the MDS system.
- **host-name**: The name of the host on which the job manager is running. This exists only for compatibility with the HTTP/1.1 protocol.
**message-size**  The length of the content of the message, not including the HTTP/1.1 header.

**version**  The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string 2.

### 1.3.7. Job Signal Request

A job signal request is used by a GRAM client to request that the job manager process a signal for a job. The arguments to the various signals are discussed in the protocol library documentation. The format of a job request message consists of the following:

```
POST job-contact HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size

protocol-version: version
"signal"
```

The values of the emphasized text items are as below:

**job-contact**  The job contact string returned in a response to a job request message, or determined by querying the MDS system.

**host-name**  The name of the host on which the job manager is running. This exists only for compatibility with the HTTP/1.1 protocol.

**message-size**  The length of the content of the message, not including the HTTP/1.1 header.

**version**  The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string 2.

**signal**  A quoted string containing the signal number and its parameters.

### 1.3.8. Job State Updates

A job status update message is sent by the job manager to all registered callback contacts when the job's status changes. The format of the job status update messages is as follows:

```
POST callback-contact HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size

protocol-version: version
job-manager-url: job-contact
status: status-code
failure-code: failure-code
```

The values of the emphasized text items are as below:

**callback-contact**  The callback contact string registered with the job manager either by being passed as the callback-contact in a job request message or in a callback register message.

**host-name**  The host part of the callback-contact URL. This exists only for compatibility with the HTTP/1.1 protocol.
message-size: The length of the content of the message, not including the HTTP/1.1 header.

version: The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string 2.

job-contact: The job contact of the job which has changed states.

1.3.9. Proxy Delegation

A proxy delegation message is sent by the client to the job manager to initiate a delegation handshake to generate a new proxy credential for the job manager. This credential is used by the job manager or the job when making further secured connections. The format of the delegation message is as follows:

POST callback-contact HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size

protocol-version: version
"renew"

If a successful (200) reply is sent in response to this message, then the client will proceed with a GSI delegation handshake. The tokens in this handshake will be framed with a 4 byte big-endian token length header. The framed tokens will then be wrapped using the GLOBUS_IO_SECURE_CHANNEL_MODE_SSL_WRAP wrapping mode. The job manager will frame response tokens in the same manner. After the job manager receives its final delegation token, it will respond with another response message that indicates whether the delegation was processed or not. This response message is a standard GRAM response message.

1.3.10. Security Attributes

The following security attributes are needed to communicate with the Gatekeeper:

- Authentication must be done using GSSAPI mutual authentication

- Messages must be wrapped with support for the delegation message. When using Globus I/O, this is accomplished by using the the GLOBUS_IO_SECURE_CHANNEL_MODE_GSI_WRAP wrapping mode.

1.4. Job State Model

As the GRAM service processes a job, the job undergoes a series of state transitions. These states and their meanings follow:

Table 10.1. GRAM Job States

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBUSGRAM_PROTOCOL_JOB_STATE_UNSUBMITTED</td>
<td>Initial job state</td>
</tr>
<tr>
<td>GLOBUSGRAM_PROTOCOL_JOB_STATE_STAGE_IN</td>
<td>Job staging in progress</td>
</tr>
<tr>
<td>GLOBUSGRAM_PROTOCOL_JOB_STATE_PENDING</td>
<td>Job submitted to LRM, awaiting execution</td>
</tr>
<tr>
<td>GLOBUSGRAM_PROTOCOL_JOB_STATE_ACTIVE</td>
<td>Job executing</td>
</tr>
<tr>
<td>GLOBUSGRAM_PROTOCOL_JOB_STATE_SUSPENDED</td>
<td>Job made progress executing but is now suspended</td>
</tr>
<tr>
<td>GLOBUSGRAM_PROTOCOL_JOB_STATE_STAGE_IN</td>
<td>Job staging in progress after job completed</td>
</tr>
<tr>
<td>GLOBUSGRAM_PROTOCOL_JOB_STATE_DONE</td>
<td>Job completed successfully</td>
</tr>
</tbody>
</table>
State | Meaning
--- | ---
GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED | Job was canceled or failed

**Figure 10.1. GRAM State Transitions**

![Diagram of GRAM State Transitions](image)
Chapter 11. Related Documentation

No related documentation links have been determined at this time.
Chapter 12. Internal Components

Internal Components¹

¹ internal-components.html
Glossary

C

certificate  A public key plus information about the certificate owner bound together by the
digital signature of a CA. In the case of a CA certificate, the certificate is self
signed, i.e. it was signed using its own private key.

