

GT 5.0.0 GRAM5 : System Administrator's Guide

GT 5.0.0 GRAM5 : System Administrator's Guide

Introduction

This guide contains advanced configuration information for system administrators working with GRAM5. It provides references to information on procedures typically performed by system administrators, including installation, configuring, deploying, and testing the installation. It also describes additional prerequisites and host settings necessary for GRAM5 operation. Readers should be familiar with the [Key Concepts](#) and [Implementation Approach](#) for GRAM5 to understand the motivation for and interaction between the various deployed components.

Important

The information in this GRAM5 Admin Guide is in addition to the basic Globus Toolkit prerequisite, overview, installation, security configuration instructions in the [Installing GT 5.0.0](#). Read through this guide before continuing!

Table of Contents

1. Building and Installing	1
1. Local Resource Manager	1
2. LRM Adapters	1
2. Configuring	2
1. Typical Configuration	2
2. Non-default Configuration	3
3. Configuration Details	3
3. Deploying	6
1. Deploying GRAM5 via inetd or xinetd	6
2. Deploying GRAM5 as a daemon	7
4. Running the SEG	8
1. Starting the SEG	8
2. Stopping the SEG	8
5. Scalability and Performance Recommendations	9
1. Server-side Recommendations	9
6. Audit Logging	10
1. Overview	10
2. Audit and Accounting Records	10
3. For More Information	11
4. Configuration	11
5. Audit Database Interface	11
7. Testing	13
1. GRAM protocol tests	13
2. GRAM client tests	13
3. GRAM Job Manager Tests	14
8. Security Considerations	16
9. Admin Debugging	17
10. GRAM5 Admin Programs	18
globus-gram-audit	19
globus-job-manager-event-generator	20
globus-job-manager	21
11. Troubleshooting	26
1. Troubleshooting tips	26
2. Errors	27
12. Usage statistics collection by the Globus Alliance	37
1. GRAM5-specific usage statistics	37
Glossary	40
Index	41

List of Tables

1.1. Supported LRM Adapters	1
2.1. LRM Adapter Make targets	2
9.1. GRAM5 Log Levels	17
11.1. GRAM5 Errors	28

List of Examples

2.1. Installing PBS LRM Adapter	3
2.2. Configuring GRAM5 to use PBS by default	5
2.3. Disabling the PBS LRM Adapter	5
4.1. Starting and Stopping the SEG	8
7.1. Running the GRAM Protocol Test Suite	13
7.2. Running the GRAM Client Test Suite	14
7.3. Running the GRAM Job Manager Test Suite	15

Chapter 1. Building and Installing

GRAM5 is built and installed as part of a default GT 5.0.0 installation. For basic installation instructions, see [Installing GT 5.0.0](#).

1. Local Resource Manager

GRAM5 depends on a local mechanism for starting and controlling jobs. GRAM5 includes a fork *local resource manager* which requires no special software to execute jobs on the local host. GRAM5 can also be configured to use additional batch facilities and schedulers such as condor, torque, or LSF. Install and configure any local resource managers you intend to use prior to configuring GRAM5.

2. LRM Adapters

GRAM5 depends on LRM adapters to execute jobs described by GRAM5 *RSL* documents on a resource managed by a LRM.

LRM adapters included in the GT 5.0.0 release are:

Table 1.1. Supported LRM Adapters

LRM Adapter Name	LRM Supported
fork	Unscheduled local execution
pbs	Torque ¹
condor	Condor ²
lsf	LSF ³
SGE	Grid Engine ⁴



Note

The pbs LRM adapter may work with other systems that implement POSIX batch environment services.

For configuration details, see "Configuring LRM adapters" in the [Configuring](#) section.

¹ <http://www.clusterresources.com/products/torque-resource-manager.php>

² <http://www.cs.wisc.edu/condor/>

³ <http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/>

⁴ <http://gridengine.sunsource.net/>

Chapter 2. Configuring

1. Typical Configuration

1.1. Configuring LRM Adapters

Three GRAM5 LRM adapters included in the GT 5.0.0 installer besides the default `fork` LRM adapter. These are installed by using the following make rules from the installer directory:

Table 2.1. LRM Adapter Make targets

LRM	Installer make target
Fork	<code>gram5-fork</code>
Condor	<code>gram5-condor</code>
LSF	<code>gram5-lsf</code>
PBS	<code>gram5-pbs</code>
SGE	<code>gram5-sge</code>

Example 2.1. Installing PBS LRM Adapter

```
% make gram5-pbs
cd gpt && OBJECT_MODE=32 ./build_gpt
build_gpt =====> installing GPT into /opt/globus
build_gpt =====> building /usr/src/globus/gpt/support/Compress-Zlib-1.21
build_gpt =====> building /usr/src/globus/gpt/support/IO-Zlib-1.01
build_gpt =====> building /usr/src/globus/gpt/support/makepatch-2.00a
build_gpt =====> building /usr/src/globus/gpt/support/Archive-Tar-0.22
build_gpt =====> building /usr/src/globus/gpt/support/PodParser-1.18
build_gpt =====> building /usr/src/globus/gpt/support/Digest-MD5-2.20
build_gpt =====> building /usr/src/globus/gpt/packaging_tools
/opt/globus/sbin/gpt-build -srcdir=source-trees/core/source gcc64dbg
gpt-build =====> Changing to /usr/src/globus/source-trees/core/source
gpt-build =====> BUILDING FLAVOR gcc64dbg
```

additional lines omitted

```
% make install
if [ ! -L /opt/globus/etc/globus_packages ]; then \
    cd /opt/globus/etc/; \
    ln -sf gpt/packages globus_packages; \
fi; \
/opt/globus/sbin/gpt-postinstall
running /opt/globus/setup/globus/setup-globus-common..[ Changing to /opt/globus/setup/glob
creating globus-sh-tools-vars.sh
creating globus-script-initializer
creating Globus::Core::Paths
checking globus-hostname
Done
```

additional lines omitted

The LRM make targets will check all dependencies they need and try to build them if they are not yet installed.

2. Non-default Configuration

2.1. Authorization

TODO: Discussion of Job Manager Callout

3. Configuration Details

3.1. LRM-Specific Scheduler Event Generator configuration files

In addition to the service configuration described above, there are LRM-specific configuration files for the Scheduler Event Generator modules. These files consist of name=value pairs separated by newlines. These files are:

3.1.1. \$GLOBUS_LOCATION/etc/globus-fork.conf

Configuration for the Fork *SEG* module implementation. The attributes names for this file are:

`log_path` Path to the SEG Fork log (used by the globus-fork-starter and the SEG). The value of this should be the path to a world-writable file. The default value for this created by the Fork setup package is `$GLOBUS_LOCATION/var/globus-fork.log`. This file must be readable by the account that the SEG is running as.

3.1.2. \$GLOBUS_LOCATION/etc/globus-condor.conf

Configuration for the *Condor* SEG module implementation. The attributes names for this file are:

`log_path` Path to the SEG Condor log (used by the Globus::GRAM::JobManager::condor perl module and Condor SEG module. The value of this should be the path to a world-readable and world-writable file. The default value for this created by the Fork setup package is `$GLOBUS_LOCATION/var/globus-condor.log`

3.1.3. \$GLOBUS_LOCATION/etc/globus-pbs.conf

Configuration for the *PBS* SEG module implementation. The attributes names for this file are:

`log_path` Path to the SEG PBS logs (used by the Globus::GRAM::JobManager::pbs perl module and PBS SEG module. The value of this should be the path to the directory containing the server logs generated by PBS. For the SEG to operate, these files must have file permissions such that the files may be read by the user the SEG is running as.

3.1.4. \$GLOBUS_LOCATION/etc/globus-lsf.conf

Configuration for the *LSF* SEG module implementation. The attributes names for this file are:

`log_path` Path to the SEG LSF log directory. This is used by the LSF SEG module. The value of this should be the path to the directory containing the server logs generated by LSF. For the SEG to operate, these files must have file permissions such that the files may be read by the user the SEG is running as.

3.1.5. \$GLOBUS_LOCATION/etc/globus-sge.conf

Configuration for the *SGE* SEG module implementation. The attributes names for this file are:

`log_path` Path to the SGE reporting file. used by the SGE SEG module. The value of this should be the path to the cell's `accounting` file generated by SGE. For the SEG to operate, this file must have file permissions such that the files may be read by the user the SEG is running as and the ARCO database uploader must not be running.

3.2. Enabling an LRM-Specific SEG module to use with GRAM5

In most situations, the SEG interface provides a more efficient way to monitor jobs than the poll method. This is configured by adding the `-seg-module LRM` command-line option to the job manager configuration in `$GLOBUS_LOCATION/etc/grid-services/jobmanager-LRM`.

3.3. Defining a default local resource manager

A client can submit a job without specifying the local resource manager (LRM) that should execute the job.

The default job manager for a particular host is determined by the contents of the file `$GLOBUS_LOCATION/etc/grid-services/jobmanager`. By default, this file is a symbolic link to the service definition for the first LRM adapter that is installed. To change the default LRM, change the link to point to a different LRM service definition.

Example 2.2. Configuring GRAM5 to use PBS by default

```
% cd $GLOBUS_LOCATION/etc/grid-services
% rm jobmanager
% ln -s jobmanager-pbs jobmanager
```

3.4. Disabling an already installed local resource manager adapter

When the GRAM5 gatekeeper accepts a new connection, it checks the contents of the service directory to determine what services are configured. To disable a service, remove the service entry from the service directory. The service entry can be recreated by running the `gpt-postinstall -force` command.

Example 2.3. Disabling the PBS LRM Adapter

```
% cd $GLOBUS_LOCATION/etc/grid-services
% rm jobmanager-pbs
```

If the PBS LRM was the default job manager, then the symlink `$GLOBUS_LOCATION/etc/grid-services/jobmanager` will no longer point to a valid file, and there will be no default job manager configured. Clients must explicitly choose some other LRM which is configured.

Chapter 3. Deploying

GRAM5 is installed as part of a standard toolkit installation. By default the `fork` LRM interface is installed and configured to use the `poll` interface.

In order to run the service, the `globus-gatekeeper` program must be started on a service node. This may be done in two supported ways, either via a super-server such as `inetd` or `xinetd`, or as a stand-alone daemon process.

1. Deploying GRAM5 via `inetd` or `xinetd`

To deploy GRAM5 to be started by `inetd`, define a service entry for the `gsigatekeeper` service in `/etc/services` file if it is not already present. The ICANN-assigned port number for the service is 2119, so add an entry like:

```
gsigatekeeper 2119/tcp
```

1.1. Deploying GRAM5 via `inetd`

To deploy GRAM5 to be started via `inetd`, modify `/etc/inetd.conf` (or your operating-system specific path to this file) by adding the following entry (on one line):

```
gsigatekeeper stream tcp nowait root /usr/bin/env env LD_LIBRARY_PATH=GLOBUS_LOCATION/lib
```

After doing so, run the operating-system specific command to have `inetd` reload its configuration.

Important

This example assumes that the `LD_LIBRARY_PATH` environment variable is needed to be able to run the GRAM5 programs and scripts. This is operating-system specific and may need to be changed. Consult the documentation for the operating system's runtime linker in the case of runtime problems.

The `GLOBUS_LOCATION` string in the above example must be replaced with the actual path where GT has been installed.

Note

When deploying GRAM5 via `xinetd`, be sure to include the command-line option `-inetd` in the `gatekeeper` configuration file; otherwise, the `gatekeeper` will log a warning each time it is started.

1.2. Deploying GRAM5 via `xinetd`

To deploy GRAM5 to be started via `xinetd`, create a file in the `/etc/xinetd.d/` directory (or the operating-system specific path to `xinetd` configuration files) called `gsigatekeeper` with the following contents:

```
service gsigatekeeper
{
    socket_type = stream
    protocol = tcp
    wait = no
    user = root
    env = LD_LIBRARY_PATH=GLOBUS_LOCATION/lib
    server = GLOBUS_LOCATION/sbin/globus-gatekeeper
```

```
server_args = -conf GLOBUS_LOCATION/etc/globus-gatekeeper.conf
disable = no
}
```

After doing so, run the operating-system specific command to have xinetd reload its configuration.

Important

This example assumes that the `LD_LIBRARY_PATH` environment variable is needed to be able to run the GRAM5 programs and scripts. This is operating-system specific and may need to be changed. Consult the documentation for the operating system's runtime linker in the case of runtime problems.

The `GLOBUS_LOCATION` string in the above example must be replaced with the actual path where GT has been installed.

Note

When deploying GRAM5 via xinetd, be sure to include the command-line option `-inetd` in the gatekeeper configuration file; otherwise, the gatekeeper will log a warning each time it is started.

2. Deploying GRAM5 as a daemon

To deploy GRAM5 to be started as a daemon, run the **globus-gatekeeper** at boot time. This may be done in an init script, via cron, or other system-specific methods. The typical command-line to run is

```
globus-gatekeeper -conf $GLOBUS_LOCATION/etc/globus-gatekeeper.conf
```

Note

When deploying GRAM5 as a daemon, be sure to include the command-line option `-f` in the gatekeeper configuration file; otherwise, the gatekeeper will log a warning each time it is started.

Chapter 4. Running the SEG

1. Starting the SEG

GRAM5 can be configured to use the **globus-job-manager-event-generator** program to monitor job state changes. This is often more efficient than using a LRM adapter's `poll` method. This program is configured when the LRM-specific bundle is installed. However, by default, the job manager does not use the SEG. It must be explicitly enabled by adding the `-seg-module LRM` option to the job manager configuration.

To start the **globus-job-manager-event-generator** program, add the following command to your system init scripts or crontab to be run at boot time:

```
$GLOBUS_LOCATION/sbin/globus-job-manager-event-generator -scheduler LRM -background -pidfile
```

This will start the **globus-job-manager-event-generator** to monitor jobs for the resource named by `LRM` in the background. The process id of the command will be written to the file named by `PIDFILE`, so that processes can check if it is running and kill it if necessary.

Important

If the job manager is configured to use a SEG module but the **globus-job-manager-event-generator** is not running for that LRM, jobs will appear to hang. It is important that the program be running whenever GRAM5 jobs might be run.

2. Stopping the SEG

To stop the SEG, kill the **globus-job-manager-event-generator** process. The `-pidfile` option makes it easy to know which process to kill. When the SEG terminates, it will remove that file.

Example 4.1. Starting and Stopping the SEG

This example shows how to start and stop the SEG using the command-line options described above.

```
% globus-job-manager-event-generator -scheduler pbs -background -pidfile $GLOBUS_LOCATION/  
Running in background (78258)  
% kill `cat $GLOBUS_LOCATION/var/globus-job-manager-seg-pbs.pid`
```

Chapter 5. Scalability and Performance Recommendations

This document includes recommendations for increasing the scalability and performance of GRAM5 in a Grid.

1. Server-side Recommendations

1.1. Filesystem related settings

1. The GRAM5 service stores job state for crash recovery on disk. By default, the `$GLOBUS_LOCATION/tmp/gram_job_state` directory is used for these files. If this path is located on a distributed file system mount, locking and updating the job state files can negatively effect performance.

To configure GRAM5 to use a local disk for job state files, modify `$GLOBUS_LOCATION/etc/globus-job-manager.conf` or `$GLOBUS_LOCATION/etc/grid-services/jobmanager-LRM` so that the argument to the `-state-file-dir` is a local directory path. That directory must be world writable with the "sticky bit" set (mode 1777).

1.2. LRM-related settings

1. The GRAM5 service can use two different interfaces to obtain job state changes, polling and using the SEG. The default method is to poll the LRM via the GRAM5 script interface. This polling method is often less efficient than the SEG method and results in a higher load on the GRAM5 service node, even when all managed jobs are waiting in the queue for execution.

The other method, uses a program implementing the SEG interface to generate LRM events which can be stored in a log file for job managers run by many different users to process. When this is used, the multiple job managers may detect job state changes without having to directly interact with the LRM.

To enable the SEG to monitor jobs for a particular LRM, install and configure the LRM-specific `gram5` bundle, run the **`globus-job-manager-event-generator`** program on a node which can access the LRM interfaces needed by the LRM-specific SEG module, and configure the LRM-specific service instance to use the SEG to monitor jobs for state changes.

Chapter 6. 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 PBS, LSF, or Condor.

Note

Remember, GRAM is not a local resource manager but rather a protocol engine for communicating with a range of different local resource managers using a standard message format.

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:

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

The audit record of each job contains the following data:

- **job_grid_id**: String representation of the resource EPR
- **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. A case study for TeraGrid can be read [here](#)¹, which also includes more information about how to use this data to get accounting information of a job, query the audit database for information via a Web Services interface, etc.

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` to the job manager configuration in either `$GLOBUS_LOCATION/etc/globus-job-manager.conf` to enable it for all job manager services, or in `$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 an 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_gram_job_manager_auditing_setup_scripts` setup package via `gpt-postinstall`. This setup script

¹ http://www.teragridforum.org/mediawiki/index.php?title=GRAM5_Audit

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.

5.1. Audit Configuration File

The auditing configuration file consists of a series of line-oriented records that define various configuration attributes used by the `globus-gram-audit` program. This file can be edited by hand or using the `$GLOBUS_LOCATION/setup/globus/setup-globus-gram-auditing` program. That program is run automatically with the default values described below when `gpt-postinstall` is run after installing the audit setup package.

The values which may be defined in the configuration file are:

Attribute Name	Values	Default when not specified on the setup command-line
DRIVER	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.	SQLite
DATABASE	The DBI data source specification to contact the audit database.	dbname= <i>GLOBUS_LOCATION</i> /var/gram_audit_database/gram_audit.db
USERNAME	Username to authenticate as to the database	
PASSWORD	Password to use to authenticate with the database	
AUDITVERSION	Version of the audit database table schemas to use. May be 1 or 1TG for this version of the software.	1

Thus, the default configuration file when `GLOBUS_LOCATION` is `/opt/globus` is

```
DRIVER:SQLite
DATABASE:dbname=/opt/globus/var/gram_audit_database/gram_audit.db
USERNAME:
PASSWORD:
AUDITVERSION:1
```

Chapter 7. Testing

There are three test suites available to verify that the GRAM5 client and service are installed corrected.

1. GRAM protocol tests

The GRAM protocol test suite tests the implementation of the GRAM protocol library which is used by the job manager and GRAM clients to process messages. The following examples shows how to run the test suite.

Example 7.1. Running the GRAM Protocol Test Suite

```
% cd $GLOBUS_LOCATION/test/globus_gram_protocol_test
% grid-proxy-init
Your identity: /DC=org/O=example/OU=grid/CN=Joe User
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Your proxy is valid until: Thu Nov 12 23:28:05 2009
% ./TESTS.pl
globus-gram-protocol-allow-attach-test.....ok
globus-gram-protocol-error-test.....ok
globus-gram-protocol-io-test.....ok
globus-gram-protocol-pack-test.....ok
pack-with-extensions-test.....ok
create-extensions-test.....ok
unpack-message-test.....ok
unpack-with-extensions-test.....ok
unpack-job-request-reply-with-extensions-test....ok
unpack-status-reply-with-extensions-test.....ok
All tests successful.
Files=10, Tests=42, 1 wallclock secs ( 0.37 cusr + 0.24 csys = 0.61 CPU)
```

2. GRAM client tests

The GRAM client test suite tests the interactions between the GRAM client API implementation and the job manager service. These tests include authentication, callback, signal, job, and cancellation tests.

Example 7.2. Running the GRAM Client Test Suite

```
% cd $GLOBUS_LOCATION/test/globus_gram_client_test
% grid-proxy-init
Your identity: /DC=org/O=example/OU=grid/CN=Joe User
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Your proxy is valid until: Thu Nov 12 23:28:05 2009
% ./TESTS.pl
globus-gram-client-activate-test.....ok
globus-gram-client-callback-contact-test.....ok
globus-gram-client-cancel-test.....ok
globus-gram-client-nonblocking-register-test....ok 1/4Failed submitting job request because
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 entity (/DC=org/O=example/
.
globus-gram-client-nonblocking-register-test....ok
globus-gram-client-refresh-credentials-test.....ok
globus-gram-client-register-test.....ok
globus-gram-client-register-callback-test.....ok 1/4Failed submitting job request because
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 entity (/DC=org/O=example/
.
globus-gram-client-register-callback-test.....ok
globus-gram-client-register-cancel-test.....ok
globus-gram-client-ping-test.....ok
globus-gram-client-status-test.....Made 3961 calls to status in 60.416390 sec
globus-gram-client-status-test.....ok
globus-gram-client-two-phase-commit-test.....ok
globus-gram-client-register-ping-test.....ok
globus-gram-client-stdio-size-test.....job manager returned 0 (Success) when I ex
globus-gram-client-stdio-size-test.....ok
version-test.....ok
All tests successful.
Files=14, Tests=33, 791 wallclock secs (32.10 cusr + 5.34 csys = 37.44 CPU)
```



Note

Some of the test cases display messages that look like errors when running. This is to be expected. The only concern should be the final lines indicating if the tests are successful or not.



Note

By default, this suite uses tests against a personal gatekeeper running the `fork` LRM. To run against another service, set the environment variable `CONTACT_STRING` to the service contact string prior to starting the tests.

3. GRAM Job Manager Tests

The GRAM job manager test suite tests the features provided by the LRM scripts, including detecting failures, staging files, and submitting different types of jobs. The following example shows how to run the job manager tests.

Example 7.3. Running the GRAM Job Manager Test Suite

```
% cd $GLOBUS_LOCATION/test/globus_gram_job_manager_test
% grid-proxy-init
Your identity: /DC=org/O=example/OU=grid/CN=Joe User
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Your proxy is valid until: Thu Nov 12 23:28:05 2009
% ./TESTS.pl
job-manager-script-test.....ok
globus-gram-job-manager-stdio-test.....ok
globus-gram-job-manager-submit-test.....ok
globus-gram-job-manager-failure-test.....ok
globus-gram-job-manager-rsl-size-test....ok
globus-gram-job-manager-user-test.....ok
All tests successful.
Files=6, Tests=137, 200 wallclock secs (32.10 cusr + 5.34 csys = 37.44 CPU)
```



Note

This test requires a GridFTP server to be running on the host running the test suite. If one is not running, then the following test cases will fail: `globus-gram-job-manager-stdio-test` subtests 2, 6, 9, 12, 13, 16, 17, 18, 19.