G

Gatekeeper  A part of GRAM that runs as root and authenticates clients prior to starting the
Job Manager.

grid map file  A file containing entries mapping certificate subjects to local user names. This
file can also serve as a access control list for GSI enabled services and is typi-
cally found in /etc/grid-security/grid-mapfile. For more informa-
tion see the Gridmap section here.

J

Job Manager  A part of GRAM that runs as a local user and interfaces with a Local Resource
Manager for that user.

L

Local Resource Manager (LRM)  A system which controls access to a compute resource, such as a compute clus-
ter or parallel computer. Such systems provide batch execution interfaces, which
GRAM uses to execute jobs. Condor, Portable Batch System, GridEngine are
examples of local resource managers.
See Also Condor, Portable Batch System, Oracle GridEngine.

LRM Adapter  The interface code between a Local Resource Manager and GRAM.
In most cases, this consists of a Perl module that implements the
Globus::GRAM::JobManager class and a Scheduler Event Generator
module.
See Also Local Resource Manager.

P

proxy certificate  A short lived certificate issued using a EEC. A proxy certificate typically has the
same effective subject as the EEC that issued it and can thus be used in its place.
GSI uses proxy certificates for single sign on and delegation of rights to other
entities.

For more information about types of proxy certificates and their compatibility
in different versions of GT, see http://dev.globus.org/wiki/Security/ProxyCert-
Types.
<table>
<thead>
<tr>
<th><strong>R</strong></th>
<th><strong>Term used to describe a GRAM job for GT2 and GT3. (Note: This is not the same as RLS - the Replica Location Service)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S</strong></td>
<td><strong>The Scheduler Event Generator (SEG) is a program which uses scheduler-specific monitoring modules to generate job state change events. Depending on scheduler-specific requirements, the SEG may need to run with privileges to enable it to obtain scheduler event notifications. As such, one SEG runs per scheduler resource. For example, on a host which provides access to both PBS and fork jobs, two SEGs, running at (potentially) different privilege levels will be running. One SEG instance exists for any particular scheduled resource instance (one for all homogeneous PBS queues, one for all fork jobs, etc). The SEG is implemented in an executable called the globus-scheduler-event-generator, located in the Globus Toolkit's libexec directory.</strong></td>
</tr>
</tbody>
</table>
Index

A
apis, 63
   overview, 63

C
compatibility, 4

D
debugging, 92
dependencies, 4

E
errors, 99

F
features, 3

P
platforms, tested, 3

T
troubleshooting, 94, 94
   errors, 94, 94
GT 5.2.2 Migrating Guide for GRAM5
GT 5.2.2 Migrating Guide for GRAM5

Abstract

The following provides available information about migrating from previous versions of the Globus Toolkit.
Table of Contents

1. Migrating GRAM from GT5.0 ................................................................. 1
   1. Admin - Migration Guide .............................................................. 1
   2. User - Migration Guide .............................................................. 1
   3. Developer - Migration Guide ...................................................... 1

2. Migrating GRAM from GT4.2 ................................................................. 2
   1. Admin - Migration Guide .............................................................. 2
   2. User - Migration Guide .............................................................. 2
   3. Developer - Migration Guide ...................................................... 2

3. Migrating GRAM from GT4.0 ................................................................. 3
   1. Admin - Migration Guide .............................................................. 3
   2. User - Migration Guide .............................................................. 3
   3. Developer - Migration Guide ...................................................... 3

Glossary ................................................................................................. 4
Chapter 1. Migrating GRAM from GT5.0

1. Admin - Migration Guide

2. User - Migration Guide

3. Developer - Migration Guide
Chapter 2. Migrating GRAM from GT4.2

The GRAM5 protocol has been designed to be backward compatible with GRAM2 protocol from GT 4.2.x. There is no compatibility between GRAM5 and the GRAM4 protocol.

1. Admin - Migration Guide

1.1. Audit Logging

GRAM5 supports generating audit records the same as GRAM2. It also adds support for a Teragrid-specific Gateway User field in the audit records. The globus_gram_job_manager_auditing package contains the audit database interface code. The globus_gram_job_manager_auditing_setup setup package configures this package. GRAM5 auditing is enabled by using the `-audit-directory` command-line option in the job manager configuration file. For more information about GRAM5 audit support, see the GRAM5 admin guide.

2. User - Migration Guide

2.1. Command-line Tools

GRAM5 provides the `globusrun` program to submit jobs to GRAM5 services. It no longer supports multi-request (duroc or MPI) jobs. GRAM5 does not provide the `globusrun-ws` as it does not support the WSRF protocols.

3. Developer - Migration Guide

3.1. API Changes

The GRAM5 version of the GRAM Client API adds support for receiving protocol extension information in callbacks and responses. All GRAM Client API functions from GRAM2 are provided in the GRAM5 API. The DUROC, DUCT, and Nexus APIs are no longer provided in GRAM5.
Chapter 3. Migrating GRAM from GT4.0

The GRAM5 protocol has been designed to be backward compatible with GRAM2 protocol from GT 4.0.x. There is no compatibility between GRAM5 and the GRAM4 protocol.

1. Admin - Migration Guide

1.1. Audit Logging

GRAM5 supports generating audit records the same as GRAM2. It also adds support for a Teragrid-specific Gateway User field in the audit records. The globus_gram_job_manager_auditing package contains the audit database interface code. The globus_gram_job_manager_auditing_setup setup package configures this package. GRAM5 auditing is enabled by using the –audit-directory command-line option in the job manager configuration file. For more information about GRAM5 audit support, see the GRAM5 admin guide.

2. User - Migration Guide

2.1. Command-line Tools

GRAM5 provides the globusrun program to submit jobs to GRAM5 services. It no longer supports multi-request (duroc or MPI) jobs. GRAM5 does not provide the globusrun-ws as it does not support the WSRF protocols.

3. Developer - Migration Guide

3.1. API Changes

The GRAM5 version of the GRAM Client API adds support for receiving protocol extension information in callbacks and responses. All GRAM Client API functions from GRAM2 are provided in the GRAM5 API. The DUROC, DUCT, and Nexus APIs are no longer provided in GRAM5.
Glossary
GT 5.2.2 GRAM5: Quality Profile

Table of Contents
1. Test Reports ........................................................................................................................................ 1
2. Test Coverage Reports ......................................................................................................................... 1
3. Known Problems in GRAM5 ................................................................................................................ 1
4. Fixed Bugs for GRAM5 ......................................................................................................................... 2

1. Test Reports
• GT 5.2.2 GRAM Protocol Test Report
• GT 5.2.2 GRAM Client Test Report
• GT 5.2.2 Condor Test Report
• GT 5.2.2 Fork (Poll) Test Report
• GT 5.2.2 Fork (SEG) Test Report
• GT 5.2.2 PBS (Poll) Test Report
• GT 5.2.2 PBS (SEG) Test Report
• GT 5.2.2 SGE (Poll) Test Report
• GT 5.2.2 SGE (SEG) Test Report

2. Test Coverage Reports
• http://www.mcs.anl.gov/~bester/gram5/coverage/5.2.2

3. Known Problems in GRAM5
• GT-48: Held Condor jobs should be reported as SUSPENDED
• GT-53: RSL eval doesn't indicate what symbol was not found

1 protocoltest.html
2 clienttest.html
3 condornone.html
4 forkpoll.html
5 forkseg.html
6 pbspoll.html
7 pbsseg.html
8 sgepoll.html
9 sgeseq.html
10 http://jira.globus.org/browse/GT-48
11 http://jira.globus.org/browse/GT-53
4. Fixed Bugs for GRAM5

- GT-149\(^\text{12}\): Memory leaks in globus-job-manager
- GT-155\(^\text{13}\): Job manager deletes job dir sometimes
- GT-163\(^\text{14}\): Condor fake-SEG loses track of job
- GT-198\(^\text{15}\): globusrun crashes when authentication fails for status check
- GT-199\(^\text{16}\): GRAM audit checks result username incorrectly
- GT-209\(^\text{17}\): job manager crash in query
- GT-212\(^\text{18}\): Missing debian packages
- GT-236\(^\text{19}\): gram audit makefile has missing parameter to mkdir
- GT-252\(^\text{20}\): Missing dependency in gass cache program
- GT-253\(^\text{21}\): gatekeeper and job manager don't build on hurd

\(^\text{12}\) http://jira.globus.org/browse/GT-149
\(^\text{13}\) http://jira.globus.org/browse/GT-155
\(^\text{14}\) http://jira.globus.org/browse/GT-163
\(^\text{15}\) http://jira.globus.org/browse/GT-198
\(^\text{16}\) http://jira.globus.org/browse/GT-199
\(^\text{17}\) http://jira.globus.org/browse/GT-209
\(^\text{18}\) http://jira.globus.org/browse/GT-212
\(^\text{19}\) http://jira.globus.org/browse/GT-236
\(^\text{20}\) http://jira.globus.org/browse/GT-252
\(^\text{21}\) http://jira.globus.org/browse/GT-253
1. Component Overview

The Grid Resource Allocation and Management (GRAM5) component is used to locate, submit, monitor, and cancel jobs on Grid computing resources. GRAM5 is not a Local Resource Manager, but rather a set of services and clients for communicating with a range of different batch/cluster job schedulers using a common protocol. GRAM5 is meant to address a range of jobs where reliable operation, stateful monitoring, credential management, and file staging are important.