Note

By default, this suite uses tests against a personal gatekeeper running the `fork` LRM. To run against another service, set the environment variable `CONTACT_STRING` to the service contact string prior to starting the tests.

Chapter 8. Security Considerations

No special security considerations exist at this time.

Chapter 9. Admin Debugging

By default, GRAM5 logs errors to `$HOME/gram_YYYYMMDD.log` where `YYYYMMDD` is the time of the log event in GMT. The log file format conforms to [CEDPS Logging Best Practices](http://www.cedps.net/index.php/LoggingBestPractices)¹. GRAM5 log files are governed by the log levels defined in the job manager configuration file. The log levels available are defined below:

Table 9.1. GRAM5 Log Levels

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

To enable logging for GRAM5, modify `$GLOBUS_LOCATION/etc/globus-job-manager.conf` so that it has either `-stdio-log PATH` to log to a file or `-enable-syslog` to log using the syslog service. To select log levels, add `-log-levels "LEVELS"` to the configuration file. The `LEVELS` string can contain any of the log levels mentioned above joined by the vertical bar character '|.

¹ <http://www.cedps.net/index.php/LoggingBestPractices>

Chapter 10. GRAM5 Admin Programs

Name

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

`globus-gram-audit` [--conf *CONFIG_FILE*] [--check] [--delete] [--audit-directory *AUDITDIR*]

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:

<code>--conf</code> <i>CONFIG_FILE</i>	Use <i>CONFIG_FILE</i> instead of the default from the configuration file for audit database configuration.
<code>--check</code>	Check whether the insertion of a record was successful by querying the database after inserting the records. This is used in tests.
<code>--delete</code>	Delete audit records from the database right after inserting them. This is used in tests to avoid filling the database with test records.
<code>--audit-directory</code> <i>DIR</i>	Look for audit records in <i>DIR</i> , 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.
<code>--query</code> <i>SQL</i>	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.

FILES

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

<code>etc/globus-gram-job-manager.conf</code>	GRAM5 job manager configuration. It includes the default path to the audit directory
<code>etc/globus-gram-audit.conf</code>	Audit configuration. It includes the information needed to contact the audit database.

Name

globus-job-manager-event-generator -- Create LRM-independent SEG files for the job manager to use

globus-job-manager-event-generator [-help] {-scheduler *LRM*} [-background] [-pidfile *PIDPATH*]

Description

The **globus-job-manager-event-generator** program is a utility which uses LRM-specific SEG parsers to generate a LRM-independent log file that a job manager instance can use to process job status change events. This program runs independently of all **globus-job-manager** instances so that only one process needs to deal with the LRM interface. The **globus-job-manager-event-generator** program can be run as a privileged user if required to interface with the LRM.

In order for **globus-job-manager-event-generator** to handle events for a particular LRM, the `globus_scheduler_event_generator_job_manager_setup` setup package must be configured after the LRM-specific setup package has been run. This can be forced by **gpt-postinstall -force** or running the command **cd \$GLOBUS_LOCATION/setup/globus; ./setup-seg-job-manager.pl**.

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

- help Print command-line option summary and exit.
- scheduler *LRM* Process events for the local resource manager named by *LRM*.
- background Run **globus-job-manager-event-generator** as a background process. It will fork a new process, print out its process ID and then the original process will terminate.
- pidfile *PIDPATH* Write the process ID of an instance of **globus-job-manager-event-generator** to the file named by *PIDPATH*. This file can be used to kill or monitor the **globus-job-manager-event-generator** process.

Files

- globus-job-manager-seg.conf Configuration file for **globus-job-manager-event-generator**. Each line consists of a string of the form `LRM_log_path=PATH`, which indicates the directory containing LRM-independent format SEG log files for the LRM. This file is created by the running the `globus_scheduler_event_generator_job_manager_setup` setup package.

See Also

globus-scheduler-event-generator(8), globus-job-manager(8)

Name

globus-job-manager -- Execute and monitor jobs

```
globus-job-manager {-type LRM} [-conf CONFIG_PATH] [-help] [-globus-host-manufacturer MANUFACTURER]
[-globus-host-cputype CPUTYPE] [-globus-host-osname OSNAME] [-globus-host-osversion OSVERSION] [-globus-
gatekeeper-host HOST] [-globus-gatekeeper-port PORT] [-globus-gatekeeper-subject SUBJECT] [-home GLOBUS_LOC-
ATION] [-target-globus-location TARGET_GLOBUS_LOCATION] [-condor-arch ARCH] [-condor-os OS] [-history
HISTORY_DIRECTORY] [-scratch-dir-base SCRATCH_DIRECTORY] [-enable-syslog] [-stdio-log LOG_DIRECTORY]
[-log-levels LEVELS] [-state-file-dir STATE_DIRECTORY] [-globus-tcp-port-range PORT_RANGE] [-x509-cert-dir
TRUSTED_CERTIFICATE_DIRECTORY] [-cache-location GASS_CACHE_DIRECTORY] [-k] [-extra-envvars
VAR=VAL, . . .] [-seg-module SEG_MODULE] [-audit-directory AUDIT_DIRECTORY] [-globus-toolkit-version
TOOLKIT_VERSION] [-disable-streaming] [-disable-usagestats] [-usagestats-target TARGET] [-service-tag SER-
VICE_TAG]
```

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 <i>LRM</i>	Execute jobs using the local resource manager named <i>LRM</i> .
-conf <i>CONFIG_PATH</i>	Read additional command-line arguments from the file <i>CONFIG_PATH</i> . If present, this must be the first command-line argument to the globus-job-manager program.
-globus-host-manufacturer <i>MANUFACTURER</i>	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 <i>MANUFACTURER</i>
-globus-host-cputype <i>CPU-TYPE</i>	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 <i>CPUTYPE</i>
-globus-host-osname <i>OS-NAME</i>	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 <i>OSNAME</i>
-globus-host-osversion <i>OSVERSION</i>	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 <i>OSVERSION</i>
-globus-gatekeeper-host <i>HOST</i>	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 <i>HOST</i>

<code>-globus-gatekeeper-port</code> <i>PORT</i>	Indicate the TCP port number of gatekeeper to which jobs are submitted to. This parameter sets the value of the <code>\$(GLOBUS_GATEKEEPER_PORT)</code> RSL substitution to <i>PORT</i>
<code>-globus-gatekeeper-subject</code> <i>SUBJECT</i>	Indicate the X.509 identity of the gatekeeper to which jobs are submitted to. This parameter sets the value of the <code>\$(GLOBUS_GATEKEEPER_SUBJECT)</code> RSL substitution to <i>SUBJECT</i>
<code>-home</code> <i>GLOBUS_LOCATION</i>	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.
<code>-target-globus-location</code> <i>TARGET_GLOBUS_LOCATION</i>	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 <code>\$(GLOBUS_LOCATION)</code> RSL substitution to <i>TARGET_GLOBUS_LOCATION</i>
<code>-history</code> <i>HISTORY_DIRECTORY</i>	Configure the job manager to write job history files to <i>HISTORY_DIRECTORY</i> . These files are described in the FILES section below.
<code>-scratch-dir-base</code> <i>SCRATCH_DIRECTORY</i>	Configure the job manager to use <i>SCRATCH_DIRECTORY</i> as the default scratch directory root if a relative path is specified in the job RSL's <code>scratch_dir</code> attribute.
<code>-enable-syslog</code>	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.
<code>-stdio-log</code> <i>LOG_DIRECTORY</i>	Configure the job manager to write log messages to files in the <i>LOG_DIRECTORY</i> directory. Files will be named <i>LOG_DIRECTORY/gram_YYYYMM-DD.log</i> . Logging is further controlled by the argument to the <code>-log-levels</code> parameter described below. The <i>LOG_DIRECTORY</i> value can include variables derived from the job manager environment using the same syntax as RSL substitutions. For example, <code>-stdio-log \$(HOME)</code> would cause each user's logs to be stored in their individual home directories.
<code>-log-levels</code> <i>LEVELS</i>	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 <code> </code> character. The default value of logging when enabled is <code>FATAL ERROR</code> .
<code>-state-file-dir</code> <i>STATE_DIRECTORY</i>	Configure the job manager to write state files to <i>STATE_DIRECTORY</i> . If not specified, the job manager uses the default of <code>\$(GLOBUS_LOCATION)/tmp/gram_job_state/</code> . This directory must be writable by all users and be on a file system which supports POSIX advisory file locks.
<code>-globus-tcp-port-range</code> <i>PORT_RANGE</i>	Configure the job manager to restrict its TCP/IP communication to use ports in the range described by <i>PORT_RANGE</i> . This value is also made available in the job environment via the <code>GLOBUS_TCP_PORT_RANGE</code> environment variable.
<code>-x509-cert-dir</code> <i>TRUSTED_CERTIFICATE_DIRECTORY</i>	Configure the job manager to search <i>TRUSTED_CERTIFICATE_DIRECTORY</i> for its list of trusted CA certificates and their signing policies. This value is also made available in the job environment via the <code>X509_CERT_DIR</code> environment variable.

<code>-cache-location</code> <code>GASS_CACHE_DIRECTORY</code>	Configure the job manager to use the path <code>GASS_CACHE_DIRECTORY</code> for its temporary GASS-cache files. This value is also made available in the job environment via the <code>GLOBUS_GASS_CACHE_DEFAULT</code> environment variable.
<code>-k</code>	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.
<code>-extra-envvars</code> <code>VAR=VAL,...</code>	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 <code>VAR=VAL</code> pairs, where <code>VAR</code> is the variable name and <code>VAL</code> is the variables value.
<code>-seg-module</code> <code>SEG_MODULE</code>	Configure the job manager to use the schedule event generator module named by <code>SEG_MODULE</code> 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.
<code>-audit-directory</code> <code>AUDIT_DIRECTORY</code>	Configure the job manager to write audit records to the directory named by <code>AUDIT_DIRECTORY</code> . This records can be loaded into a database using the globus-gram-audit program.
<code>-globus-toolkit-version</code> <code>TOOLKIT_VERSION</code>	Configure the job manager to use <code>TOOLKIT_VERSION</code> as the version for audit and usage stats records.
<code>-service-tag</code> <code>SERVICE_TAG</code>	Configure the job manager to use <code>SERVICE_TAG</code> 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 <code>untagged</code> will be used.
<code>-disable-streaming</code>	Configure the job manager to disable file streaming. This is propagated to the LRM script interface but has no effect in GRAM5.
<code>-disable-usagestats</code>	Disable sending of any usage stats data, even if <code>-usagestats-target</code> is present in the configuration.
<code>-usagestats-target</code> <code>TARGET</code>	Send usage packets to a data collection service for analysis. The <code>TARGET</code> string consists of a comma-separated list of <code>HOST:PORT</code> combinations, each containing an optional list of data to send. See Usage Stats Packets ¹ for more information about the tags. Special tag strings of <code>all</code> (which enables all tags) and <code>default</code> may be used, or a sequence of characters for the various tags.
<code>-condor-arch</code> <code>ARCH</code>	Set the architecture specification for condor jobs to be <code>ARCH</code> in job classified ads generated by the GRAM5 condor LRM script. This is required for the condor LRM but ignored for all others.
<code>-condor-os</code> <code>OS</code>	Set the operating system specification for condor jobs to be <code>OS</code> in job classified ads generated by the GRAM5 condor LRM script. This is required for the condor LRM but ignored for all others.

¹ <http://confluence.globus.org/display/~bester/GRAM5+Usage+Stats+Packets>

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_HOST- TP_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.

Files

<code>\$HOME/.globus/job/HOST- NAME/LRM.TAG.red</code>	Job manager delegated user credential.
<code>\$HOME/.globus/job/HOST- NAME/LRM.TAG.lock</code>	Job manager state lock file.
<code>\$HOME/.globus/job/HOST- NAME/LRM.TAG.pid</code>	Job manager pid file.
<code>\$HOME/.globus/job/HOST- NAME/LRM.TAG.sock</code>	Job manager socket for inter-job manager communications.
<code>\$HOME/.globus/job/HOST- NAME/JOB_ID/</code>	Job-specific state directory.
<code>\$HOME/.globus/job/HOST- NAME/JOB_ID/stdin</code>	Standard input which has been staged from a remote URL.
<code>\$HOME/.globus/job/HOST- NAME/JOB_ID/stdout</code>	Standard output which will be staged from a remote URL.

<code>\$HOME/.globus/job/HOST-NAME/JOB_ID/stderr</code>	Standard error which will be staged from a remote URL.
<code>\$HOME/.globus/job/HOST-NAME/JOB_ID/x509_user_proxy</code>	Job-specific delegated credential.
<code>\$GLOBUS_LOCATION/tmp/gram_job_state/job.HOST-NAME.JOB_ID</code>	Job state file.
<code>\$GLOBUS_LOCATION/tmp/gram_job_state/job.HOST-NAME.JOB_ID.lock</code>	Job state lock file. In most cases this will be a symlink to the job manager lock file.
<code>\$GLOBUS_LOCATION/etc/globus-job-manager.conf</code>	Default location of the global job manager configuration file.
<code>\$GLOBUS_LOCATION/etc/grid-services/jobmanager-LRM</code>	Default location of the LRM-specific gatekeeper configuration file.

See Also

`globusrun(1)`, `globus-gatekeeper(8)`, `globus-personal-gatekeeper(1)`, `globus-gram-audit(8)`

Chapter 11. Troubleshooting

For a list of error codes generated by GRAM5, see [Section 2, “Errors”](#).

For information about sys admin logging, see [Admin Debugging](#) in the GRAM5 Admin Guide.

1. Troubleshooting tips

In case you run into problems you can do the following

- Check the GRAM5 documentation. Maybe you'll find hints here to solve your problem.
- Check the GRAM5 log for errors.

In case you don't find anything suspicious you can increase the log-level of GRAM5 or other relevant components. Maybe the additional logging-information will tell you what's going wrong.

- Send e-mails to <gram-user@globus.org>. You'll have to subscribe to a list before you can send an e-mail to it. See [here](#)¹ for general e-mail lists and information on how to subscribe to a list and [here](#)² for GRAM specific lists.

¹ http://dev.globus.org/wiki/Mailing_Lists

² http://dev.globus.org/wiki/GRAM#Mailing_Lists

2. Errors

Table 11.1. GRAM5 Errors

Error Code	Reason	Possible Solutions
1	one of the RSL parameters is not supported	Check RSL documentation
2	the RSL length is greater than the maximum allowed	Use RSL substitutions to reduce length of RSL strings
3	an I/O operation failed	Enable trace logging and report to gram-dev@globus.org
4	jobmanager unable to set default to the directory requested	Check that RSL <code>directory</code> attribute refers to a directory that exists on the target system.
5	the executable does not exist	Check that the RSL <code>executable</code> attribute refers to an executable that exists on the target system.
6	of an unused INSUFFICIENT_FUNDS	Unimplemented feature.
7	authentication with the remote server failed	Check that the contact string contains the proper X.509 DN.
8	the user cancelled the job	Don't cancel jobs you want to complete.
9	the system cancelled the job	Check RSL requirements such as maximum time and memory are valid for the job.
10	data transfer to the server failed	Check gatekeeper and/or job manager logs to see why the process failed.
11	the stdin file does not exist	Check that the RSL <code>stdin</code> attribute refers to a file that exists on the target system or has a valid ftp, gsiftp, http, or https URL.
12	the connection to the server failed (check host and port)	Check that the service is running on the expected TCP/IP port. Check that no firewall prevents contacting that TCP/IP port. Check <code>\$GLOBUS_LOCATION/var/globus-gatekeeper.log</code> for runtime configuration errors.
13	the provided RSL 'maxtime' value is not an integer	Check that the RSL <code>maxtime</code> value evaluates to an integer.
14	the provided RSL 'count' value is not an integer	Check that the RSL <code>count</code> value evaluates to an integer.
15	the job manager received an invalid RSL	Check that the RSL string can be parsed by using <code>globusrun -p RSL</code> .
16	the job manager failed in allowing others to make contact	Check job manager log.
17	the job failed when the job manager attempted to run it	Verify that the LRM is configured properly.
18	an invalid paradyn was specified	OBSOLETE IN GRAM2
19	the provided RSL 'jobtype' value is invalid	The RSL <code>jobtype</code> attribute is not indicated as supported by the LRM. Valid <code>jobtype</code> values are <code>single</code> , <code>multiple</code> , <code>mpi</code> , and <code>condor</code> .
20	the provided RSL 'myjob' value is invalid	OBSOLETE IN GRAM5

Error Code	Reason	Possible Solutions
21	the job manager failed to locate an internal script argument file	Check that <code>\$GLOBUS_LOCATION/libexec/globus-job-manager-script.pl</code> exists and is executable. Check that the LRM-specific perl module is located in <code>\$GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/</code> 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.
22	the job manager failed to create an internal script argument file	Check that your home directory is writable and not full.
23	the job manager detected an invalid job state	Check job manager logs.
24	the job manager detected an invalid script response	Check job manager logs. This is likely a bug in the LRM script.
25	the job manager detected an invalid script status	Check job manager logs. This is likely a bug in the LRM script.
26	the provided RSL 'jobtype' value is not supported by this job manager	Check that the RSL <code>jobtype</code> attribute is implemented by the LRM script. Note that some job types require configuration
27	unused ERROR_UNIMPLEMENTED	LRM does not support some feature included in the job request.
28	the job manager failed to create an internal script submission file	Check that the user's home file system is not full. Check job manager log
29	the job manager cannot find the user proxy	Check that client is delegating a proxy when authenticating with the gatekeeper. Check that the user's home filesystem and the <code>/tmp</code> file system are not full.
30	the job manager failed to open the user proxy	Check that the user's home filesystem and the <code>/tmp</code> file system are not full.
31	the job manager failed to cancel the job as requested	Check that the user's home filesystem and the <code>/tmp</code> file system are not full.
32	system memory allocation failed	Check job manager log for details.
33	the interprocess job communication initialization failed	OBSOLETE IN GRAM5
34	the interprocess job communication setup failed	OBSOLETE IN GRAM5
35	the provided RSL 'host count' value is invalid	Check that the RSL <code>host_count</code> attribute evaluates to an integer.
36	one of the provided RSL parameters is unsupported	Check job manager log for details about invalid parameter.
37	the provided RSL 'queue' parameter is invalid	Check that the RSL <code>queue</code> attribute evaluates to a string that corresponds to an LRM-specific queue name.
38	the provided RSL 'project' parameter is invalid	Check that the RSL <code>project</code> attribute evaluates to a string that corresponds to an LRM-specific project name.

Error Code	Reason	Possible Solutions
39	the provided RSL string includes variables that could not be identified	Check that all RSL substitutions are defined before being used in the job description.
40	the provided RSL 'environment' parameter is invalid	Check that the RSL <code>environment</code> attribute contains a sequence of <code>VARIABLE VALUE</code> pairs.
41	the provided RSL 'dryrun' parameter is invalid	Remove the RSL <code>dryrun</code> attribute from the job description.
42	the provided RSL is invalid (an empty string)	Include a non-empty RSL string in your job submission request.
43	the job manager failed to stage the executable	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.
44	the job manager failed to stage the stdin file	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.
45	the requested job manager type is invalid	OBSOLETE IN GRAM5
46	the provided RSL 'arguments' parameter is invalid	OBSOLETE IN GRAM2
47	the gatekeeper failed to run the job manager	Check the gatekeeper or job manager logs for more information.
48	the provided RSL could not be properly parsed	Check that the RSL string can be parsed by using globusrun -p RSL .
49	there is a version mismatch between GRAM components	Ask system administrator to upgrade GRAM service to GRAM2 or GRAM5
50	the provided RSL 'arguments' parameter is invalid	Check that the RSL <code>arguments</code> attribute evaluates to a sequence of strings.
51	the provided RSL 'count' parameter is invalid	Check that the RSL <code>count</code> attribute evaluates to a positive integer value.
52	the provided RSL 'directory' parameter is invalid	Check that the RSL <code>directory</code> attribute evaluates to a string.
53	the provided RSL 'dryrun' parameter is invalid	Check that the RSL <code>dryrun</code> attribute evaluates to either <code>yes</code> or <code>no</code> .
54	the provided RSL 'environment' parameter is invalid	Check that the RSL <code>environment</code> attribute evaluates to a sequence of <code>VARIABLE, VALUE</code> pairs.
55	the provided RSL 'executable' parameter is invalid	Check that the RSL <code>executable</code> attribute evaluates to a string value.
56	the provided RSL 'host_count' parameter is invalid	Check that the RSL <code>host_count</code> attribute evaluates to a positive integer value.
57	the provided RSL 'jobtype' parameter is invalid	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>

Error Code	Reason	Possible Solutions
58	the provided RSL 'maxtime' parameter is invalid	Check that the RSL <code>maxtime</code> attribute evaluates to a positive integer value.
59	the provided RSL 'myjob' parameter is invalid	OBSOLETE IN GRAM5.
60	the provided RSL 'paradyn' parameter is invalid	OBSOLETE IN GRAM2.
61	the provided RSL 'project' parameter is invalid	Check that the RSL <code>project</code> attribute evaluates to a string value.
62	the provided RSL 'queue' parameter is invalid	Check that the RSL <code>queue</code> attribute evaluates to a string value.
63	the provided RSL 'stderr' parameter is invalid	Check that the RSL <code>stderr</code> attribute evaluates to a string value or a sequence of <i>DESTINATION</i> URLs with optional <i>CACHE_TAG</i> string parameters.
64	the provided RSL 'stdin' parameter is invalid	Check that the RSL <code>stdin</code> attribute evaluates to a string value.
65	the provided RSL 'stdout' parameter is invalid	Check that the RSL <code>stdout</code> attribute evaluates to a string value or a sequence of <i>DESTINATION</i> URLs with optional <i>CACHE_TAG</i> string parameters.
66	the job manager failed to locate an internal script	Check job manager log for more details.
67	the job manager failed on the system call <code>pipe()</code>	OBSOLETE IN GRAM5
68	the job manager failed on the system call <code>fcntl()</code>	OBSOLETE IN GRAM2
69	the job manager failed to create the temporary stdout filename	OBSOLETE IN GRAM5
70	the job manager failed to create the temporary stderr filename	OBSOLETE IN GRAM5
71	the job manager failed on the system call <code>fork()</code>	OBSOLETE IN GRAM2
72	the executable file permissions do not allow execution	Check that the RSL <code>executable</code> attribute refers to an executable program or script.
73	the job manager failed to open stdout	Check that the RSL <code>stdout</code> attribute refers to one or more valid destination files or URLs.
74	the job manager failed to open stderr	Check that the RSL <code>stderr</code> attribute refers to one or more valid destination files or URLs.
75	the cache file could not be opened in order to relocate the user proxy	Check that the user's home directory is writable and not full on the GRAM5 service node.
76	cannot access cache files in <code>~/globus/.gass_cache</code> , check permissions, quota, and disk space	Check that the user's home directory is writable and not full on the GRAM5 service node.
77	the job manager failed to insert the contact in the client contact list	Check job manager log

Error Code	Reason	Possible Solutions
78	the contact was not found in the job manager's client contact list	Don't attempt to unregister callback contacts that are not registered
79	connecting to the job manager failed. Possible reasons: job terminated, invalid job contact, network problems, ...	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.
80	the syntax of the job contact is invalid	Check the syntax of job contact string.
81	the executable parameter in the RSL is undefined	Include the RSL <code>executable</code> in all job requests.
82	the job manager service is misconfigured. <code>condor arch</code> undefined	Add the <code>-condor-arch</code> to the command-line or configuration file for a job manager configured to use the <code>condor</code> LRM.
83	the job manager service is misconfigured. <code>condor os</code> undefined	Add the <code>-condor-os</code> to the command-line or configuration file for a job manager configured to use the <code>condor</code> LRM.
84	the provided RSL 'min_memory' parameter is invalid	Check that the RSL <code>min_memory</code> attribute evaluates to a positive integer value.
85	the provided RSL 'max_memory' parameter is invalid	Check that the RSL <code>max_memory</code> attribute evaluates to a positive integer value.
86	the RSL 'min_memory' value is not zero or greater	Check that the RSL <code>min_memory</code> attribute evaluates to a positive integer value.
87	the RSL 'max_memory' value is not zero or greater	Check that the RSL <code>max_memory</code> attribute evaluates to a positive integer value.
88	the creation of a HTTP message failed	Check job manager log.
89	parsing incoming HTTP message failed	Check job manager log.
90	the packing of information into a HTTP message failed	Check job manager log.
91	an incoming HTTP message did not contain the expected information	Check job manager log.
92	the job manager does not support the service that the client requested	Check that the client is talking to the correct service
93	the gatekeeper failed to find the requested service	OBSOLETE IN GRAM2
94	the jobmanager does not accept any new requests (shutting down)	Execute queries before the job has been cleaned up.
95	the client failed to close the listener associated with the callback URL	Call <code>globus_gram_client_callback_disallow()</code> with a valid the callback contact.
96	the gatekeeper contact cannot be parsed	Check the syntax of the gatekeeper contact string you are attempting to contact.
97	the job manager could not find the 'poe' command	OBSOLETE IN GRAM2
98	the job manager could not find the 'mpirun' command	Configure the LRM script with <code>mpirun</code> in your path.

Error Code	Reason	Possible Solutions
99	the provided RSL 'start_time' parameter is invalid	OBSOLETE IN GRAM2
100	the provided RSL 'reservation_handle' parameter is invalid	OBSOLETE IN GRAM2
101	the provided RSL 'max_wall_time' parameter is invalid	Check that the RSL <code>max_wall_time</code> attribute evaluates to a positive integer.
102	the RSL 'max_wall_time' value is not zero or greater	Check that the RSL <code>max_wall_time</code> attribute evaluates to a positive integer.
103	the provided RSL 'max_cpu_time' parameter is invalid	Check that the RSL <code>max_cpu_time</code> attribute evaluates to a positive integer.
104	the RSL 'max_cpu_time' value is not zero or greater	Check that the RSL <code>max_cpu_time</code> attribute evaluates to a positive integer.
105	the job manager is misconfigured, a scheduler script is missing	Check that the administrator has configured the LRM by running its setup script.
106	the job manager is misconfigured, a scheduler script has invalid permissions	Check that the administrator has installed the <code>GLLOBUS_LOCATION/libexec/globus-job-manager-script.pl</code> script. Check that the file system containing that script allows file execution.
107	the job manager failed to signal the job	OBSOLETE IN GRAM2
108	the job manager did not recognize/support the signal type	Check that your signal operation is using the correct signal constant.
109	the job manager failed to get the job id from the local scheduler	OBSOLETE IN GRAM2
110	the job manager is waiting for a commit signal	Send a two-phase commit signal to the job manager to acknowledge receiving the job contact from the job manager.
111	the job manager timed out while waiting for a commit signal	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.
112	the provided RSL 'save_state' parameter is invalid	Check that the RSL <code>save_state</code> attribute is set to <code>yes</code> or <code>no</code> .
113	the provided RSL 'restart' parameter is invalid	Check that the RSL <code>restart</code> attribute evaluates to a string containing a job contact string.
114	the provided RSL 'two_phase' parameter is invalid	Check that the RSL <code>two_phase</code> attribute evaluates to a positive integer.
115	the RSL 'two_phase' value is not zero or greater	Check that the RSL <code>two_phase</code> attribute evaluates to a positive integer.
116	the provided RSL 'stdout_position' parameter is invalid	OBSOLETE IN GRAM5
117	the RSL 'stdout_position' value is not zero or greater	OBSOLETE IN GRAM5

Error Code	Reason	Possible Solutions
118	the provided RSL 'stderr_position' parameter is invalid	OBSOLETE IN GRAM5
119	the RSL 'stderr_position' value is not zero or greater	OBSOLETE IN GRAM5
120	the job manager restart attempt failed	OBSOLETE IN GRAM2
121	the job state file doesn't exist	Check that the job contact you are trying to restart matches one that the job manager returned to you.
122	could not read the job state file	Check that the state file directory is not full.
123	could not write the job state file	Check that the state file directory is not full.
124	old job manager is still alive	Contact the returned job manager contact to manage the job you are trying to restart.
125	job manager state file TTL expired	OBSOLETE in GRAM2
126	it is unknown if the job was submitted	Check job manager log.
127	the provided RSL 'remote_io_url' parameter is invalid	Check that the RSL <code>remote_io_url</code> attribute evaluates to a string value.
128	could not write the remote io url file	Check that the user's home file system on the job manager service node is writable and not full.
129	the standard output/error size is different	Send a stdio update signal to redirect the job manager output to a new URL
130	the job manager was sent a stop signal (job is still running)	Submit a restart request to monitor the job.
131	the user proxy expired (job is still running)	Generate a new proxy and then submit a restart request to monitor the job.
132	the job was not submitted by original job-manager	OBSOLETE IN GRAM2
133	the job manager is not waiting for that commit signal	Do not send a commit signal to a job that is not waiting for a commit signal.
134	the provided RSL scheduler specific parameter is invalid	Check the LRM-specific documentation to determine what values are legal for the RSL extensions implemented by the LRM.
135	the job manager could not stage in a file	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.
136	the scratch directory could not be created	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.
137	the provided 'gass_cache' parameter is invalid	Check that the RSL <code>gass_cache</code> attribute evaluates to a string.
138	the RSL contains attributes which are not valid for job submission	Do not use restart- or signal-only RSL attributes when submitting a job.

Error Code	Reason	Possible Solutions
139	the RSL contains attributes which are not valid for stdio update	Do not use submit- or restart-only RSL attributes when sending a stdio update signal to a job.
140	the RSL contains attributes which are not valid for job restart	Do not use submit- or signal-only RSL attributes when restarting a job.
141	the provided RSL 'file_stage_in' parameter is invalid	Check that the RSL <code>file_stage_in</code> attribute evaluates to a sequence of <i>SOURCE DESTINATION</i> pairs.
142	the provided RSL 'file_stage_in_shared' parameter is invalid	Check that the RSL <code>file_stage_in_shared</code> attribute evaluates to a sequence of <i>SOURCE DESTINATION</i> pairs.
143	the provided RSL 'file_stage_out' parameter is invalid	Check that the RSL <code>file_stage_out</code> attribute evaluates to a sequence of <i>SOURCE DESTINATION</i> pairs.
144	the provided RSL 'gass_cache' parameter is invalid	Check that the RSL <code>gass_cache</code> attribute evaluates to a string.
145	the provided RSL 'file_cleanup' parameter is invalid	Check that the RSL <code>file_clean_up</code> attribute evaluates to a sequence of strings.
146	the provided RSL 'scratch_dir' parameter is invalid	Check that the RSL <code>scratch_dir</code> attribute evaluates to a string.
147	the provided scheduler-specific RSL parameter is invalid	Check the LRM-specific documentation to determine what values are legal for the RSL extensions implemented by the LRM.
148	a required RSL attribute was not defined in the RSL spec	Check that the RSL <code>executable</code> attribute is present in your job request RSL. Check that the RSL <code>restart</code> attributes is present in your restart RSL.
149	the <code>gass_cache</code> attribute points to an invalid cache directory	Check that the RSL <code>gass_cache</code> attributes evaluates to a directory that exists or can be created. Check that the user's home file system is writable and not full.
150	the provided RSL 'save_state' parameter has an invalid value	Check that the RSL <code>save_state</code> attribute has a value of <code>yes</code> or <code>no</code> .
151	the job manager could not open the RSL attribute validation file	Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf</code> is present and readable on the job manager service node. Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf</code> is readable on the job manager service node if present.
152	the job manager could not read the RSL attribute validation file	Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf</code> is valid. Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf</code> is valid if present.
153	the provided RSL 'proxy_timeout' is invalid	Check that RSL <code>proxy_timeout</code> attribute evaluates to a positive integer.
154	the RSL 'proxy_timeout' value is not greater than zero	Check that RSL <code>proxy_timeout</code> attribute evaluates to a positive integer.

Error Code	Reason	Possible Solutions
155	the job manager could not stage out a file	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.
156	the job contact string does not match any which the job manager is handling	Check that the job contact string matches one returned from a job request.
157	proxy delegation failed	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.
158	the job manager could not lock the state lock file	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.
159	an invalid globus_io_clientattr_t was used.	Check that you have initialized the globus_io_clientattr_t attribute prior to using it with the GRAM client API.
160	an null parameter was passed to the gram library	Check that you are passing legal values to all GRAM API calls.
161	the job manager is still streaming output	OBSOLETE IN GRAM5
162	the authorization system denied the request	Check with your GRAM system administrator to allow a particular certificate to be authorized.
163	the authorization system reported a failure	Check with your system administrator to verify that the authorization system is configured properly.
164	the authorization system denied the request - invalid job id	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.
165	the authorization system denied the request - not authorized to run the specified executable	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.
166	the provided RSL 'user_name' parameter is invalid.	Check that the RSL user_name attribute evaluates to a string.
167	the job is not running in the account named by the 'user_name' parameter.	Ask with the GRAM system administrator to add an authorization entry to allow your credential to run jobs as the specified user account.

Chapter 12. 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 ../../Usage_Stats.html

Glossary

C

Condor A job scheduler mechanism supported by GRAM. See <http://www.cs.wisc.edu/condor/> for more information.

L

LSF A job scheduler mechanism supported by GRAM.

For more information, see <http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/>⁴.

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.

⁴ <http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/>

Index

U

usage statistics, 37

A

audit logging, 10

C

configuration interface, 2

 default local resource manager, 5

 disabling local resource manager adapter, 5

 lrm-specific, 3

 non-default, 3

 authorization, 3

 typical, 2

 LRM adapters, 2

configuring, 2

 default local resource manager, 5

 disabling local resource manager adapter, 5

 lrm-specific, 3

 non-default, 3

 authorization, 3

 typical, 2

 LRM adapters, 2

D

deploying, 6

E

errors, 27

I

installing

 prerequisites

 local scheduler, 1

 lrm adapter, 1

P

performance guide, 9

 server-side, 9

S

SEG, 8

T

troubleshooting, 26

 check documentation, 26

 errors, 26

 gram log, 26

 mailing lists, 26

GT 5.0.0 GRAM5: User's Guide

GT 5.0.0 GRAM5: User's Guide

Introduction

GRAM services provide secure, remote job submission to different *local resource managers* in a Grid environment. This document describes how to use the RSL language and command-line interfaces provided in GT 5.0.0 to submit jobs to Grid resources.

Table of Contents

1. Using GRAM5	1
1. Preparing to use GRAM	1
2. Java Client API Download	1
3. Delegating credentials	2
4. Submitting jobs	2
5. Using GRAM5 with Condor-G	13
I. GRAM5 Commands	14
globusrun	15
globus-job-cancel	19
globus-job-clean	20
globus-job-get-output	21
globus-job-run	23
globus-job-status	26
globus-job-submit	28
globus-personal-gatekeeper	31
globus-gram-audit	33
globus-gatekeeper	34
globus-job-manager	37
globus-job-manager-event-generator	42
globus-fork-starter	43
2. Troubleshooting	45
1. Troubleshooting tips	45
2. Errors	46
3. Known Problems in GRAM5	56
1. Known Problems	56
4. Usage statistics collection by the Globus Alliance	59
1. GRAM5-specific usage statistics	59
Glossary	62
Index	63

List of Tables

2.1. GRAM5 Errors	47
-------------------------	----

List of Examples

1.1. Generating a proxy with grid-proxy-init	1
1.2. Gatekeeper Service Contact Examples	3
1.3. Minimal job using globus-job-run	4
1.4. Multiprocess job using globus-job-run	4
1.5. Canceling an interactive job	4
1.6. Setting job environment variables with globus-job-run	4
1.7. Using custom RSL clauses with globus-job-run	5
1.8. Constructing RSL strings with globus-job-run	5
1.9. globus-job-submit	6
1.10. Checking RSL Syntax	7
1.11. GRAM Authentication test	7
1.12. GRAM version check	8
1.13. Basic Interactive Job	8
1.14. Basic Batch Job	8
1.15. Refreshing a Credential	9
1.16. Proxy Expiration Example	9
1.17. File stage in	10
1.18. File stage in shared	10
1.19. File stage out	10
1.20. Staging to scratch directory	11
1.21. Cleaning up a file	11
1.22. Two phase commit example	12
1.23. Restart example	12
1.24. Java example	13

Chapter 1. Using GRAM5

1. Preparing to use GRAM

The first step to being able to use GRAM5 after installation is to acquire a temporary Grid credential to use to authenticate with the GRAM5 service and any file services your job requires. Normally this is done via either **grid-proxy-init** or via the MyProxy service.

1.1. Proxy credentials with grid-proxy-init

To generate a proxy credential using the **grid-proxy-init** program, execute the command with no arguments. By default, it will generate an impersonation proxy with a lifetime of 12 hours.

Example 1.1. Generating a proxy with grid-proxy-init

This example creates a 12 hour impersonation proxy to use to authenticate with grid services such as GRAM5:

```
% bin/grid-proxy-init
Your identity: /O=Grid/OU=Example/CN=Joe User
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Your proxy is valid until: Tue Oct 26 01:33:42 2010
```

Important

In order to generate a proxy credential, you must have first been issued an identity credential by some certificate authority that is trusted by the GRAM5 resource you want to use. To learn more about certificates and Grid security in general, please read Security Key Concepts.

2. Java Client API Download

GT 5.0.0 does not include any of the CoG JGlobus Java APIs that were included in the GT4 release series. But, the JGlobus APIs can still be used with the GT5 services. You can get them directly from the CoG JGlobus releases; see the following link:

http://dev.globus.org/wiki/CoG_jglobus

Consider the following when determining which version of CoG JGlobus to use:

- The GRAM development team used CoG JGlobus version 1.6.0 for performance testing.
- The BIRN project used CoG JGlobus version 1.6.0 (plus patches) for GridFTP testing. All patches are included in 1.8.0.
- At the time of the GT 5.0.0 release, 1.8.0 was the recommended version. In general, the latest recommended CoG JGlobus version should be used.

3. Delegating credentials

The credential created in the previous section is used to authenticate with the GRAM5 service as well as to delegate a limited proxy of that credential to the service so that it can process the job. This credential delegation occurs when the **globus-gatekeeper** service is first contacted when a job is to be submitted. By default, the tools provided with GT 5.0.0 delegate a *limited proxy*. This limited proxy can be used to authenticate with other services on the client's behalf, but with the services knowing that the proxy is not under direct control by the user.

3.1. Delegated Credential Usage

The delegated proxy can be used by the GRAM5 service and the job in a few different ways:

1. The GRAM5 service uses the credential to send job state notification messages to clients which have registered to receive them.
2. The GRAM5 service uses the credential to contact GASS and GridFTP file servers to stage files to and from the execution resource
3. The job executed by the GRAM5 service can use the delegated credential for application-specific purposes.



Note

In GRAM5, the Job Manager may manage multiple jobs simultaneously. It will use the delegated proxy with the most time left for authentication. Individual GRAM5 jobs will have separate proxies.

globusrun **globus-job-run**, and **globus-job-submit** commands delegate credentials automatically when submitting a job. Additionally, **globusrun** can refresh the credentials used by the job and job manager, after the job manager is started.

4. Submitting jobs

This section describes the steps needed to submit jobs to resources managed by GRAM5 services. It describes how resources are named, tools for submitting and monitoring jobs, and the RSL language which describes requirements for jobs.

4.1. 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 1.2. 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`

4.2. Running 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 GRAM5 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`** as if the job were run interactively. Note that input to the job must be located in a file prior to running the job; true interactive I/O is not supported by GRAM5.

The **`globus-job-run`** program has command-line options to control most aspects of jobs run by GRAM5. 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 1.3. 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:2119/jobmanager-pbs`.

```
% globus-job-run grid.example.org:2119/jobmanager-pbs /bin/hostname
node1.grid.example.org
```

Example 1.4. Multiprocess job using globus-job-run

The following command line submits ten instances of an executable `a.out`, staging it from the client host to the service node using GASS. The `a.out` program prints the name of the host it is executing on.

```
% globus-job-run grid.example.org:2119/jobmanager-pbs -np 10 -s a.out
node1.grid.example.org
node3.grid.example.org
node2.grid.example.org
node5.grid.example.org
node4.grid.example.org
node8.grid.example.org
node6.grid.example.org
node9.grid.example.org
node7.grid.example.org
node10.grid.example.org
```

Example 1.5. 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:2119/jobmanager-pbs /bin/sleep 90
Control-C
GRAM Job failed because the user cancelled the job (error code 8)
```

Example 1.6. 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.

```
% globus-job-run grid.example.org:2119/jobmanager-pbs -env TEST=1 -env GRID=1 /usr/bin/env
HOME=/home/juser
LOGNAME=juser
GLOBUS_GRAM_JOB_CONTACT=https://client.example.org:3882/16001579536700793196/5295612977485
GLOBUS_LOCATION=/opt/globus-5.0.0
GLOBUS_GASS_CACHE_DEFAULT=/home/juser/.globus/.gass_cache
TEST=1
X509_USER_PROXY=/home/juser/.globus/job/mactop.local/16001579536700793196.5295612977485997
GRID=1
```

Example 1.7. 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:2119/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 1.8. 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:2119/jobmanager-pbs -np 5 -x '&(jobtype=mpi)' -
&(jobtype=mpi)
  (executable="a.out")
  (environment= ("GRID" "1") ("TEST" "1"))
  (count=5)
```

4.3. Submitting 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.

Example 1.9. 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**.

```
% globus-job-submit grid.example.org:2119/jobmanager-pbs /bin/hostname
https://grid.example.org:38843/16001600430615223386/5295612977486013582/
% globus-job-status https://grid.example.org:38843/16001600430615223386/529561297748601358
PENDING
% globus-job-status https://grid.example.org:38843/16001600430615223386/529561297748601358
ACTIVE
% globus-job-status https://grid.example.org:38843/16001600430615223386/529561297748601358
DONE
% globus-job-get-output -r grid.example.org:2119/jobmanager-fork \
  https://grid.example.org:38843/16001600430615223386/5295612977486013582/
node1.grid.example.org
% globus-job-clean -r grid.example.org:2119/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.

4.4. Using 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.

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 `-dump_rsl` 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.

4.4.1. 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 1.10. 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

4.4.2. 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 1.11. GRAM Authentication test

```
% globusrun -a -r grid.example.org:2119/jobmanager-pbs  
GRAM Authentication test successful  
% globusrun -a -r grid.example.org:2119/jobmanager-lsf  
GRAM Authentication test failure: the gatekeeper failed to find the requested service  
% globusrun -a -r grid.example.org:2119/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.

4.4.3. Checking GRAM service version

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

Example 1.12. GRAM version check

```
% globusrun -j -r grid.example.org:2119/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
```

4.4.4. 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 command 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.