2. Feature summary

New Features new since 5.2.1:

- Improved memory management and process management.
- Improved scalability and reliability

Other Standard Supported Features

- Remote job execution and management
- Uniform and flexible interface to local resource managers
- File staging before and after job execution
- File and directory clean up after job termination
- Service auditing for each submitted

Removed Features

- Condor SEG module is no longer included. Its functionality has been moved into the core of the job manager program.
3. Summary of Changes in GRAM5

3.1. New Features: GRAM5

None.

3.2. Improvements: GRAM5

- GT-154: Kill off perl processes when idle
- GT-185: globus-personal-gatekeeper creates too-long paths on MacOS
- GT-205: gatekeeper should log a message when it exits due to the presence of /etc/nologin
- GT-224: Manage GRAM execution per client host for scalability for different clients

4. Fixed Bugs for GRAM5

- GT-149: Memory leaks in globus-job-manager
- GT-155: Job manager deletes job dir sometimes
- GT-163: Condor fake-SEG loses track of job
- GT-198: globusrun crashes when authentication fails for status check
- GT-199: GRAM audit checks result username incorrectly
- GT-209: job manager crash in query
- GT-212: Missing debian packages
- GT-236: gram audit makefile has missing parameter to mkdir
- GT-253: Missing dependency in gass cache program
- GT-253: gatekeeper and job manager don't build on hurd

5. Known Problems in GRAM5

- GT-48: Held Condor jobs should be reported as SUSPENDED

1 http://jira.globus.org/browse/GT-154
2 http://jira.globus.org/browse/GT-185
3 http://jira.globus.org/browse/GT-205
4 http://jira.globus.org/browse/GT-224
5 http://jira.globus.org/browse/GT-149
6 http://jira.globus.org/browse/GT-155
7 http://jira.globus.org/browse/GT-163
8 http://jira.globus.org/browse/GT-198
9 http://jira.globus.org/browse/GT-199
10 http://jira.globus.org/browse/GT-209
11 http://jira.globus.org/browse/GT-212
12 http://jira.globus.org/browse/GT-236
13 http://jira.globus.org/browse/GT-253
14 http://jira.globus.org/browse/GT-253
15 http://jira.globus.org/browse/GT-48
• GT-53\textsuperscript{16}: RSL eval doesn't indicate what symbol was not found

# 6. Technology dependencies

GRAM depends on the following GT components:

• Globus Common
• GSI C
• GridFTP server

# 7. Tested platforms

Tested platforms for GRAM5:

• Linux
  • CentOS 4 x86_64
  • CentOS 5, 6 x86_64, i386
  • Fedora 15, 16, 17 x86_64, i386
  • Red Hat Enterprise Linux 5, 6 x86_64, i386
  • Scientific Linux 5, 6 x86_64, i386
  • Debian 6, 7 (testing) x86_64, i386
  • Ubuntu 10.04LTS, 10.10, 11.04, 11.10, 12.04LTS x86_64, i386
• Mac OS X
  • Mac OS X 10.7 (Lion)
• Solaris
  • Solaris 11

# 8. Backward compatibility summary

Protocol changes in GRAM since GT4 series:

• The GRAM5 service uses a superset of the GRAM2 protocol for communication between the client and service. The extensions supported in GRAM5 are implemented in such a way that they are ignored by GRAM2 services or clients. These extensions provide improved error messages and version detection.

• GRAM5 does not support task co-allocation using DUROC and its related protocols. Jobs submitted using DUROC directives will fail.

• GRAM5 does not support file streaming. The standard output and standard error streams are sent after the job completes instead of during execution. As a special case, support for the Condor grid monitor program implements a small subset of the streaming capabilities of GRAM2 in GT 4.2.x.

\textsuperscript{16}http://jira.globus.org/browse/GT-53
9. Associated Standards

None

10. For More Information

See GRAM5 for more information about this component.

Glossary

L

Local Resource Manager (LRM)  A system which controls access to a compute resource, such as a compute cluster or parallel computer. Such systems provide batch execution interfaces, which GRAM uses to execute jobs. Condor, Portable Batch System, GridEngine are examples of local resource managers.

See Also Condor, Portable Batch System, Oracle GridEngine.