Example 1.13. 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
```

4.4.5. 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 1.14. Basic Batch Job

```
% globusrun -b -r grid.example.org:2119/jobmanager-pbs "&(executable=/bin/sleep)(arguments
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:38824/16001608125017717261/529561297748601998
PENDING
% globusrun -k https://grid.example.org:38824/16001608125017717261/5295612977486019989/
%
```

4.4.6. Refreshing a GRAM5 Credential

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

Example 1.15. Refreshing a Credential

```
% globusrun -refresh-proxy https://grid.example.org:38824/16001608125017717261/52956129774
% echo $?
0
```



Note

In GT 5.0.0, **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.

4.4.7. 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 1.16. Proxy Expiration Example

```
% globusrun -r grid.example.org "&(executable=a.out)"
globus_gram_client_callback_allow successful
GRAM Job submission successful
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE
GLOBUS_GRAM_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 .....
Your proxy is valid until: Tue Nov 10 04:25:03 2009
% globusrun -r grid.example.org "&(restart="https://grid.example.org:1997/1600170047757511
globus_gram_client_callback_allow successful
GRAM Job submission successful
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE
GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE
```

4.4.8. 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 1.17. File stage in

```
% globusrun -s -r grid.example.org:2119/jobmanager-pbs \  
  "&(rsl_substitution = (GRIDFTP_SERVER gsiftp://gridftp.example.org)) \  
  (executable=/bin/ls) \  
  (arguments=/tmp/staged_file) \  
  (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 1.18. File stage in shared

```
% globusrun -s -r grid.example.org:2119/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/.ga
```

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 1.19. File stage out

```
% globusrun -r grid.example.org:2119/jobmanager-pbs \  
  "&(executable=a.out) \  
  (file_stage_out = (results.txt ftp://anonymous:nopass@ftp.example.org/incoming/resul" \  
%
```

Note

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

4.4.9. 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 1.20. Staging to scratch directory

```
% globusrun -s grid.example.org:2119/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 1.21. Cleaning up a file

```
% globusrun -s grid.example.org:2119/jobmanager-pbs \  
  "&(executable = /bin/touch) \  
  (arguments = temporary_file) \  
  (file_clean_up = temporary_file)" \  
%
```

4.4.10. 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 1.22. 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:2119/jobmanager-pbs \  
  "&(two_phase = 30) \  
    (save_state = yes) \  
    (executable = a.out)"
```

```
globus_gram_client_callback_allow successful  
GRAM Job submission successful  
GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING  
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE  
GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE  
%
```

4.4.11. 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 1.23. 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:2119/jobmanager-pbs /bin/sleep 90  
https://grid.example.org:38824/16001746665595486521/5295612977486005662/  
% globusrun -r grid.example.org:2119/jobmanager-pbs \  
  "&(restart = https://grid.example.org:38824/16001746665595486521/5295612977486005662/)"  
globus_gram_client_callback_allow successful  
GRAM Job submission successful  
GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE  
%
```

4.4.12. 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 1.24. Java example

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

```
% globusrun -r grid.example.org:2119/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')" \  
globus_gram_client_callback_allow successful \  
GRAM Job submission successful \  
GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING \  
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE \  
GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE \  
%
```

5. Using GRAM5 with Condor-G

Condor-G users should upgrade their clients to condor 7.4.0 or later to achieve highest performance. That version includes the `gt5` grid type, which includes client-side optimizations to improve performance. To use an older Condor-G client, be sure to set the `GRIDMANAGER_MAX_JOBMANAGERS_PER_RESOURCE` attribute to 0 to disable the Condor-G client attempts to stop and restart the job manager service. Also, disable the Grid Monitor when using a GRAM5 resource by setting the `ENABLE_GRID_MONITOR` configuration attribute to `FALSE`.

GRAM5 Commands

Name

globusrun -- Execute and manage jobs via GRAM

```
globusrun [-help] [-usage] [-version] [-versions]
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 [-n] [-no-interrupt] [-full-proxy] [-D] { -y | -refresh-proxy } { JOB_ID }
globusrun { -status } { JOB_ID }
globusrun [-q] [-quiet] [-o] [-output-enable] [-s] [-server] [-w] [-write-allow] [-n] [-no-interrupt] [-b] [-batch] [-F]
[-fast-batch] [-full-proxy] [-D] [-d] [-dryrun]
{ -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 will 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 `-k` or `-kill` command-line option is present, **globusrun** will attempt to terminate the job named by *JOB_ID*. If successful, **globusrun** will exit with zero; otherwise it will display an explanation of the problem and exit with a non-zero exit code.

When the `-y` or `-refresh-proxy` command-line option is present, **globusrun** will attempt to delegate a new X.509 proxy to the job manager which is managing the job named by *JOB_ID*. If successful, **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 `-full-proxy` or `-D` command-line options to enable full proxy delegation. The default is limited proxy delegation.

When the `-status` command-line option is present, **globusrun** will attempt to determine the current state of the job. If successful, the state will be printed to standard output and **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, **globusrun** will submit the job to a GRAM service. By default, **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 **globusrun** program can also function as a GASS file server to allow the **globus-job-manager** program to stage files to and from the machine on which **globusrun** is executed to the GRAM service node. This behavior is controlled by the `-s`, `-o`, and `-w` command-line options.

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

Options

The full set of options to **globusrun** consist of:

<code>-help</code>	Display a help message to standard error and exit.
<code>-usage</code>	Display a one-line usage summary to standard error and exit.
<code>-version</code>	Display the software version of globusrun to standard error and exit.
<code>-versions</code>	Display the software version of all modules used by globusrun (including DiRT information) to standard error and then exit.
<code>-p, -parse</code>	Do a parse check on the job specification and print diagnostics. If a parse error occurs, globusrun exits with a non-zero exit code.
<code>-f RSL_FILENAME, -file RSL_FILENAME</code>	Read job specification from the file named by <i>RSL_FILENAME</i> .
<code>-n, -no-interrupt</code>	Disable handling of the SIGINT signal, so that the interrupt character (typically Control-C) causes globusrun to terminate without canceling the job.
<code>-r RESOURCE_CONTACT, -resource RESOURCE_CONTACT</code>	Submit the request to the resource specified by <i>RESOURCE_CONTACT</i> . A resource may be specified in the following ways: <ul style="list-style-type: none"> • <i>HOST</i> • <i>HOST:PORT</i> • <i>HOST:PORT/SERVICE</i> • <i>HOST/SERVICE</i> • <i>HOST:/SERVICE</i> • <i>HOST::SUBJECT</i> • <i>HOST:PORT:SUBJECT</i> • <i>HOST/SERVICE:SUBJECT</i>

- *HOST:/SERVICE:SUBJECT*
- *HOST:PORT/SERVICE:SUBJECT*

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

-j, -jobmanager-version	Print the software version being run by the service running at <i>RESOURCE_CONTACT</i> .
-k <i>JOB_ID</i> , -kill <i>JOB_ID</i>	Kill the job named by <i>JOB_ID</i>
-D, -full-proxy	Delegate a full impersonation proxy to the service. By default, a limited proxy is delegated when needed.
-y, -refresh-proxy	Delegate a new proxy to the service processing <i>JOB_ID</i> .
-status	Display the current status of the job named by <i>JOB_ID</i> .
-q, -quiet	Do not display job state change or exit code information.
-o, -output-enable	Start a GASS server within the globusrun application that allows access to its standard output and standard error streams only. Also, augment the <i>RSL_SPECIFICATION</i> with a definition of the GLOBUSRUN_GASS_URL RSL substitution and add <code>stdout</code> and <code>stderr</code> clauses which redirect the output and error streams of the job to the output and error streams of the interactive globusrun command. If this is specified, then globusrun acts as though the <code>-q</code> were also specified.
-s, -server	Start a GASS server within the globusrun application that allows access to its standard output and standard error streams for writing and any file local the the globusrun invocation for reading. Also, augment the <i>RSL_SPECIFICATION</i> with a definition of the GLOBUSRUN_GASS_URL RSL substitution and add <code>stdout</code> and <code>stderr</code> clauses which redirect the output and error streams of the job to the output and error streams of the interactive globusrun command. If this is specified, then globusrun acts as though the <code>-q</code> were also specified.
-w, -write-allow	Start a GASS server within the globusrun application that allows access to its standard output and standard error streams for writing and any file local the the globusrun invocation for reading or writing. Also, augment the <i>RSL_SPECIFICATION</i> with a definition of the GLOBUSRUN_GASS_URL RSL substitution and add <code>stdout</code> and <code>stderr</code> clauses which redirect the output and error streams of the job to the output and error streams of the interactive globusrun command. If this is specified, then globusrun acts as though the <code>-q</code> 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 <i>JOB_ID</i> 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

```
globus-job-cancel [ -f | -force ] [ -q | -quiet ] JOBID  
globus-job-cancel [-help] [-usage] [-version] [-versions]
```

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

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

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

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

```
globus-job-clean [ -r RESOURCE | -resource RESOURCE ]  
[ -f | -force ] [ -q | -quiet ] JOBID  
globus-job-clean [-help] [-usage] [-version] [-versions]
```

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:

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

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

```
globus-job-get-output [ -r RESOURCE | -resource RESOURCE ]  
[ -out | -err ] [ -t LINES | -tail LINES ] [ -follow LINES | -f LINES ] JOBID  
globus-job-get-output [-help] [-usage] [-version] [-versions]
```

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:

<code>-help, -usage</code>	Display a help message to standard error and exit.
<code>-version</code>	Display the software version of the globus-job-get-output program to standard output.
<code>-version</code>	Display the software version of the globus-job-get-output program including DiRT information to standard output.
<code>-resource <i>RESOURCE</i>,</code> <code>-r <i>RESOURCE</i></code>	Submit the retrieval job to the resource named by <i>RESOURCE</i> instead of the default GRAM service on the same host as the job contact.
<code>-out</code>	Retrieve only the standard output stream of the job. The default is to retrieve both standard output and standard error.
<code>-err</code>	Retrieve only the standard error stream of the job. The default is to retrieve both standard output and standard error.
<code>-tail <i>LINES</i>, -t</code> <code><i>LINES</i></code>	Print only the last <i>LINES</i> 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 <code>-f</code> or <code>-follow</code> options.
<code>-follow <i>LINES</i>, -f</code> <code><i>LINES</i></code>	Print the last <i>LINES</i> 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 <code>-t</code> or <code>-tail</code> 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.

BUGS

The `-f` and `-follow` don't work in GRAM5.

Name

globus-job-run -- Execute a job using GRAM

```
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]...
[-stdin [ -l | -s ] STDIN_FILE ] [-stdout [ -l | -s ] STDOUT_FILE ] [-stderr [ -l | -s ] STDERR_FILE ]
[-x RSL_CLAUSE]
[ -l | -s ] EXECUTABLE [ARGUMENT...]
globus-job-run [-help] [-usage] [-version] [-versions]
```

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:

<code>-help, -usage</code>	Display a help message to standard error and exit.
<code>-version</code>	Display the software version of the globus-job-run program to standard output.
<code>-version</code>	Display the software version of the globus-job-run program including DiRT information to standard output.
<code>-dumprsl</code>	Translate the command-line options to globus-job-run into an RSL expression that can be used with tools such as globusrun .
<code>-dryrun</code>	Submit the job request to the GRAM service with the <code>dryrun</code> 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.
<code>-verify</code>	Submit the job request to the GRAM service with the <code>dryrun</code> option enabled and then without it enabled if the <code>dryrun</code> is successful.
<code>-file ARGUMENT_FILE</code>	Read additional command-line options from <i>ARGUMENT_FILE</i> .
<code>-np PROCESSES, -count PRO-</code> <code>CESSSES</code>	Start <i>PROCESSES</i> instances of the executable as a single job.

<p><code>-m MAX_TIME, -maxtime MAX_TIME</code></p>	<p>Schedule the job to run for a maximum of <i>MAX_TIME</i> minutes.</p>
<p><code>-p PROJECT, -project PRO- JECT</code></p>	<p>Request that the job use the allocation <i>PROJECT</i> when submitting the job to the LRM.</p>
<p><code>-q QUEUE, -queue QUEUE</code></p>	<p>Request that the job be submitted to the LRM using the named <i>QUEUE</i>.</p>
<p><code>-d DIRECTORY, -directory DIRECTORY</code></p>	<p>Run the job in the directory named by <i>DIRECTORY</i>. 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.</p>
<p><code>-env NAME=VALUE</code></p>	<p>Define an environment variable named by <i>NAME</i> with the value <i>VALUE</i> in the job environment. This option may be specified multiple times to define multiple environment variables.</p>
<p><code>-stdin [-l -s] STDIN_FILE</code></p>	<p>Use the file named by <i>STDIN_FILE</i> as the standard input of the job. If the <code>-l</code> option is specified, then this file is interpreted to be on a file system local to the LRM. If the <code>-s</code> 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.</p>
<p><code>-stdout [-l -s] STDOUT_FILE</code></p>	<p>Use the file named by <i>STDOUT_FILE</i> as the destination for the standard output of the job. If the <code>-l</code> option is specified, then this file is interpreted to be on a file system local to the LRM. If the <code>-s</code> 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.</p>
<p><code>-stderr [-l -s] STDERR_FILE</code></p>	<p>Use the file named by <i>STDERR_FILE</i> as the destination for the standard error of the job. If the <code>-l</code> option is specified, then this file is interpreted to be on a file system local to the LRM. If the <code>-s</code> 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.</p>
<p><code>-x RSL_CLAUSE</code></p>	<p>Add a set of custom RSL attributes described by <i>RSL_CLAUSE</i> 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.</p>
<p><code>-l</code></p>	<p>When included outside the context of <code>-stdin</code>, <code>-stdout</code>, or <code>-stderr</code> command-line options, <code>-l</code> option alters the interpretation of the executable path. If the <code>-l</code> option is specified, then the executable is interpreted to be on a file system local to the LRM.</p>
<p><code>-s</code></p>	<p>When included outside the context of <code>-stdin</code>, <code>-stdout</code>, or <code>-stderr</code> command-line options, <code>-s</code> option alters the interpretation of the executable path. If the <code>-s</code> 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.</p>

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

`globus-job-status` *JOBID*

`globus-job-status` [-help] [-usage] [-version] [-versions]

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, -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 standard 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 can not 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

```
globus-job-submit [-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]...
[-stdin [ -l | -s ] STDIN_FILE ] [-stdout [ -l | -s ] STDOUT_FILE ] [-stderr [ -l | -s ] STDERR_FILE ]
[-x RSL_CLAUSE]
[ -l | -s ] EXECUTABLE [ARGUMENT...]
globus-job-submit [-help] [-usage] [-version] [-versions]
```

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 `-dumprsl`, `-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:

<code>-help, -usage</code>	Display a help message to standard error and exit.
<code>-version</code>	Display the software version of the globus-job-submit program to standard output.
<code>-versions</code>	Display the software version of the globus-job-submit program including DiRT information to standard output.
<code>-dumprsl</code>	Translate the command-line options to globus-job-submit into an RSL expression that can be used with tools such as globusrun .
<code>-dryrun</code>	Submit the job request to the GRAM service with the <code>dryrun</code> 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.
<code>-verify</code>	Submit the job request to the GRAM service with the <code>dryrun</code> option enabled and then without it enabled if the <code>dryrun</code> is successful.

<code>-file ARGUMENT_FILE</code>	Read additional command-line options from <i>ARGUMENT_FILE</i> .
<code>-np PROCESSES, -count PRO- CESSES</code>	Start <i>PROCESSES</i> instances of the executable as a single job.
<code>-m MAX_TIME, -maxtime MAX_TIME</code>	Schedule the job to run for a maximum of <i>MAX_TIME</i> minutes.
<code>-p PROJECT, -project PRO- JECT</code>	Request that the job use the allocation <i>PROJECT</i> when submitting the job to the LRM.
<code>-q QUEUE, -queue QUEUE</code>	Request that the job be submitted to the LRM using the named <i>QUEUE</i> .
<code>-d DIRECTORY, -directory DIRECTORY</code>	Run the job in the directory named by <i>DIRECTORY</i> . 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.
<code>-env NAME=VALUE</code>	Define an environment variable named by <i>NAME</i> with the value <i>VALUE</i> in the job environment. This option may be specified multiple times to define multiple environment variables.
<code>-stdin [-l -s] STDIN_FILE</code>	Use the file named by <i>STDIN_FILE</i> as the standard input of the job. If the <code>-l</code> option is specified, then this file is interpreted to be on a file system local to the LRM. If the <code>-s</code> 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.
<code>-stdout [-l -s] STDOUT_FILE</code>	Use the file named by <i>STDOUT_FILE</i> as the destination for the standard output of the job. If the <code>-l</code> option is specified, then this file is interpreted to be on a file system local to the LRM. If the <code>-s</code> 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.
<code>-stderr [-l -s] STDERR_FILE</code>	Use the file named by <i>STDERR_FILE</i> as the destination for the standard error of the job. If the <code>-l</code> option is specified, then this file is interpreted to be on a file system local to the LRM. If the <code>-s</code> 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.
<code>-x RSL_CLAUSE</code>	Add a set of custom RSL attributes described by <i>RSL_CLAUSE</i> 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 .
<code>-l</code>	When included outside the context of <code>-stdin</code> , <code>-stdout</code> , or <code>-stderr</code> command-line options, <code>-l</code> option alters the interpretation of the executable path. If the <code>-l</code> option is specified, then the executable is interpreted to be on a file system local to the LRM.
<code>-s</code>	When included outside the context of <code>-stdin</code> , <code>-stdout</code> , or <code>-stderr</code> command-line options, <code>-s</code> option alters the interpretation of the executable path. If the <code>-s</code> 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)

Name

`globus-personal-gatekeeper --` Manage a user's personal gatekeeper daemon

```
globus-personal-gatekeeper [-help] [-usage] [-version] [-versions] [-list] [-directory CONTACT]
globus-personal-gatekeeper [-debug] {-start} [-jmtime LRM] [-auditdir AUDIT_DIRECTORY] [-port PORT]
[-log [=DIRECTORY]] [-seg] [-acctfile ACCOUNTING_FILE]
globus-personal-gatekeeper [-killall] [-kill]
```

Description

The **globus-personal-gatekeeper** command is a utility which manages a gatekeeper and job manager service for a single user. Depending on the command-line arguments it will operate in one of several modes. In the first set of arguments indicated in the synopsis, the program provides information about the **globus-personal-gatekeeper** command or about instances of the **globus-personal-gatekeeper** that are running currently. The second set of arguments indicated in the synopsis provide control over starting a new **globus-personal-gatekeeper** instance. The final set of arguments provide control for terminating one or more **globus-personal-gatekeeper** instances.

The `-start` mode will create a new subdirectory of `$HOME/.globus` and write the configuration files needed to start a **globus-gatekeeper** daemon which will invoke the **globus-job-manager** service when new authenticated connections are made to its service port. The **globus-personal-gatekeeper** then exits, printing the contact string for the new gatekeeper prefixed by `GRAM contact:` to standard output. In addition to the arguments described above, any arguments described in **globus-job-manager(8)** can be appended to the command-line and will be added to the job manager configuration for the service started by the **globus-gatekeeper**.

The new **globus-gatekeeper** will continue to run in the background until killed by invoking **globus-personal-gatekeeper** with the `-kill` or `-killall` argument. When killed, it will kill the **globus-gatekeeper** and **globus-job-manager** processes, remove state files and configuration data, and then exit. Jobs which are running when the personal gatekeeper is killed will continue to run, but their job directory will be destroyed so they may fail in the LRM.

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

<code>-help, -usage</code>	Print command-line option summary and exit
<code>-version</code>	Print software version
<code>-versions</code>	Print software version including DiRT information
<code>-list</code>	Print a list of all currently running personal gatekeepers. These entries will be printed one per line.
<code>-directory CONTACT</code>	Print the configuration directory for the personal gatekeeper with the contact string <i>CONTACT</i> .
<code>-debug</code>	Print additional debugging information when starting a personal gatekeeper. This option is ignored in other modes.
<code>-start</code>	Start a new personal gatekeeper process.
<code>-jmtime LRM</code>	Use <i>LRM</i> as the local resource manager interface. If not provided when starting a personal gatekeeper, the job manager will use the default <code>fork</code> LRM.
<code>-auditdir AUDIT_DIRECTORY</code>	Write audit report files to <i>AUDIT_DIRECTORY</i> . If not provided, the job manager will not write any audit files.

- `-port PORT` Listen for gatekeeper TCP/IP connections on the port *PORT*. If not provided, the gatekeeper will let the operating system choose.
- `-log[=DIRECTORY]` Write job manager log files to *DIRECTORY*. If *DIRECTORY* is omitted, the default of `$HOME` will be used. If this option is not present, the job manager will not write any log files.
- `-seg` Try to use the SEG mechanism to receive job state change information, instead of polling for these. These require either the system administrator or the user to run an instance of the **globus-job-manager-event-generator** program for the LRM specified by the `-jmtime` option.
- `-acctfile ACCOUNTING_FILE` Write gatekeeper accounting entries to *ACCOUNTING_FILE*. If not provided, no accounting records are written.

Examples

This example shows the output when starting a new personal gatekeeper which will schedule jobs via the lsf LRM, with debugging enabled.

```
% globus-personal-gatekeeper -start -jmtime lsf
```

```
verifying setup...
```

```
done.
```

```
GRAM contact: personal-grid.example.org:57846:/DC=org/DC=example/CN=Joe User
```

This example shows the output when listing the current active personal gatekeepers.

```
% globus-personal-gatekeeper -list
```

```
personal-grid.example.org:57846:/DC=org/DC=example/CN=Joe User
```

This example shows the output when querying the configuration directory for the above personal gatekeeper. gatekeepers.

```
% globus-personal-gatekeeper -directory "personal-grid.example.org:57846:/DC=org/DC=example/CN=Joe User"
```

```
/home/juser/.globus/.personal-gatekeeper.personal-grid.example.org.1337
```

```
% globus-personal-gatekeeper -kill "personal-grid.example.org:57846:/DC=org/DC=example/CN=Joe User"
```

```
killing gatekeeper: "personal-grid.example.org:57846:/DC=org/DC=example/CN=Joe User"
```

See Also

globusrun(1), globus-job-manager(8), globus-gatekeeper(8)

Name

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

globus-gram-audit [--conf *CONFIG_FILE*] [--check] [--delete] [--audit-directory *AUDITDIR*]

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:

<code>--conf <i>CONFIG_FILE</i></code>	Use <i>CONFIG_FILE</i> instead of the default from the configuration file for audit database configuration.
<code>--check</code>	Check whether the insertion of a record was successful by querying the database after inserting the records. This is used in tests.
<code>--delete</code>	Delete audit records from the database right after inserting them. This is used in tests to avoid filling the database with test records.
<code>--audit-directory <i>DIR</i></code>	Look for audit records in <i>DIR</i> , 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.
<code>--query <i>SQL</i></code>	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.

FILES

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

<code>etc/globus-gram-job-manager.conf</code>	GRAM5 job manager configuration. It includes the default path to the audit directory
<code>etc/globus-gram-audit.conf</code>	Audit configuration. It includes the information needed to contact the audit database.

Name

globus-gatekeeper -- Authorize and execute a grid service on behalf of a user

```
globus-gatekeeper [-help]
[-conf PARAMETER_FILE]
[-test] [ -d | -debug ]
{ -inetd | -f }
[ -p PORT | -port PORT ]
[-home PATH] [ -l LOGFILE | -logfile LOGFILE ]
[-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]
```

Description

The **globus-gatekeeper** program is a meta-server similar to **inetd** or **xinetd** that starts other services after authenticating the TCP connection using GSSAPI.

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 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 <i>PARAMETER_FILE</i>	Load configuration parameters from <i>PARAMETER_FILE</i> . 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.

- 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 754 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 uses its working directory.
- l *LOGFILE*, -logfile *LOGFILE* Write status log entries to *LOGFILE*
- acctfile *ACCTFILE* Set the path to write accounting records to *ACCTFILE*. If not set, no accounting records will be written.
- e *LIBEXECDIR* Look for service executables in *LIBEXECDIR*. If not specified, the default of *HOME/libexec* is used.
- launch_method *fork_and_exit*, *fork_and_wait*, *fork*, *exec*, *execv*, *execvp*, *execve* 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*. If this is a relative path, it is interpreted relative to the *HOME* value. If this is not specified, the default of *HOME/etc/grid-services* is used.
- 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. If not specified, the default of *HOME/etc/grid-mapfile* is used.
- 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_cert_file *TRUSTED_CERT_FILE* OBSOLETE GSI OPTION
- 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 Assume authentication with Kerberos 5 GSSAPI instead of X.509 GSSAPI.

`-globusmap KMAP` Assume authentication with Kerberos 5 GSSAPI instead of X.509 GSSAPI and use *KMAP* as the path to the kerberos principal to POSIX user mapping file.

ENVIRONMENT

If the following variables affect the execution of **globus-gatekeeper**

`X509_CERT_DIR` Directory containing X.509 trust anchors and signing policy files.

`X509_USER_PROXY` Path to file containing an X.509 proxy.

`X509_USER_CERT` Path to file containing an X.509 user certificate.

`X509_USER_KEY` Path to file containing an X.509 user key.

Files

`$GLOBUS_LOCATION/etc/globus-gatekeeper.conf` Default path to gatekeeper configuration file.

`$GLOBUS_LOCATION/etc/grid-services/SERVICENAME` Service configuration for *SERVICENAME*.

See also

`globusrun(1)`, `globus-job-manager(8)`

Name

globus-job-manager -- Execute and monitor jobs

```
globus-job-manager {-type LRM} [-conf CONFIG_PATH] [-help] [-globus-host-manufacturer MANUFACTURER]
[-globus-host-cputype CPUTYPE] [-globus-host-osname OSNAME] [-globus-host-osversion OSVERSION] [-globus-
gatekeeper-host HOST] [-globus-gatekeeper-port PORT] [-globus-gatekeeper-subject SUBJECT] [-home GLOBUS_LOC-
ATION] [-target-globus-location TARGET_GLOBUS_LOCATION] [-condor-arch ARCH] [-condor-os OS] [-history
HISTORY_DIRECTORY] [-scratch-dir-base SCRATCH_DIRECTORY] [-enable-syslog] [-stdio-log LOG_DIRECTORY]
[-log-levels LEVELS] [-state-file-dir STATE_DIRECTORY] [-globus-tcp-port-range PORT_RANGE] [-x509-cert-dir
TRUSTED_CERTIFICATE_DIRECTORY] [-cache-location GASS_CACHE_DIRECTORY] [-k] [-extra-envvars
VAR=VAL, . . .] [-seg-module SEG_MODULE] [-audit-directory AUDIT_DIRECTORY] [-globus-toolkit-version
TOOLKIT_VERSION] [-disable-streaming] [-disable-usagestats] [-usagestats-target TARGET] [-service-tag SER-
VICE_TAG]
```

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 <i>LRM</i>	Execute jobs using the local resource manager named <i>LRM</i> .
-conf <i>CONFIG_PATH</i>	Read additional command-line arguments from the file <i>CONFIG_PATH</i> . If present, this must be the first command-line argument to the globus-job-manager program.
-globus-host-manufacturer <i>MANUFACTURER</i>	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 <i>MANUFACTURER</i>
-globus-host-cputype <i>CPU-TYPE</i>	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 <i>CPUTYPE</i>
-globus-host-osname <i>OS-NAME</i>	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 <i>OSNAME</i>
-globus-host-osversion <i>OSVERSION</i>	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 <i>OSVERSION</i>
-globus-gatekeeper-host <i>HOST</i>	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 <i>HOST</i>

<code>-globus-gatekeeper-port</code> <i>PORT</i>	Indicate the TCP port number of gatekeeper to which jobs are submitted to. This parameter sets the value of the <code>\$(GLOBUS_GATEKEEPER_PORT)</code> RSL substitution to <i>PORT</i>
<code>-globus-gatekeeper-subject</code> <i>SUBJECT</i>	Indicate the X.509 identity of the gatekeeper to which jobs are submitted to. This parameter sets the value of the <code>\$(GLOBUS_GATEKEEPER_SUBJECT)</code> RSL substitution to <i>SUBJECT</i>
<code>-home</code> <i>GLOBUS_LOCATION</i>	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.
<code>-target-globus-location</code> <i>TARGET_GLOBUS_LOCATION</i>	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 <code>\$(GLOBUS_LOCATION)</code> RSL substitution to <i>TARGET_GLOBUS_LOCATION</i>
<code>-history</code> <i>HISTORY_DIRECTORY</i>	Configure the job manager to write job history files to <i>HISTORY_DIRECTORY</i> . These files are described in the FILES section below.
<code>-scratch-dir-base</code> <i>SCRATCH_DIRECTORY</i>	Configure the job manager to use <i>SCRATCH_DIRECTORY</i> as the default scratch directory root if a relative path is specified in the job RSL's <code>scratch_dir</code> attribute.
<code>-enable-syslog</code>	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.
<code>-stdio-log</code> <i>LOG_DIRECTORY</i>	Configure the job manager to write log messages to files in the <i>LOG_DIRECTORY</i> directory. Files will be named <i>LOG_DIRECTORY/gram_YYYYMM-DD.log</i> . Logging is further controlled by the argument to the <code>-log-levels</code> parameter described below. The <i>LOG_DIRECTORY</i> value can include variables derived from the job manager environment using the same syntax as RSL substitutions. For example, <code>-stdio-log \$(HOME)</code> would cause each user's logs to be stored in their individual home directories.
<code>-log-levels</code> <i>LEVELS</i>	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 <code> </code> character. The default value of logging when enabled is <code>FATAL ERROR</code> .
<code>-state-file-dir</code> <i>STATE_DIRECTORY</i>	Configure the job manager to write state files to <i>STATE_DIRECTORY</i> . If not specified, the job manager uses the default of <code>\$(GLOBUS_LOCATION)/tmp/gram_job_state/</code> . This directory must be writable by all users and be on a file system which supports POSIX advisory file locks.
<code>-globus-tcp-port-range</code> <i>PORT_RANGE</i>	Configure the job manager to restrict its TCP/IP communication to use ports in the range described by <i>PORT_RANGE</i> . This value is also made available in the job environment via the <code>GLOBUS_TCP_PORT_RANGE</code> environment variable.
<code>-x509-cert-dir</code> <i>TRUSTED_CERTIFICATE_DIRECTORY</i>	Configure the job manager to search <i>TRUSTED_CERTIFICATE_DIRECTORY</i> for its list of trusted CA certificates and their signing policies. This value is also made available in the job environment via the <code>X509_CERT_DIR</code> environment variable.

<code>-cache-location</code> <code>GASS_CACHE_DIRECTORY</code>	Configure the job manager to use the path <code>GASS_CACHE_DIRECTORY</code> for its temporary GASS-cache files. This value is also made available in the job environment via the <code>GLOBUS_GASS_CACHE_DEFAULT</code> environment variable.
<code>-k</code>	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.
<code>-extra-envvars</code> <code>VAR=VAL,...</code>	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 <code>VAR=VAL</code> pairs, where <code>VAR</code> is the variable name and <code>VAL</code> is the variables value.
<code>-seg-module</code> <code>SEG_MODULE</code>	Configure the job manager to use the schedule event generator module named by <code>SEG_MODULE</code> 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.
<code>-audit-directory</code> <code>AUDIT_DIRECTORY</code>	Configure the job manager to write audit records to the directory named by <code>AUDIT_DIRECTORY</code> . This records can be loaded into a database using the globus-gram-audit program.
<code>-globus-toolkit-version</code> <code>TOOLKIT_VERSION</code>	Configure the job manager to use <code>TOOLKIT_VERSION</code> as the version for audit and usage stats records.
<code>-service-tag</code> <code>SERVICE_TAG</code>	Configure the job manager to use <code>SERVICE_TAG</code> 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 <code>untagged</code> will be used.
<code>-disable-streaming</code>	Configure the job manager to disable file streaming. This is propagated to the LRM script interface but has no effect in GRAM5.
<code>-disable-usagestats</code>	Disable sending of any usage stats data, even if <code>-usagestats-target</code> is present in the configuration.
<code>-usagestats-target</code> <code>TARGET</code>	Send usage packets to a data collection service for analysis. The <code>TARGET</code> string consists of a comma-separated list of <code>HOST:PORT</code> combinations, each containing an optional list of data to send. See Usage Stats Packets ¹ for more information about the tags. Special tag strings of <code>all</code> (which enables all tags) and <code>default</code> may be used, or a sequence of characters for the various tags.
<code>-condor-arch</code> <code>ARCH</code>	Set the architecture specification for condor jobs to be <code>ARCH</code> in job classified ads generated by the GRAM5 condor LRM script. This is required for the condor LRM but ignored for all others.
<code>-condor-os</code> <code>OS</code>	Set the operating system specification for condor jobs to be <code>OS</code> in job classified ads generated by the GRAM5 condor LRM script. This is required for the condor LRM but ignored for all others.

¹ <http://confluence.globus.org/display/~bester/GRAM5+Usage+Stats+Packets>

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_HOST- TP_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.

Files

<code>\$HOME/.globus/job/HOST- NAME/LRM.TAG.red</code>	Job manager delegated user credential.
<code>\$HOME/.globus/job/HOST- NAME/LRM.TAG.lock</code>	Job manager state lock file.
<code>\$HOME/.globus/job/HOST- NAME/LRM.TAG.pid</code>	Job manager pid file.
<code>\$HOME/.globus/job/HOST- NAME/LRM.TAG.sock</code>	Job manager socket for inter-job manager communications.
<code>\$HOME/.globus/job/HOST- NAME/JOB_ID/</code>	Job-specific state directory.
<code>\$HOME/.globus/job/HOST- NAME/JOB_ID/stdin</code>	Standard input which has been staged from a remote URL.
<code>\$HOME/.globus/job/HOST- NAME/JOB_ID/stdout</code>	Standard output which will be staged from a remote URL.

<code>\$HOME/.globus/job/HOST-NAME/JOB_ID/stderr</code>	Standard error which will be staged from a remote URL.
<code>\$HOME/.globus/job/HOST-NAME/JOB_ID/x509_user_proxy</code>	Job-specific delegated credential.
<code>\$GLOBUS_LOCATION/tmp/gram_job_state/job.HOST-NAME.JOB_ID</code>	Job state file.
<code>\$GLOBUS_LOCATION/tmp/gram_job_state/job.HOST-NAME.JOB_ID.lock</code>	Job state lock file. In most cases this will be a symlink to the job manager lock file.
<code>\$GLOBUS_LOCATION/etc/globus-job-manager.conf</code>	Default location of the global job manager configuration file.
<code>\$GLOBUS_LOCATION/etc/grid-services/jobmanager-LRM</code>	Default location of the LRM-specific gatekeeper configuration file.

See Also

globusrun(1), globus-gatekeeper(8), globus-personal-gatekeeper(1), globus-gram-audit(8)

Name

globus-job-manager-event-generator -- Create LRM-independent SEG files for the job manager to use

globus-job-manager-event-generator [-help] {-scheduler *LRM*} [-background] [-pidfile *PIDPATH*]

Description

The **globus-job-manager-event-generator** program is a utility which uses LRM-specific SEG parsers to generate a LRM-independent log file that a job manager instance can use to process job status change events. This program runs independently of all **globus-job-manager** instances so that only one process needs to deal with the LRM interface. The **globus-job-manager-event-generator** program can be run as a privileged user if required to interface with the LRM.

In order for **globus-job-manager-event-generator** to handle events for a particular LRM, the `globus_scheduler_event_generator_job_manager_setup` setup package must be configured after the LRM-specific setup package has been run. This can be forced by **gpt-postinstall -force** or running the command **cd \$GLOBUS_LOCATION/setup/globus; ./setup-seg-job-manager.pl**.

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

- help Print command-line option summary and exit.
- scheduler *LRM* Process events for the local resource manager named by *LRM*.
- background Run **globus-job-manager-event-generator** as a background process. It will fork a new process, print out its process ID and then the original process will terminate.
- pidfile *PIDPATH* Write the process ID of an instance of **globus-job-manager-event-generator** to the file named by *PIDPATH*. This file can be used to kill or monitor the **globus-job-manager-event-generator** process.

Files

globus-job-manager-seg.conf Configuration file for **globus-job-manager-event-generator**. Each line consists of a string of the form `LRM_log_path=PATH`, which indicates the directory containing LRM-independent format SEG log files for the LRM. This file is created by the running the `globus_scheduler_event_generator_job_manager_setup` setup package.

See Also

globus-scheduler-event-generator(8), globus-job-manager(8)

Name

globus-fork-starter -- Start and monitor a fork job

globus-fork-starter

Description

The **globus-fork-starter** program is executes jobs specified on its standard input stream, recording the job state changes to a file defined in the `$GLOBUS_LOCATION/etc/globus-fork.conf` configuration file. It runs until its standard input stream is closed and all jobs it is managing have terminated. The log generated by this program can be used by the SEG to provide job state changes and exit codes to the GRAM service. The **globus-fork-starter** program is typically started by the fork GRAM module.

The **globus-fork-starter** program expects its input to be a series of task definitions, separated by the newline character, each representing a separate job. Each task definition contains a number of fields, separated by the colon character. The first field is always the literal string `100` indicating the message format, the second field is a unique job tag that will be distinguish the reply from this program when multiple jobs are submitted. The rest of fields contain attribute bindings. The supported attributes are:

<code>directory</code>	Working directory of the job
<code>environment</code>	Comma-separated list of strings defining environment variables. The form of these strings is <code>var=value</code>
<code>count</code>	Number of processes to start
<code>executable</code>	Full path to the executable to run
<code>arguments</code>	Comma-separated list of command-line arguments for the job
<code>stdin</code>	Full path to a file containing the input of the job
<code>stdout</code>	Full path to a file to write the output of the job to
<code>stderr</code>	Full path to a file to write the error stream of the job

Within each field, the following characters may be escaped by preceding them with the backslash character:

- backslash (`\`)
- semicolon (`;`)
- comma (`,`)
- equal (`=`)

Additionally, newline can be represented within a field by using the escape sequence `\n`.

For each job the **globus-fork-starter** processes, it replies by writing a single line to standard output. The replies again consist of a number of fields separated by the semicolon character.

For a successful job start, the first field of the reply is the literal `101`, the second field is the tag from the input, and the third field is a comma-separated list of SEG job identifiers which consist the concatenation of a UUID and a process id. The **globus-fork-starter** program will write state changes to the SEG log using these job identifiers.

For a failure, the first field of the reply is the literal 102, the second field is the tag from the input, the third field is the integer representation of a GRAM error code, and the fourth field is a string explaining the error.

ENVIRONMENT

If the following variables affect the execution of **globus-fork-starter**

GLOBUS_LOCATION Path to Globus Toolkit installation. This is used to locate the `globus-fork.conf` configuration file.

Files

`$GLOBUS_LOCATION/etc/globus-fork.conf` Path to fork SEG configuration file.

Chapter 2. Troubleshooting

For a list of error codes generated by GRAM5, see [Section 2, “Errors”](#).

For information about sys admin logging, see [Admin Debugging](#) in the GRAM5 Admin Guide.

1. Troubleshooting tips

In case you run into problems you can do the following

- Check the GRAM5 documentation. Maybe you'll find hints here to solve your problem.
- Check the GRAM5 log for errors.

In case you don't find anything suspicious you can increase the log-level of GRAM5 or other relevant components. Maybe the additional logging-information will tell you what's going wrong.

- Send e-mails to <gram-user@globus.org>. You'll have to subscribe to a list before you can send an e-mail to it. See [here](#)¹ for general e-mail lists and information on how to subscribe to a list and [here](#)² for GRAM specific lists.

¹ http://dev.globus.org/wiki/Mailing_Lists

² http://dev.globus.org/wiki/GRAM#Mailing_Lists

2. Errors

Table 2.1. GRAM5 Errors

Error Code	Reason	Possible Solutions
1	one of the RSL parameters is not supported	Check RSL documentation
2	the RSL length is greater than the maximum allowed	Use RSL substitutions to reduce length of RSL strings
3	an I/O operation failed	Enable trace logging and report to gram-dev@globus.org
4	jobmanager unable to set default to the directory requested	Check that RSL <code>directory</code> attribute refers to a directory that exists on the target system.
5	the executable does not exist	Check that the RSL <code>executable</code> attribute refers to an executable that exists on the target system.
6	of an unused <code>INSUFFICIENT_FUNDS</code>	Unimplemented feature.
7	authentication with the remote server failed	Check that the contact string contains the proper X.509 DN.
8	the user cancelled the job	Don't cancel jobs you want to complete.
9	the system cancelled the job	Check RSL requirements such as maximum time and memory are valid for the job.
10	data transfer to the server failed	Check gatekeeper and/or job manager logs to see why the process failed.
11	the stdin file does not exist	Check that the RSL <code>stdin</code> attribute refers to a file that exists on the target system or has a valid ftp, gsiftp, http, or https URL.
12	the connection to the server failed (check host and port)	Check that the service is running on the expected TCP/IP port. Check that no firewall prevents contacting that TCP/IP port. Check <code>\$GLOBUS_LOCATION/var/globus-gatekeeper.log</code> for runtime configuration errors.
13	the provided RSL 'maxtime' value is not an integer	Check that the RSL <code>maxtime</code> value evaluates to an integer.
14	the provided RSL 'count' value is not an integer	Check that the RSL <code>count</code> value evaluates to an integer.
15	the job manager received an invalid RSL	Check that the RSL string can be parsed by using <code>globusrun -p RSL</code> .
16	the job manager failed in allowing others to make contact	Check job manager log.
17	the job failed when the job manager attempted to run it	Verify that the LRM is configured properly.
18	an invalid paradyn was specified	OBSOLETE IN GRAM2
19	the provided RSL 'jobtype' value is invalid	The RSL <code>jobtype</code> attribute is not indicated as supported by the LRM. Valid <code>jobtype</code> values are <code>single</code> , <code>multiple</code> , <code>mpi</code> , and <code>condor</code> .
20	the provided RSL 'myjob' value is invalid	OBSOLETE IN GRAM5

Error Code	Reason	Possible Solutions
21	the job manager failed to locate an internal script argument file	Check that <code>\$GLOBUS_LOCATION/libexec/globus-job-manager-script.pl</code> exists and is executable. Check that the LRM-specific perl module is located in <code>\$GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/</code> 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.
22	the job manager failed to create an internal script argument file	Check that your home directory is writable and not full.
23	the job manager detected an invalid job state	Check job manager logs.
24	the job manager detected an invalid script response	Check job manager logs. This is likely a bug in the LRM script.
25	the job manager detected an invalid script status	Check job manager logs. This is likely a bug in the LRM script.
26	the provided RSL 'jobtype' value is not supported by this job manager	Check that the RSL <code>jobtype</code> attribute is implemented by the LRM script. Note that some job types require configuration
27	unused ERROR_UNIMPLEMENTED	LRM does not support some feature included in the job request.
28	the job manager failed to create an internal script submission file	Check that the user's home file system is not full. Check job manager log
29	the job manager cannot find the user proxy	Check that client is delegating a proxy when authenticating with the gatekeeper. Check that the user's home filesystem and the <code>/tmp</code> file system are not full.
30	the job manager failed to open the user proxy	Check that the user's home filesystem and the <code>/tmp</code> file system are not full.
31	the job manager failed to cancel the job as requested	Check that the user's home filesystem and the <code>/tmp</code> file system are not full.
32	system memory allocation failed	Check job manager log for details.
33	the interprocess job communication initialization failed	OBSOLETE IN GRAM5
34	the interprocess job communication setup failed	OBSOLETE IN GRAM5
35	the provided RSL 'host count' value is invalid	Check that the RSL <code>host_count</code> attribute evaluates to an integer.
36	one of the provided RSL parameters is unsupported	Check job manager log for details about invalid parameter.
37	the provided RSL 'queue' parameter is invalid	Check that the RSL <code>queue</code> attribute evaluates to a string that corresponds to an LRM-specific queue name.
38	the provided RSL 'project' parameter is invalid	Check that the RSL <code>project</code> attribute evaluates to a string that corresponds to an LRM-specific project name.

Error Code	Reason	Possible Solutions
39	the provided RSL string includes variables that could not be identified	Check that all RSL substitutions are defined before being used in the job description.
40	the provided RSL 'environment' parameter is invalid	Check that the RSL <code>environment</code> attribute contains a sequence of <code>VARIABLE VALUE</code> pairs.
41	the provided RSL 'dryrun' parameter is invalid	Remove the RSL <code>dryrun</code> attribute from the job description.
42	the provided RSL is invalid (an empty string)	Include a non-empty RSL string in your job submission request.
43	the job manager failed to stage the executable	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.
44	the job manager failed to stage the stdin file	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.
45	the requested job manager type is invalid	OBSOLETE IN GRAM5
46	the provided RSL 'arguments' parameter is invalid	OBSOLETE IN GRAM2
47	the gatekeeper failed to run the job manager	Check the gatekeeper or job manager logs for more information.
48	the provided RSL could not be properly parsed	Check that the RSL string can be parsed by using globsrun -p RSL .
49	there is a version mismatch between GRAM components	Ask system administrator to upgrade GRAM service to GRAM2 or GRAM5
50	the provided RSL 'arguments' parameter is invalid	Check that the RSL <code>arguments</code> attribute evaluates to a sequence of strings.
51	the provided RSL 'count' parameter is invalid	Check that the RSL <code>count</code> attribute evaluates to a positive integer value.
52	the provided RSL 'directory' parameter is invalid	Check that the RSL <code>directory</code> attribute evaluates to a string.
53	the provided RSL 'dryrun' parameter is invalid	Check that the RSL <code>dryrun</code> attribute evaluates to either <code>yes</code> or <code>no</code> .
54	the provided RSL 'environment' parameter is invalid	Check that the RSL <code>environment</code> attribute evaluates to a sequence of <code>VARIABLE, VALUE</code> pairs.
55	the provided RSL 'executable' parameter is invalid	Check that the RSL <code>executable</code> attribute evaluates to a string value.
56	the provided RSL 'host_count' parameter is invalid	Check that the RSL <code>host_count</code> attribute evaluates to a positive integer value.
57	the provided RSL 'jobtype' parameter is invalid	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>

Error Code	Reason	Possible Solutions
58	the provided RSL 'maxtime' parameter is invalid	Check that the RSL <code>maxtime</code> attribute evaluates to a positive integer value.
59	the provided RSL 'myjob' parameter is invalid	OBSOLETE IN GRAM5.
60	the provided RSL 'paradyn' parameter is invalid	OBSOLETE IN GRAM2.
61	the provided RSL 'project' parameter is invalid	Check that the RSL <code>project</code> attribute evaluates to a string value.
62	the provided RSL 'queue' parameter is invalid	Check that the RSL <code>queue</code> attribute evaluates to a string value.
63	the provided RSL 'stderr' parameter is invalid	Check that the RSL <code>stderr</code> attribute evaluates to a string value or a sequence of <i>DESTINATION</i> URLs with optional <i>CACHE_TAG</i> string parameters.
64	the provided RSL 'stdin' parameter is invalid	Check that the RSL <code>stdin</code> attribute evaluates to a string value.
65	the provided RSL 'stdout' parameter is invalid	Check that the RSL <code>stdout</code> attribute evaluates to a string value or a sequence of <i>DESTINATION</i> URLs with optional <i>CACHE_TAG</i> string parameters.
66	the job manager failed to locate an internal script	Check job manager log for more details.
67	the job manager failed on the system call <code>pipe()</code>	OBSOLETE IN GRAM5
68	the job manager failed on the system call <code>fcntl()</code>	OBSOLETE IN GRAM2
69	the job manager failed to create the temporary stdout filename	OBSOLETE IN GRAM5
70	the job manager failed to create the temporary stderr filename	OBSOLETE IN GRAM5
71	the job manager failed on the system call <code>fork()</code>	OBSOLETE IN GRAM2
72	the executable file permissions do not allow execution	Check that the RSL <code>executable</code> attribute refers to an executable program or script.
73	the job manager failed to open stdout	Check that the RSL <code>stdout</code> attribute refers to one or more valid destination files or URLs.
74	the job manager failed to open stderr	Check that the RSL <code>stderr</code> attribute refers to one or more valid destination files or URLs.
75	the cache file could not be opened in order to relocate the user proxy	Check that the user's home directory is writable and not full on the GRAM5 service node.
76	cannot access cache files in <code>~/globus/.gass_cache</code> , check permissions, quota, and disk space	Check that the user's home directory is writable and not full on the GRAM5 service node.
77	the job manager failed to insert the contact in the client contact list	Check job manager log

Error Code	Reason	Possible Solutions
78	the contact was not found in the job manager's client contact list	Don't attempt to unregister callback contacts that are not registered
79	connecting to the job manager failed. Possible reasons: job terminated, invalid job contact, network problems, ...	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.
80	the syntax of the job contact is invalid	Check the syntax of job contact string.
81	the executable parameter in the RSL is undefined	Include the RSL <code>executable</code> in all job requests.
82	the job manager service is misconfigured. <code>condor arch</code> undefined	Add the <code>-condor-arch</code> to the command-line or configuration file for a job manager configured to use the <code>condor</code> LRM.
83	the job manager service is misconfigured. <code>condor os</code> undefined	Add the <code>-condor-os</code> to the command-line or configuration file for a job manager configured to use the <code>condor</code> LRM.
84	the provided RSL 'min_memory' parameter is invalid	Check that the RSL <code>min_memory</code> attribute evaluates to a positive integer value.
85	the provided RSL 'max_memory' parameter is invalid	Check that the RSL <code>max_memory</code> attribute evaluates to a positive integer value.
86	the RSL 'min_memory' value is not zero or greater	Check that the RSL <code>min_memory</code> attribute evaluates to a positive integer value.
87	the RSL 'max_memory' value is not zero or greater	Check that the RSL <code>max_memory</code> attribute evaluates to a positive integer value.
88	the creation of a HTTP message failed	Check job manager log.
89	parsing incoming HTTP message failed	Check job manager log.
90	the packing of information into a HTTP message failed	Check job manager log.
91	an incoming HTTP message did not contain the expected information	Check job manager log.
92	the job manager does not support the service that the client requested	Check that the client is talking to the correct service
93	the gatekeeper failed to find the requested service	OBSOLETE IN GRAM2
94	the jobmanager does not accept any new requests (shutting down)	Execute queries before the job has been cleaned up.
95	the client failed to close the listener associated with the callback URL	Call <code>globus_gram_client_callback_disallow()</code> with a valid the callback contact.
96	the gatekeeper contact cannot be parsed	Check the syntax of the gatekeeper contact string you are attempting to contact.
97	the job manager could not find the 'poe' command	OBSOLETE IN GRAM2
98	the job manager could not find the 'mpirun' command	Configure the LRM script with <code>mpirun</code> in your path.

Error Code	Reason	Possible Solutions
99	the provided RSL 'start_time' parameter is invalid	OBSOLETE IN GRAM2
100	the provided RSL 'reservation_handle' parameter is invalid	OBSOLETE IN GRAM2
101	the provided RSL 'max_wall_time' parameter is invalid	Check that the RSL <code>max_wall_time</code> attribute evaluates to a positive integer.
102	the RSL 'max_wall_time' value is not zero or greater	Check that the RSL <code>max_wall_time</code> attribute evaluates to a positive integer.
103	the provided RSL 'max_cpu_time' parameter is invalid	Check that the RSL <code>max_cpu_time</code> attribute evaluates to a positive integer.
104	the RSL 'max_cpu_time' value is not zero or greater	Check that the RSL <code>max_cpu_time</code> attribute evaluates to a positive integer.
105	the job manager is misconfigured, a scheduler script is missing	Check that the administrator has configured the LRM by running its setup script.
106	the job manager is misconfigured, a scheduler script has invalid permissions	Check that the administrator has installed the <code>GLLOBUS_LOCATION/libexec/globus-job-manager-script.pl</code> script. Check that the file system containing that script allows file execution.
107	the job manager failed to signal the job	OBSOLETE IN GRAM2
108	the job manager did not recognize/support the signal type	Check that your signal operation is using the correct signal constant.
109	the job manager failed to get the job id from the local scheduler	OBSOLETE IN GRAM2
110	the job manager is waiting for a commit signal	Send a two-phase commit signal to the job manager to acknowledge receiving the job contact from the job manager.
111	the job manager timed out while waiting for a commit signal	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.
112	the provided RSL 'save_state' parameter is invalid	Check that the RSL <code>save_state</code> attribute is set to <code>yes</code> or <code>no</code> .
113	the provided RSL 'restart' parameter is invalid	Check that the RSL <code>restart</code> attribute evaluates to a string containing a job contact string.
114	the provided RSL 'two_phase' parameter is invalid	Check that the RSL <code>two_phase</code> attribute evaluates to a positive integer.
115	the RSL 'two_phase' value is not zero or greater	Check that the RSL <code>two_phase</code> attribute evaluates to a positive integer.
116	the provided RSL 'stdout_position' parameter is invalid	OBSOLETE IN GRAM5
117	the RSL 'stdout_position' value is not zero or greater	OBSOLETE IN GRAM5

Error Code	Reason	Possible Solutions
118	the provided RSL 'stderr_position' parameter is invalid	OBSOLETE IN GRAM5
119	the RSL 'stderr_position' value is not zero or greater	OBSOLETE IN GRAM5
120	the job manager restart attempt failed	OBSOLETE IN GRAM2
121	the job state file doesn't exist	Check that the job contact you are trying to restart matches one that the job manager returned to you.
122	could not read the job state file	Check that the state file directory is not full.
123	could not write the job state file	Check that the state file directory is not full.
124	old job manager is still alive	Contact the returned job manager contact to manage the job you are trying to restart.
125	job manager state file TTL expired	OBSOLETE in GRAM2
126	it is unknown if the job was submitted	Check job manager log.
127	the provided RSL 'remote_io_url' parameter is invalid	Check that the RSL <code>remote_io_url</code> attribute evaluates to a string value.
128	could not write the remote io url file	Check that the user's home file system on the job manager service node is writable and not full.
129	the standard output/error size is different	Send a stdio update signal to redirect the job manager output to a new URL
130	the job manager was sent a stop signal (job is still running)	Submit a restart request to monitor the job.
131	the user proxy expired (job is still running)	Generate a new proxy and then submit a restart request to monitor the job.
132	the job was not submitted by original job-manager	OBSOLETE IN GRAM2
133	the job manager is not waiting for that commit signal	Do not send a commit signal to a job that is not waiting for a commit signal.
134	the provided RSL scheduler specific parameter is invalid	Check the LRM-specific documentation to determine what values are legal for the RSL extensions implemented by the LRM.
135	the job manager could not stage in a file	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.
136	the scratch directory could not be created	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.
137	the provided 'gass_cache' parameter is invalid	Check that the RSL <code>gass_cache</code> attribute evaluates to a string.
138	the RSL contains attributes which are not valid for job submission	Do not use restart- or signal-only RSL attributes when submitting a job.

Error Code	Reason	Possible Solutions
139	the RSL contains attributes which are not valid for stdio update	Do not use submit- or restart-only RSL attributes when sending a stdio update signal to a job.
140	the RSL contains attributes which are not valid for job restart	Do not use submit- or signal-only RSL attributes when restarting a job.
141	the provided RSL 'file_stage_in' parameter is invalid	Check that the RSL <code>file_stage_in</code> attribute evaluates to a sequence of <i>SOURCE DESTINATION</i> pairs.
142	the provided RSL 'file_stage_in_shared' parameter is invalid	Check that the RSL <code>file_stage_in_shared</code> attribute evaluates to a sequence of <i>SOURCE DESTINATION</i> pairs.
143	the provided RSL 'file_stage_out' parameter is invalid	Check that the RSL <code>file_stage_out</code> attribute evaluates to a sequence of <i>SOURCE DESTINATION</i> pairs.
144	the provided RSL 'gass_cache' parameter is invalid	Check that the RSL <code>gass_cache</code> attribute evaluates to a string.
145	the provided RSL 'file_cleanup' parameter is invalid	Check that the RSL <code>file_clean_up</code> attribute evaluates to a sequence of strings.
146	the provided RSL 'scratch_dir' parameter is invalid	Check that the RSL <code>scratch_dir</code> attribute evaluates to a string.
147	the provided scheduler-specific RSL parameter is invalid	Check the LRM-specific documentation to determine what values are legal for the RSL extensions implemented by the LRM.
148	a required RSL attribute was not defined in the RSL spec	Check that the RSL <code>executable</code> attribute is present in your job request RSL. Check that the RSL <code>restart</code> attributes is present in your restart RSL.
149	the <code>gass_cache</code> attribute points to an invalid cache directory	Check that the RSL <code>gass_cache</code> attributes evaluates to a directory that exists or can be created. Check that the user's home file system is writable and not full.
150	the provided RSL 'save_state' parameter has an invalid value	Check that the RSL <code>save_state</code> attribute has a value of <code>yes</code> or <code>no</code> .
151	the job manager could not open the RSL attribute validation file	Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf</code> is present and readable on the job manager service node. Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf</code> is readable on the job manager service node if present.
152	the job manager could not read the RSL attribute validation file	Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf</code> is valid. Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf</code> is valid if present.
153	the provided RSL 'proxy_timeout' is invalid	Check that RSL <code>proxy_timeout</code> attribute evaluates to a positive integer.
154	the RSL 'proxy_timeout' value is not greater than zero	Check that RSL <code>proxy_timeout</code> attribute evaluates to a positive integer.

Error Code	Reason	Possible Solutions
155	the job manager could not stage out a file	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.
156	the job contact string does not match any which the job manager is handling	Check that the job contact string matches one returned from a job request.
157	proxy delegation failed	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.
158	the job manager could not lock the state lock file	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.
159	an invalid globus_io_clientattr_t was used.	Check that you have initialized the globus_io_clientattr_t attribute prior to using it with the GRAM client API.
160	an null parameter was passed to the gram library	Check that you are passing legal values to all GRAM API calls.
161	the job manager is still streaming output	OBSOLETE IN GRAM5
162	the authorization system denied the request	Check with your GRAM system administrator to allow a particular certificate to be authorized.
163	the authorization system reported a failure	Check with your system administrator to verify that the authorization system is configured properly.
164	the authorization system denied the request - invalid job id	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.
165	the authorization system denied the request - not authorized to run the specified executable	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.
166	the provided RSL 'user_name' parameter is invalid.	Check that the RSL user_name attribute evaluates to a string.
167	the job is not running in the account named by the 'user_name' parameter.	Ask with the GRAM system administrator to add an authorization entry to allow your credential to run jobs as the specified user account.

Chapter 3. Known Problems in GRAM5

1. Known Problems

The following problems and limitations are known to exist for GRAM5 at the time of the 5.0.0 release:

1.1. Limitations

- [list limitations]

1.2. Outstanding bugs

- [Bug 108](#):¹ Fork perl zombies
- [Bug 106](#):² Fix test failures with SGE LRM adapter
- [Bug 105](#):³ Held Condor jobs should be reported as SUSPENDED
- [Bug 104](#):⁴ globus-job-manager-event-generator loads all historical events the first time run
- [Bug 103](#):⁵ Ease two phase end commit timeout
- [Bug 102](#):⁶ Fix Two Phase Commit Semantics for Failed Jobs
- [Bug 100](#):⁷ one of the RSL parameters is not supported error doesn't indicate which it is
- [Bug 99](#):⁸ Add a high-level diagram for the approach doc
- [Bug 98](#):⁹ Add Condor-G doc for using GRAM 2 and 5
- [Bug 96](#):¹⁰ GRAM-106 SGE LRM mishandles invalid environment definition
- [Bug 95](#):¹¹ GRAM-106 SGE LRM doesn't check for executable permissions
- [Bug 94](#):¹² GRAM-106 SGE LRM doesn't check for executable existence
- [Bug 93](#):¹³ GRAM-106 SGE LRM script doesn't handle environment vars with whitespace
- [Bug 92](#):¹⁴ stdout to local file doesn't work if count >1

¹ <http://jira.globus.org/browse/GRAM-108>

² <http://jira.globus.org/browse/GRAM-106>

³ <http://jira.globus.org/browse/GRAM-105>

⁴ <http://jira.globus.org/browse/GRAM-104>

⁵ <http://jira.globus.org/browse/GRAM-103>

⁶ <http://jira.globus.org/browse/GRAM-102>

⁷ <http://jira.globus.org/browse/GRAM-100>

⁸ <http://jira.globus.org/browse/GRAM-99>

⁹ <http://jira.globus.org/browse/GRAM-98>

¹⁰ <http://jira.globus.org/browse/GRAM-96>

¹¹ <http://jira.globus.org/browse/GRAM-95>

¹² <http://jira.globus.org/browse/GRAM-94>

¹³ <http://jira.globus.org/browse/GRAM-93>

¹⁴ <http://jira.globus.org/browse/GRAM-92>

- [Bug 88:](#)¹⁵ Missed two phase commit causes job to not be destroyed
- [Bug 86:](#)¹⁶ Prioritize script invocations to improve throughput
- [Bug 80:](#)¹⁷ GRAM 5 beta2 release
- [Bug 79:](#)¹⁸ Add support for OSG's "NFS Lite" concept
- [Bug 77:](#)¹⁹ GRAM zombie
- [Bug 71:](#)²⁰ GRAM protocol test package contains expired test certificate
- [Bug 70:](#)²¹ globus-job-status acts strange for completed jobs in GRAM5
- [Bug 69:](#)²² globus-job-get-output -f doesn't work in GRAM5
- [Bug 68:](#)²³ Bad error when proxy is too short-lived
- [Bug 54:](#)²⁴ make globus-job-manager-event-generator not require configuration by default
- [Bug 53:](#)²⁵ Generalize log path configuration
- [Bug 51:](#)²⁶ configurable control of number of perl scripts that can run simultaneously
- [Bug 47:](#)²⁷ simplify the throughput tester program and use improved version as doc
- [Bug 24:](#)²⁸ Debug/verbose flags for globusrun, globus-job-run
- [Bug 23:](#)²⁹ Improved error codes and error reporting for users
- [Bug 22:](#)³⁰ client connections can't be timed out
- [Bug 15:](#)³¹ transition from httpg to https
- [Bug 14:](#)³² increase availability of GRAM in linux distributions
- [Bug 12:](#)³³ Gatekeeper's syslog output cannot be controlled
- [Bug 5:](#)³⁴ Add gram-level prologue and epilogue script execution for mpi jobs

¹⁵ <http://jira.globus.org/browse/GRAM-88>

¹⁶ <http://jira.globus.org/browse/GRAM-86>

¹⁷ <http://jira.globus.org/browse/GRAM-80>

¹⁸ <http://jira.globus.org/browse/GRAM-79>

¹⁹ <http://jira.globus.org/browse/GRAM-77>

²⁰ <http://jira.globus.org/browse/GRAM-71>

²¹ <http://jira.globus.org/browse/GRAM-70>

²² <http://jira.globus.org/browse/GRAM-69>

²³ <http://jira.globus.org/browse/GRAM-68>

²⁴ <http://jira.globus.org/browse/GRAM-54>

²⁵ <http://jira.globus.org/browse/GRAM-53>

²⁶ <http://jira.globus.org/browse/GRAM-51>

²⁷ <http://jira.globus.org/browse/GRAM-47>

²⁸ <http://jira.globus.org/browse/GRAM-24>

²⁹ <http://jira.globus.org/browse/GRAM-23>

³⁰ <http://jira.globus.org/browse/GRAM-22>

³¹ <http://jira.globus.org/browse/GRAM-15>

³² <http://jira.globus.org/browse/GRAM-14>

³³ <http://jira.globus.org/browse/GRAM-12>

³⁴ <http://jira.globus.org/browse/GRAM-5>

- Bug 4:³⁵ Add support for a "managed fork" service
- Bug 2:³⁶ Investigate how to setup GRAM5 services in a HA setup

³⁵ <http://jira.globus.org/browse/GRAM-4>

³⁶ <http://jira.globus.org/browse/GRAM-2>

Chapter 4. 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 ../../Usage_Stats.html

Glossary

Index

B

bugs
 outstanding, 56

E

errors, 46

J

jobs
 preparing
 delegating credentials, 2
 generate valid proxy, 1

L

limitations, 56

T

troubleshooting, 45
 check documentation, 45
 errors, 45
 gram log, 45
 mailing lists, 45

U

usage statistics, 59

GT 5.0.0 GRAM5: Developer's Guide

GT 5.0.0 GRAM5: Developer's Guide

Introduction

This guide is intended to help a developer create GRAM5 clients in C. It provides an overview of the concepts and APIs needed to interact with GRAM services.

Table of Contents

1. Before you begin	1
1. Feature summary	1
2. Tested platforms	1
3. Backward compatibility summary	2
4. Technology dependencies	2
5. Security Considerations	2
2. GRAM5 Concepts for Developers	3
1. Blocking and Nonblocking Function Variants	3
2. Service Contact Strings	3
3. Job State Callbacks and Polling	3
4. Credential Management	4
5. RSL	4
3. Basic GRAM Client Scenarios	5
1. "Ping" a Job Manager	5
2. Check a Job Manager Version	6
3. Submitting a Job	8
4. Submitting a Job and Processing Job State Callbacks	10
5. Polling Job Status	13
6. Canceling a Job	15
7. Refreshing Job Credential	16
4. Advanced GRAM Client Scenarios	18
1. Non-blocking Job Submission	18
2. Custom Security Attributes	21
3. Modifying RSL	24
5. Tutorials	28
6. APIs	29
1. Programming Model Overview	29
7. RSL Specification v1.1	30
1. RSL Syntax Overview	30
2. RSL Tokenization Overview	31
3. RSL Substitution Semantics	32
4. RSL Attribute Summary	32
5. Simple RSL Examples	35
6. RSL grammar and tokenization rules	36
8. Debugging	38
1. Basic Debugging Methods	38
2. Advanced Debugging Methods	39
9. Troubleshooting	40
1. Troubleshooting tips	40
2. Errors	41
10. Semantics and syntax of protocols	51
1. GRAM5 Protocol	51
11. Related Documentation	59
12. Internal Components	60
Glossary	61
Index	62

List of Figures

10.1. GRAM State Transitions	58
------------------------------------	----

List of Tables

2.1. GRAM Contact String Types 3
6.1. GRAM Client APIs 29
9.1. GRAM5 Errors 42
10.1. GRAM Job States 58

List of Examples

7.1. Quoted Literal Examples	32
7.2. GRAM5 Job Request Examples	36

Chapter 1. Before you begin

1. Feature summary

New Features new since 4.2.x

- Server-side architectural changes to improve scalability, performance, and reliability
- Improved error notification protocol compared to GRAM2
- Teragrid Gateway Identity support for job auditing
- Usage stats messages
- Added support for Sun Grid Engine (SGE)

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

- The GRAM5 client tools have dropped support for the Duroc API for task coallocation
- The GRAM5 service no longer streams output and error during job execution; instead this data is send after the job terminates
- The GRAM5 service no longer provides intra-job communication via the DUCT API
- The GRAM5 does not rely on XML schemas and WSDL service definitions

2. Tested platforms

Tested platforms for GRAM5:

- Linux
 - CentOS 5.3 x86_64
 - Debain 4.0 x86_64
- Mac OS X
 - Mac OS X 10.5.8

3. Backward compatibility summary

Protocol changes in GRAM since GT4.2.x 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 coallocation 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.

4. Technology dependencies

GRAM depends on the following GT components:

- Globus Common
- GSI C
- GridFTP server

5. Security Considerations

No special security considerations exist at this time.

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

Type	Meaning
Gatekeeper Service Contact	This string describes how to contact a gatekeeper service. It is used to submit jobs, send "ping" 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 GRAM5 User's Guide .
Callback Contact	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.
Job Contact	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.

3. Job State Callbacks and Polling

GRAM clients 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.

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.

```
/*
 * 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;
    }
    /*
```

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;
}
/* End of gram_ping_example.c */
```

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.

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

#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char *argv[])
{
    int rc;
    globus_hashtable_t extensions = NULL;
    globus_gram_protocol_extension_t * extension_value;

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

² [gram_version_example.c](#)

```

    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);
out:
    return rc;
}
/* 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](#)⁴ 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>

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[])
```

⁴ 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)\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\n");
        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 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.

```

```
    */
    rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
out:
    return rc;
}
/* End of gram_submit_and_wait_example.c */
```

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
```

⁵ [gram_poll_example.c](#)

```

    * this program.
    */
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.

```

/*
 * 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 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;
    }

    /*
     * Cancel the job named by the first argument to
     * this program.
     */
    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);
    }

    /*
     * Deactivating the module allows it to free memory and close network

```

⁶ gram_cancel_example.c

```
    * connections.
    */
    rc = globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
out:
    return rc;
}
/* End of gram_cancel_example.c */
```

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](#)⁷ 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;

    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]);

    /*
     * 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;
    }
    /*
```

⁷ [gram_refresh_example.c](#)

```
* 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.

```
/*
 * 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)
```

¹ [gram_nonblocking_submit_example.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("Submitting %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)

```

```

{
    fprintf(stderr, "Error initializing condition variable %d\n", rc);

    goto destroy_mutex;
}
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);
}
globus_mutex_unlock(&monitor.mutex);

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

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_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.

```

/*
 * 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
    {

```

² [gram_attr_example.c](#)

```
        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;
    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("Submitting 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 example³](#) can be downloaded.

```

/*
 * These headers contain declarations for the globus_module,
 * the GRAM Client, RSL, and protocol functions
 */
#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
 * into the RSL. We'll start by making an empty sequence
 */
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
 * attribute.
 */
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 */

```

Chapter 5. Tutorials

The following tutorials are available for GRAM5 developers:

- [GRAM5 Scheduler Interface Tutorial](#)¹

¹ scheduler-tutorial.html

Chapter 6. APIs

1. Programming Model Overview

1.1. C API Documentation Links

Table 6.1. GRAM Client APIs

Name	Purpose
<u>GRAM Protocol</u> ¹	Low-level functions for processing GRAM protocol messages. Symbolic constants for RSL attributes, signals, and job states.
<u>GRAM Client</u> ²	Functions for submitting job requests, sending signals, and listening for job state updates.
<u>RSL</u> ³	Functions for parsing and manipulating job specifications in the RSL language.

¹ http://www.globus.org/api/c-globus-5.0.0/globus_gram_protocol/html/main.html

² http://www.globus.org/api/c-globus-5.0.0/globus_gram_client/html/main.html

³ http://www.globus.org/api/c-globus-5.0.0/globus_rsl/html/main.html

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 5.0.0 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.

4.1. GRAM5 Common RSL Attributes

<code>arguments</code>	The command line arguments for the executable. Use quotes, if a space is required in a single argument.
<code>count</code>	The number of executions of the executable.
<code>directory</code>	Specifies the path of the directory the jobmanager will use as the default directory for the requested job.

<code>dry_run</code>	If <code>dryrun = yes</code> then the jobmanager will not submit the job for execution and will return success.
<code>environment</code>	The environment variables that will be defined for the executable in addition to default set that is given to the job by the jobmanager.
<code>executable</code>	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.
<code>file_clean_up</code>	Specifies a list of files which will be removed after the job is completed.
<code>file_stage_in</code>	Specifies a list of ("remote URL" "local file") pairs which indicate files to be staged to the nodes which will run the job.
<code>file_stage_in_shared</code>	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.
<code>file_stage_out</code>	Specifies a list of ("local file" "remote URL") pairs which indicate files to be staged from the job to a GASS-compatible file server.
<code>gass_cache</code>	Specifies location to override the GASS cache location.
<code>gram_my_job</code>	Obsolete and ignored.
<code>host_count</code>	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.
<code>job_type</code>	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.)
<code>library_path</code>	Specifies a list of paths to be appended to the system-specific library path environment variables.
<code>max_cpu_time</code>	Explicitly set the maximum cputime for a single execution of the executable. The units is in minutes. The value will go through an <code>atoi()</code> conversion in order to get an integer. If the GRAM scheduler cannot set cputime, then an error will be returned.
<code>max_memory</code>	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 <code>atoi()</code> conversion in order to get an integer. If the GRAM scheduler cannot set <code>maxMemory</code> , then an error will be returned.
<code>max_time</code>	The maximum walltime or cputime for a single execution of the executable. Walltime or cputime is selected by the GRAM scheduler being interfaced. The units is in minutes. The value will go through an <code>atoi()</code> conversion in order to get an integer.
<code>max_wall_time</code>	Explicitly set the maximum walltime for a single execution of the executable. The units is in minutes. The value will go through an <code>atoi()</code> conversion in order to get an integer. If the GRAM scheduler cannot set walltime, then an error will be returned.
<code>min_memory</code>	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 <code>atoi()</code> 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 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 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 its 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.
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.
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.
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.

```
(* 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:

```
& (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:

```
& (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

- [1] `specification` ::= `relation` /* relation */
 | '+' `spec-list` /* multi-re-quest */
 | '&' `spec-list` /* conjunct-re-quest */
 | '|' `spec-list` /* disjunct-request */
- [2] `spec-list` ::= '(' `specification` ')' `spec-list`
 | '(' `specification` ')'
- [3] `relation` ::= 'rsl_substitution' '=' `binding-sequence` /* Substitution variable definition */
 | `attribute op value-sequence` /* Attribute binding relation */
- [4] `binding-sequence` ::= `binding binding-sequence`
 | `binding`
- [5] `binding` ::= '(' `string-literal simple-value` ')' /* Substitution variable definition */
- [6] `attribute` ::= `string-literal` /* attribute */
- [7] `op` ::= '=' | '!=' | '>' | '>=' | '<' | '<='
- [8] `value-sequence` ::= `value value-sequence` | `value`
- [9] `value` ::= '(' `value-sequence` ')' | `simple-value`
- [10] `simple-value` ::= `string-literal` /* String */
 | `simple-value '#' simple-value` /* Concatenation */
 | `implicit-concat` | `variable-reference`
- [11] `variable-reference` ::= '\$(' `string-literal` ')' /* Variable Reference */
- [12] `implicit-concat` ::= (`unquoted-literal`)? (`implicit-concat-core`)+
- [13] `implicit-concat-core` ::= `variable-reference` | (`variable-reference`) (`unquoted-literal`)
- [14] `string-literal` ::= `quoted-literal` | `unquoted-literal`
- [15] `quoted-literal` ::= ''' (([`^`]) | (""))* ''' /* Single-quote delimiter with escaping */
 | "" (([`^`"]) | (""))* "" /* Double-quote delimiter with escaping */
 | '^' c(([`^c`] | (cc))* c) /* User defined delimiter c with escaping */
- [16] `unquoted-literal` ::= ([`^t\v\n+&|()=<>!"'^#$`])+ /* Non-special characters */
- [17] `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:

```
% 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:

```
% $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 2, “Errors”](#).

For information about sys admin logging, see [Admin Debugging](#) in the GRAM5 Admin Guide.

1. Troubleshooting tips

In case you run into problems you can do the following

- Check the GRAM5 documentation. Maybe you'll find hints here to solve your problem.
- Check the GRAM5 log for errors.

In case you don't find anything suspicious you can increase the log-level of GRAM5 or other relevant components. Maybe the additional logging-information will tell you what's going wrong.

- Send e-mails to <gram-user@globus.org>. You'll have to subscribe to a list before you can send an e-mail to it. See [here](#)¹ for general e-mail lists and information on how to subscribe to a list and [here](#)² for GRAM specific lists.

¹ http://dev.globus.org/wiki/Mailing_Lists

² http://dev.globus.org/wiki/GRAM#Mailing_Lists

2. Errors

Table 9.1. GRAM5 Errors

Error Code	Reason	Possible Solutions
1	one of the RSL parameters is not supported	Check RSL documentation
2	the RSL length is greater than the maximum allowed	Use RSL substitutions to reduce length of RSL strings
3	an I/O operation failed	Enable trace logging and report to gram-dev@globus.org
4	jobmanager unable to set default to the directory requested	Check that RSL <code>directory</code> attribute refers to a directory that exists on the target system.
5	the executable does not exist	Check that the RSL <code>executable</code> attribute refers to an executable that exists on the target system.
6	of an unused INSUFFICIENT_FUNDS	Unimplemented feature.
7	authentication with the remote server failed	Check that the contact string contains the proper X.509 DN.
8	the user cancelled the job	Don't cancel jobs you want to complete.
9	the system cancelled the job	Check RSL requirements such as maximum time and memory are valid for the job.
10	data transfer to the server failed	Check gatekeeper and/or job manager logs to see why the process failed.
11	the stdin file does not exist	Check that the RSL <code>stdin</code> attribute refers to a file that exists on the target system or has a valid ftp, gsiftp, http, or https URL.
12	the connection to the server failed (check host and port)	Check that the service is running on the expected TCP/IP port. Check that no firewall prevents contacting that TCP/IP port. Check <code>\$GLOBUS_LOCATION/var/globus-gatekeeper.log</code> for runtime configuration errors.
13	the provided RSL 'maxtime' value is not an integer	Check that the RSL <code>maxtime</code> value evaluates to an integer.
14	the provided RSL 'count' value is not an integer	Check that the RSL <code>count</code> value evaluates to an integer.
15	the job manager received an invalid RSL	Check that the RSL string can be parsed by using <code>globusrun -p RSL</code> .
16	the job manager failed in allowing others to make contact	Check job manager log.
17	the job failed when the job manager attempted to run it	Verify that the LRM is configured properly.
18	an invalid paradyn was specified	OBSOLETE IN GRAM2
19	the provided RSL 'jobtype' value is invalid	The RSL <code>jobtype</code> attribute is not indicated as supported by the LRM. Valid <code>jobtype</code> values are <code>single</code> , <code>multiple</code> , <code>mpi</code> , and <code>condor</code> .
20	the provided RSL 'myjob' value is invalid	OBSOLETE IN GRAM5

Error Code	Reason	Possible Solutions
21	the job manager failed to locate an internal script argument file	Check that <code>\$GLOBUS_LOCATION/libexec/globus-job-manager-script.pl</code> exists and is executable. Check that the LRM-specific perl module is located in <code>\$GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/</code> 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.
22	the job manager failed to create an internal script argument file	Check that your home directory is writable and not full.
23	the job manager detected an invalid job state	Check job manager logs.
24	the job manager detected an invalid script response	Check job manager logs. This is likely a bug in the LRM script.
25	the job manager detected an invalid script status	Check job manager logs. This is likely a bug in the LRM script.
26	the provided RSL 'jobtype' value is not supported by this job manager	Check that the RSL <code>jobtype</code> attribute is implemented by the LRM script. Note that some job types require configuration
27	unused ERROR_UNIMPLEMENTED	LRM does not support some feature included in the job request.
28	the job manager failed to create an internal script submission file	Check that the user's home file system is not full. Check job manager log
29	the job manager cannot find the user proxy	Check that client is delegating a proxy when authenticating with the gatekeeper. Check that the user's home filesystem and the <code>/tmp</code> file system are not full.
30	the job manager failed to open the user proxy	Check that the user's home filesystem and the <code>/tmp</code> file system are not full.
31	the job manager failed to cancel the job as requested	Check that the user's home filesystem and the <code>/tmp</code> file system are not full.
32	system memory allocation failed	Check job manager log for details.
33	the interprocess job communication initialization failed	OBSOLETE IN GRAM5
34	the interprocess job communication setup failed	OBSOLETE IN GRAM5
35	the provided RSL 'host count' value is invalid	Check that the RSL <code>host_count</code> attribute evaluates to an integer.
36	one of the provided RSL parameters is unsupported	Check job manager log for details about invalid parameter.
37	the provided RSL 'queue' parameter is invalid	Check that the RSL <code>queue</code> attribute evaluates to a string that corresponds to an LRM-specific queue name.
38	the provided RSL 'project' parameter is invalid	Check that the RSL <code>project</code> attribute evaluates to a string that corresponds to an LRM-specific project name.

Error Code	Reason	Possible Solutions
39	the provided RSL string includes variables that could not be identified	Check that all RSL substitutions are defined before being used in the job description.
40	the provided RSL 'environment' parameter is invalid	Check that the RSL <code>environment</code> attribute contains a sequence of <code>VARIABLE VALUE</code> pairs.
41	the provided RSL 'dryrun' parameter is invalid	Remove the RSL <code>dryrun</code> attribute from the job description.
42	the provided RSL is invalid (an empty string)	Include a non-empty RSL string in your job submission request.
43	the job manager failed to stage the executable	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.
44	the job manager failed to stage the stdin file	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.
45	the requested job manager type is invalid	OBSOLETE IN GRAM5
46	the provided RSL 'arguments' parameter is invalid	OBSOLETE IN GRAM2
47	the gatekeeper failed to run the job manager	Check the gatekeeper or job manager logs for more information.
48	the provided RSL could not be properly parsed	Check that the RSL string can be parsed by using globusrun -p RSL .
49	there is a version mismatch between GRAM components	Ask system administrator to upgrade GRAM service to GRAM2 or GRAM5
50	the provided RSL 'arguments' parameter is invalid	Check that the RSL <code>arguments</code> attribute evaluates to a sequence of strings.
51	the provided RSL 'count' parameter is invalid	Check that the RSL <code>count</code> attribute evaluates to a positive integer value.
52	the provided RSL 'directory' parameter is invalid	Check that the RSL <code>directory</code> attribute evaluates to a string.
53	the provided RSL 'dryrun' parameter is invalid	Check that the RSL <code>dryrun</code> attribute evaluates to either <code>yes</code> or <code>no</code> .
54	the provided RSL 'environment' parameter is invalid	Check that the RSL <code>environment</code> attribute evaluates to a sequence of <code>VARIABLE, VALUE</code> pairs.
55	the provided RSL 'executable' parameter is invalid	Check that the RSL <code>executable</code> attribute evaluates to a string value.
56	the provided RSL 'host_count' parameter is invalid	Check that the RSL <code>host_count</code> attribute evaluates to a positive integer value.
57	the provided RSL 'jobtype' parameter is invalid	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>

Error Code	Reason	Possible Solutions
58	the provided RSL 'maxtime' parameter is invalid	Check that the RSL <code>maxtime</code> attribute evaluates to a positive integer value.
59	the provided RSL 'myjob' parameter is invalid	OBSOLETE IN GRAM5.
60	the provided RSL 'paradyn' parameter is invalid	OBSOLETE IN GRAM2.
61	the provided RSL 'project' parameter is invalid	Check that the RSL <code>project</code> attribute evaluates to a string value.
62	the provided RSL 'queue' parameter is invalid	Check that the RSL <code>queue</code> attribute evaluates to a string value.
63	the provided RSL 'stderr' parameter is invalid	Check that the RSL <code>stderr</code> attribute evaluates to a string value or a sequence of <i>DESTINATION</i> URLs with optional <i>CACHE_TAG</i> string parameters.
64	the provided RSL 'stdin' parameter is invalid	Check that the RSL <code>stdin</code> attribute evaluates to a string value.
65	the provided RSL 'stdout' parameter is invalid	Check that the RSL <code>stdout</code> attribute evaluates to a string value or a sequence of <i>DESTINATION</i> URLs with optional <i>CACHE_TAG</i> string parameters.
66	the job manager failed to locate an internal script	Check job manager log for more details.
67	the job manager failed on the system call <code>pipe()</code>	OBSOLETE IN GRAM5
68	the job manager failed on the system call <code>fcntl()</code>	OBSOLETE IN GRAM2
69	the job manager failed to create the temporary stdout filename	OBSOLETE IN GRAM5
70	the job manager failed to create the temporary stderr filename	OBSOLETE IN GRAM5
71	the job manager failed on the system call <code>fork()</code>	OBSOLETE IN GRAM2
72	the executable file permissions do not allow execution	Check that the RSL <code>executable</code> attribute refers to an executable program or script.
73	the job manager failed to open stdout	Check that the RSL <code>stdout</code> attribute refers to one or more valid destination files or URLs.
74	the job manager failed to open stderr	Check that the RSL <code>stderr</code> attribute refers to one or more valid destination files or URLs.
75	the cache file could not be opened in order to relocate the user proxy	Check that the user's home directory is writable and not full on the GRAM5 service node.
76	cannot access cache files in <code>~/globus/.gass_cache</code> , check permissions, quota, and disk space	Check that the user's home directory is writable and not full on the GRAM5 service node.
77	the job manager failed to insert the contact in the client contact list	Check job manager log

Error Code	Reason	Possible Solutions
78	the contact was not found in the job manager's client contact list	Don't attempt to unregister callback contacts that are not registered
79	connecting to the job manager failed. Possible reasons: job terminated, invalid job contact, network problems, ...	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.
80	the syntax of the job contact is invalid	Check the syntax of job contact string.
81	the executable parameter in the RSL is undefined	Include the RSL <code>executable</code> in all job requests.
82	the job manager service is misconfigured. <code>condor arch</code> undefined	Add the <code>-condor-arch</code> to the command-line or configuration file for a job manager configured to use the <code>condor</code> LRM.
83	the job manager service is misconfigured. <code>condor os</code> undefined	Add the <code>-condor-os</code> to the command-line or configuration file for a job manager configured to use the <code>condor</code> LRM.
84	the provided RSL 'min_memory' parameter is invalid	Check that the RSL <code>min_memory</code> attribute evaluates to a positive integer value.
85	the provided RSL 'max_memory' parameter is invalid	Check that the RSL <code>max_memory</code> attribute evaluates to a positive integer value.
86	the RSL 'min_memory' value is not zero or greater	Check that the RSL <code>min_memory</code> attribute evaluates to a positive integer value.
87	the RSL 'max_memory' value is not zero or greater	Check that the RSL <code>max_memory</code> attribute evaluates to a positive integer value.
88	the creation of a HTTP message failed	Check job manager log.
89	parsing incoming HTTP message failed	Check job manager log.
90	the packing of information into a HTTP message failed	Check job manager log.
91	an incoming HTTP message did not contain the expected information	Check job manager log.
92	the job manager does not support the service that the client requested	Check that the client is talking to the correct service
93	the gatekeeper failed to find the requested service	OBSOLETE IN GRAM2
94	the jobmanager does not accept any new requests (shutting down)	Execute queries before the job has been cleaned up.
95	the client failed to close the listener associated with the callback URL	Call <code>globus_gram_client_callback_disallow()</code> with a valid the callback contact.
96	the gatekeeper contact cannot be parsed	Check the syntax of the gatekeeper contact string you are attempting to contact.
97	the job manager could not find the 'poe' command	OBSOLETE IN GRAM2
98	the job manager could not find the 'mpirun' command	Configure the LRM script with <code>mpirun</code> in your path.

Error Code	Reason	Possible Solutions
99	the provided RSL 'start_time' parameter is invalid	OBSOLETE IN GRAM2
100	the provided RSL 'reservation_handle' parameter is invalid	OBSOLETE IN GRAM2
101	the provided RSL 'max_wall_time' parameter is invalid	Check that the RSL <code>max_wall_time</code> attribute evaluates to a positive integer.
102	the RSL 'max_wall_time' value is not zero or greater	Check that the RSL <code>max_wall_time</code> attribute evaluates to a positive integer.
103	the provided RSL 'max_cpu_time' parameter is invalid	Check that the RSL <code>max_cpu_time</code> attribute evaluates to a positive integer.
104	the RSL 'max_cpu_time' value is not zero or greater	Check that the RSL <code>max_cpu_time</code> attribute evaluates to a positive integer.
105	the job manager is misconfigured, a scheduler script is missing	Check that the administrator has configured the LRM by running its setup script.
106	the job manager is misconfigured, a scheduler script has invalid permissions	Check that the administrator has installed the <code>GLLOBUS_LOCATION/libexec/globus-job-manager-script.pl</code> script. Check that the file system containing that script allows file execution.
107	the job manager failed to signal the job	OBSOLETE IN GRAM2
108	the job manager did not recognize/support the signal type	Check that your signal operation is using the correct signal constant.
109	the job manager failed to get the job id from the local scheduler	OBSOLETE IN GRAM2
110	the job manager is waiting for a commit signal	Send a two-phase commit signal to the job manager to acknowledge receiving the job contact from the job manager.
111	the job manager timed out while waiting for a commit signal	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.
112	the provided RSL 'save_state' parameter is invalid	Check that the RSL <code>save_state</code> attribute is set to <code>yes</code> or <code>no</code> .
113	the provided RSL 'restart' parameter is invalid	Check that the RSL <code>restart</code> attribute evaluates to a string containing a job contact string.
114	the provided RSL 'two_phase' parameter is invalid	Check that the RSL <code>two_phase</code> attribute evaluates to a positive integer.
115	the RSL 'two_phase' value is not zero or greater	Check that the RSL <code>two_phase</code> attribute evaluates to a positive integer.
116	the provided RSL 'stdout_position' parameter is invalid	OBSOLETE IN GRAM5
117	the RSL 'stdout_position' value is not zero or greater	OBSOLETE IN GRAM5

Error Code	Reason	Possible Solutions
118	the provided RSL 'stderr_position' parameter is invalid	OBSOLETE IN GRAM5
119	the RSL 'stderr_position' value is not zero or greater	OBSOLETE IN GRAM5
120	the job manager restart attempt failed	OBSOLETE IN GRAM2
121	the job state file doesn't exist	Check that the job contact you are trying to restart matches one that the job manager returned to you.
122	could not read the job state file	Check that the state file directory is not full.
123	could not write the job state file	Check that the state file directory is not full.
124	old job manager is still alive	Contact the returned job manager contact to manage the job you are trying to restart.
125	job manager state file TTL expired	OBSOLETE in GRAM2
126	it is unknown if the job was submitted	Check job manager log.
127	the provided RSL 'remote_io_url' parameter is invalid	Check that the RSL <code>remote_io_url</code> attribute evaluates to a string value.
128	could not write the remote io url file	Check that the user's home file system on the job manager service node is writable and not full.
129	the standard output/error size is different	Send a stdio update signal to redirect the job manager output to a new URL
130	the job manager was sent a stop signal (job is still running)	Submit a restart request to monitor the job.
131	the user proxy expired (job is still running)	Generate a new proxy and then submit a restart request to monitor the job.
132	the job was not submitted by original job-manager	OBSOLETE IN GRAM2
133	the job manager is not waiting for that commit signal	Do not send a commit signal to a job that is not waiting for a commit signal.
134	the provided RSL scheduler specific parameter is invalid	Check the LRM-specific documentation to determine what values are legal for the RSL extensions implemented by the LRM.
135	the job manager could not stage in a file	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.
136	the scratch directory could not be created	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.
137	the provided 'gass_cache' parameter is invalid	Check that the RSL <code>gass_cache</code> attribute evaluates to a string.
138	the RSL contains attributes which are not valid for job submission	Do not use restart- or signal-only RSL attributes when submitting a job.

Error Code	Reason	Possible Solutions
139	the RSL contains attributes which are not valid for stdio update	Do not use submit- or restart-only RSL attributes when sending a stdio update signal to a job.
140	the RSL contains attributes which are not valid for job restart	Do not use submit- or signal-only RSL attributes when restarting a job.
141	the provided RSL 'file_stage_in' parameter is invalid	Check that the RSL <code>file_stage_in</code> attribute evaluates to a sequence of <i>SOURCE DESTINATION</i> pairs.
142	the provided RSL 'file_stage_in_shared' parameter is invalid	Check that the RSL <code>file_stage_in_shared</code> attribute evaluates to a sequence of <i>SOURCE DESTINATION</i> pairs.
143	the provided RSL 'file_stage_out' parameter is invalid	Check that the RSL <code>file_stage_out</code> attribute evaluates to a sequence of <i>SOURCE DESTINATION</i> pairs.
144	the provided RSL 'gass_cache' parameter is invalid	Check that the RSL <code>gass_cache</code> attribute evaluates to a string.
145	the provided RSL 'file_cleanup' parameter is invalid	Check that the RSL <code>file_clean_up</code> attribute evaluates to a sequence of strings.
146	the provided RSL 'scratch_dir' parameter is invalid	Check that the RSL <code>scratch_dir</code> attribute evaluates to a string.
147	the provided scheduler-specific RSL parameter is invalid	Check the LRM-specific documentation to determine what values are legal for the RSL extensions implemented by the LRM.
148	a required RSL attribute was not defined in the RSL spec	Check that the RSL <code>executable</code> attribute is present in your job request RSL. Check that the RSL <code>restart</code> attributes is present in your restart RSL.
149	the <code>gass_cache</code> attribute points to an invalid cache directory	Check that the RSL <code>gass_cache</code> attributes evaluates to a directory that exists or can be created. Check that the user's home file system is writable and not full.
150	the provided RSL 'save_state' parameter has an invalid value	Check that the RSL <code>save_state</code> attribute has a value of <code>yes</code> or <code>no</code> .
151	the job manager could not open the RSL attribute validation file	Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf</code> is present and readable on the job manager service node. Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf</code> is readable on the job manager service node if present.
152	the job manager could not read the RSL attribute validation file	Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/globus-gram-job-manager.rvf</code> is valid. Check that <code>\$GLOBUS_LOCATION/share/globus_gram_job_manager/LRM.rvf</code> is valid if present.
153	the provided RSL 'proxy_timeout' is invalid	Check that RSL <code>proxy_timeout</code> attribute evaluates to a positive integer.
154	the RSL 'proxy_timeout' value is not greater than zero	Check that RSL <code>proxy_timeout</code> attribute evaluates to a positive integer.

Error Code	Reason	Possible Solutions
155	the job manager could not stage out a file	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.
156	the job contact string does not match any which the job manager is handling	Check that the job contact string matches one returned from a job request.
157	proxy delegation failed	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.
158	the job manager could not lock the state lock file	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.
159	an invalid globus_io_clientattr_t was used.	Check that you have initialized the globus_io_clientattr_t attribute prior to using it with the GRAM client API.
160	an null parameter was passed to the gram library	Check that you are passing legal values to all GRAM API calls.
161	the job manager is still streaming output	OBSOLETE IN GRAM5
162	the authorization system denied the request	Check with your GRAM system administrator to allow a particular certificate to be authorized.
163	the authorization system reported a failure	Check with your system administrator to verify that the authorization system is configured properly.
164	the authorization system denied the request - invalid job id	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.
165	the authorization system denied the request - not authorized to run the specified executable	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.
166	the provided RSL 'user_name' parameter is invalid.	Check that the RSL user_name attribute evaluates to a string.
167	the job is not running in the account named by the 'user_name' parameter.	Ask with the GRAM system administrator to add an authorization entry to allow your credential to run jobs as the specified user account.

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

```
rs1: &( executable = "/bin/echo" )
    ( arguments = "hello" )
```

becomes

```
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:

<code>exit-code</code>	Job exit code. Sent in job state callbacks and in job status replies when the job completes.
<code>gt3-failure-type</code>	Failure detail type for staging errors. Sent in job state callbacks and in job status replies when a job fails.
<code>gt3-failure-message</code>	Failure detail message for more context for errors. Sent in job state callbacks and in job status replies when a job fails.
<code>gt3-failure-source</code>	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.
<code>gt3-failure-destination</code>	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.
<code>version</code>	Job manager package version. Sent in all messages from the job manager.
<code>toolkit-version</code>	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 % HEX HEX 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

<code>job-manager-name</code>	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".
<code>host-name</code>	The name of the host on which the gatekeeper is running. This exists only for compatibility with the HTTP/1.1 protocol.
<code>message-size</code>	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:

```
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 username 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:

```
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:

<i>job-contact</i>	The job contact string returned in a response to a job request message, or determined by querying the MDS system.
<i>host-name</i>	The name of the host on which the job manager is running. This exists only for compatibility with the HTTP/1.1 protocol.
<i>message-size</i>	The length of the content of the message, not including the HTTP/1.1 header.
<i>version</i>	The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string "2".
<i>callback-contact</i>	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:

<i>job-contact</i>	The job contact string returned in a response to a job request message, or determined by querying the MDS system.
<i>host-name</i>	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.

<i>host-name</i>	The host part of the callback-contact URL. This exists only for compatibility with the HTTP/1.1 protocol.
<i>message-size</i>	The length of the content of the message, not including the HTTP/1.1 header.
<i>version</i>	The version of the GRAM protocol which is being used. For the protocol defined in this document, the value must be the string 2.
<i>job-contact</i>	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:

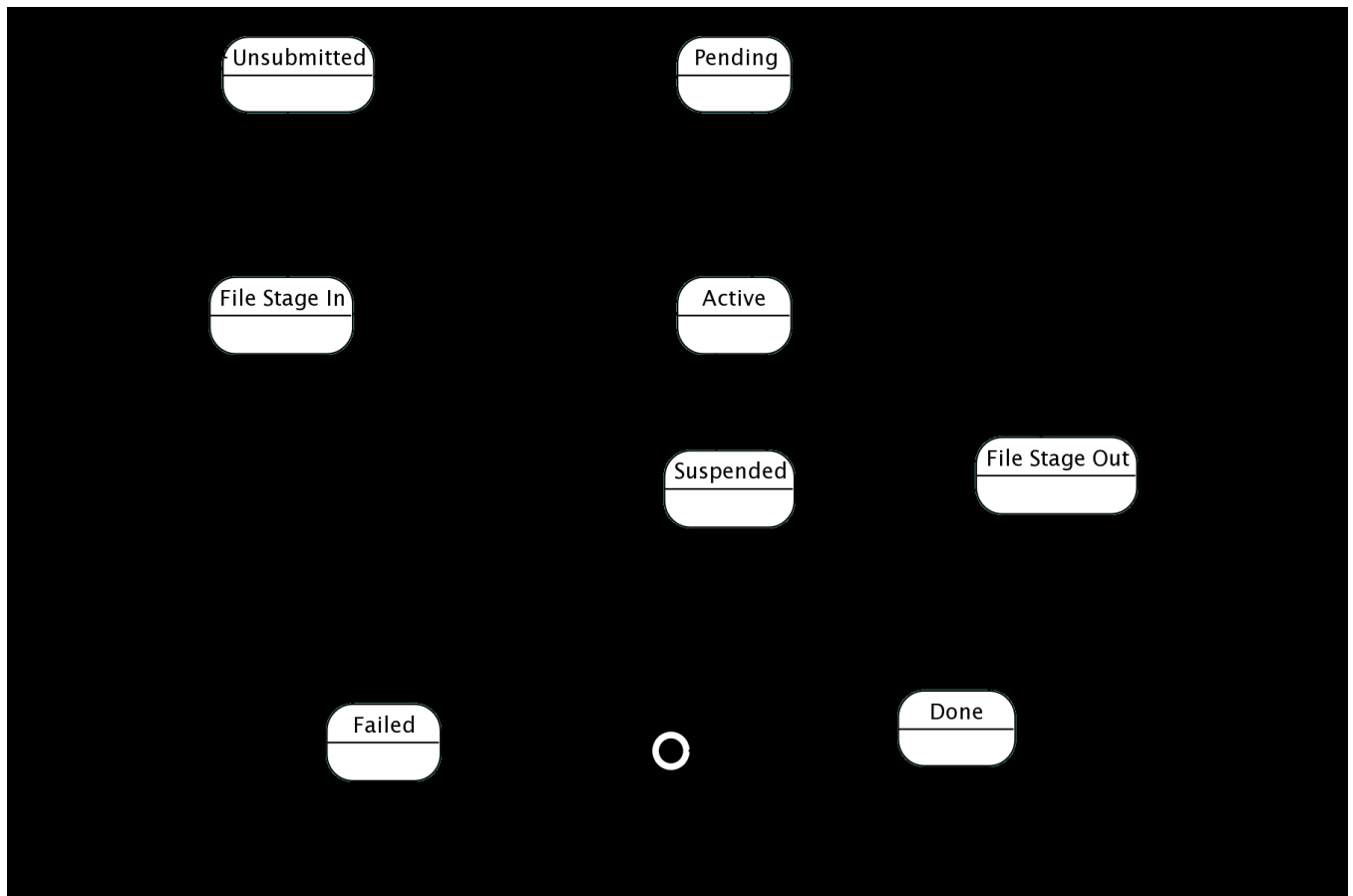
- Authentication must be done using GSSAPI mutual authentication
- Messages must be wrapped with support for the delegation message. When using Globus I/O, this is accomplished by using the the GLOBUS_IO_SECURE_CHANNEL_MODE_GSI_WRAP wrapping mode.

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

State	Meaning
GLOBUS_GRAM_PROTOCOL_JOB_STATE_UNSUBMITTED	Initial job state
GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_IN	Job staging in progress
GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING	Job submitted to LRM, awaiting execution
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE	Job executing
GLOBUS_GRAM_PROTOCOL_JOB_STATE_SUSPENDED	Job made progress executing but is now suspended
GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_OUT	Job staging in progress after job completed
GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE	Job completed successfully
GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED	Job was canceled or failed

Figure 10.1. GRAM State Transitions

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

Index

A

apis, 29
 overview, 29

C

compatibility, 2

D

debugging, 38
dependencies, 2

E

errors, 41

F

features, 1

P

platforms, tested, 1

T

troubleshooting, 40
 check documentation, 40
 errors, 40
 gram log, 40
 mailing lists, 40

GT 5.0.0 Migrating Guide for GRAM5

GT 5.0.0 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 GT4.2	1
1. Admin - Migration Guide	1
2. User - Migration Guide	1
3. Developer - Migration Guide	1
2. Migrating GRAM from GT4.0	2
1. Admin - Migration Guide	2
2. User - Migration Guide	2
3. Developer - Migration Guide	2
3. Migrating GRAM from GT3	3
4. Migrating GRAM from GT2	5
1. Admin - Migration Guide	5
2. User - Migration Guide	6
3. Developer - API and RSL Migration Guide	7
Glossary	8

Chapter 1. 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 2. 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.

Chapter 3. Migrating GRAM from GT3

Migrating to GT 5.0.0 from GT version 3.2:

- The 5.0.0 protocol has been changed to be WSRF compliant. There is no backward compatibility between 5.0.0 and 3.2.

API changes since GT 3.2:

- The *MJFS* `create` operation has become `createManagedJob` and, now provides the option to send a *uuid*. A client can use this *uuid* to recover a job EPR in the event that the reply message is not received. Given this new scheme, the `start` operation was removed. The `createManagedJob()` operation also allows a notification subscription request to be specified. This is the only way to reliably get all job state notifications.
- The *MJS* `start` operation has been removed. Its purpose was to ensure that the client had received the job EPR prior to the job being executed (and thus consuming resources), and is redundant with the *uuid* functionality.

New GRAM Client Submission Tool:

- *globusrun-ws* has replaced `managed-job-globusrun` as the GRAM5 client submission program. The main reason was performance. The cost of JVM startup for each job submission through **managed-job-globusrun** was too much. **globusrun-ws** is written in C and thus avoids the JVM startup cost. **globusrun-ws** is very similar in functionality to **managed-job-globusrun**, but you will need to become familiar with the arguments and options.

RSL Schema Changes Since GT 3.2:

- RSL Substitutions RSL substitution syntax has changed to allow for a simpler RSL schema that can be parsed by standard tools. In GT 3.2, applications could define arbitrary RSL substitutions within an RSL document and rely on the GRAM service to resolve them. In GT5 GRAM5, this feature is no longer present. In GT 5.0.0 there are 5 RSL variables: `${GLOBUS_USER_HOME}`, `${GLOBUS_USER_NAME}`, `${GLOBUS_SCRATCH_DIR}`, and `${GLOBUS_LOCATION}`.
- `executable` is now a single local file path. Remote URLs are no longer allowed. If executable staging is desired, it should be added to the `fileStageIn` directive.
- `stdin` is now a single local file path. Remote URLs are no longer allowed. If `stdin` staging is desired, it should be added to the `fileStageIn` directive.
- `stdout` is now a single local file path, instead of a list of remote URLs. If `stdout` staging is desired, it should be added to the `fileStageOut` directive.
- `stderr` is now a single local file path, instead of a list of remote URLs. If `stderr` staging is desired, it should be added to the `fileStageOut` directive.
- `scratchDirectory` has been removed.
- `gramMyJobType` has been removed. "Collective" functionality is always available if a job chooses to use it.
- `dryRun` has been removed. This is obsolete given the addition of the `holdState` attribute. setting `holdState` to "StageIn" should prevent the job from being submitted to the local *scheduler*. It can then be canceled once the StageIn-Hold state notification is received.
- `remoteIoUrl` has been removed. This was a hack for GRAM2 involved with staging via GASS, and has no relevancy in the current implementation.

- File Staging related RSL attributes have been replaced with RFT file transfer attributes/syntax.
- RSL substitution definitions and substitution references have been removed in order to be able to use standard XML parsing/serialization tools.
- RSL variables have been added. These are keywords denoted in the form of `${variable name}` that can be found in certain RSL attributes.
- Explicit credential references have been added, which, along with use of the new `DelegationFactory` service, replace the old implicit delegation model.

Fault changes since GT version 3.2:

- `CacheFaultType` was removed since there is no longer a GASS cache.
- `RepeatedlyStartedFaultType` was removed since there is no longer a `start` operation. Repeat creates with the same submission ID simply return the job EPR.
- `SLAFaultType` was changed to `ServiceLevelAgreementFaultType` for clarification.
- `StreamServiceCreationFaultType` was removed since there is no longer a stream service.
- `UnresolvedSubstitutionReferencesFaultType` was removed since there is no longer support for substitution definitions and references in the RSL.
- `DatabaseAccessFaultType` was removed since a database is no longer used to save job data.

Chapter 4. Migrating GRAM from GT2

1. Admin - Migration Guide

1.1. Installation / Deployment Differences

In GRAM2, jobs are submitted to a job manager process started by a Gatekeeper process. In GRAM5, the job submit protocol is the same; however, all jobs for a particular user and LRM run in the same job manager process.

1.2. Security Differences

1.2.1. Proxies and Delegation

In GRAM2, the GRAM client is required to delegate a proxy credential to the Gatekeeper so that the job manager can send authenticated job state change messages. Because each job is monitored by a separate job manager in GRAM2, each job manager has access to a different delegated credential. In GRAM5, the shared job manager uses the delegated credential with the latest expiration time as its credential.

In GRAM2, the client can control the job manager proxy timeout by setting the value of `proxy_timeout` RSL attribute to a time interval in seconds indicating when the job manager will exit if the proxy is about to expire. In GRAM5, this is a job manager-wide setting and the `proxy_timeout` RSL attribute is ignored.

1.3. Network Communication

In GRAM2, the standard output and standard error streams of a job may be sent in near real time to a file server such as the GASS server embedded in a **globusrun** execution. In GRAM5 the same RSL syntax is used to name output and error stream destinations, but those are not sent until after the job execution is complete.

In GRAM2, the job manager implements intra-job communication via DUCT and task synchronization via DUROC. These features have been dropped in GRAM5.

GRAM5 adds various protocol extensions to the GRAM2 protocol. This is done in a way such that the existing GRAM2 protocol processors will ignore the extensions to the messages. Details about the protocol and its extensions can be found in the [GRAM5 public interface document](#).

1.4. LRM Interaction Differences

In GRAM2, all file system and LRM interactions occur within a perl module called by the `globus-job-manager-script.pl` program. Scheduler-specific perl modules implement a number of methods which are used by the job manager:

- `submit`
- `poll`
- `cancel`
- `signal`
- `make_scratchdir`
- `remove_scratchdir`

- stage_in
- stage_out
- cache_cleanup
- remote_io_file_create
- proxy_relocate
- proxy_update

Only a small set of these script methods are used in the GRAM5 implementation. The subset used is:

- submit
- signal (called when not using SEG)
- poll (called when not using SEG)
- cancel

Some of the functionality has been moved into the job manager or other services for performance reasons.

1.5. Local Node Impact

In GRAM2, each job submitted would cause the following processes to be created:

- gatekeeper (short lived)
- job manager (lives the duration of the job)
- perl script (short lived 4 or more instances depending on job type)
- perl script poll called periodically

In GRAM5, each job causes the following processes to be created

- **globus-gatekeeper** (short lived)
- **globus-job-manager** (short lived for all but one concurrent instance)
- **globus-job-manager-script** (up to 5 per lifetime of the job manager)
- **globus-fork-starter** (up to 5 per job manager when using the fork LRM and the SEG).

Additionally, there will be a per-scheduler instance of the *SEG*-related program, **globus-job-manager-event-generator**.

2. User - Migration Guide

2.1. Command Line Tools

The **globusrun** tool in GRAM2 supports DUROC and MPICH-G jobs. This feature has been removed in GRAM5, as well as the command-line options related to it.

3. Developer - API and RSL Migration Guide

The DUROC and DUCT APIs have been removed in GRAM5.

Glossary

R

Resource Specification Language (RSL)

Term used to describe a GRAM job for GT2 and GT3. (Note: This is not the same as RLS - the Replica Location Service)

S

scheduler

Term used to describe a job scheduler mechanism to which GRAM interfaces. It is a networked system for submitting, controlling, and monitoring the workload of batch jobs in one or more computers. The jobs or tasks are scheduled for execution at a time chosen by the subsystem according to an available policy and availability of resources. Popular job schedulers include Portable Batch System (PBS), Platform LSF, and IBM LoadLeveler.

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.

U

Universally Unique Identifier (UUID)

Identifier that is immutable and unique across time and space.

GT 5.0.0 GRAM5: Quality Profile

Table of Contents

1. Test coverage reports	1
2. Code analysis reports	1
3. Outstanding bugs	1
4. Bug Fixes	3
5. GRAM5 Throughput Tests	3
6. GRAM5 Condor-G Tests	5

<titleabbrev>Quality Profile</titleabbrev>

1. Test coverage reports

- [GT 5.0.0 Test Coverage Status Report](#)¹

2. Code analysis reports

- No code analysis reports have been generated at this time.

3. Outstanding bugs

- [Bug 108](#):² Fork perl zombies
- [Bug 106](#):³ Fix test failures with SGE LRM adapter
- [Bug 105](#):⁴ Held Condor jobs should be reported as SUSPENDED
- [Bug 104](#):⁵ globus-job-manager-event-generator loads all historical events the first time run
- [Bug 103](#):⁶ Ease two phase end commit timeout
- [Bug 102](#):⁷ Fix Two Phase Commit Semantics for Failed Jobs
- [Bug 100](#):⁸ one of the RSL parameters is not supported error doesn't indicate which it is
- [Bug 99](#):⁹ Add a high-level diagram for the approach doc
- [Bug 98](#):¹⁰ Add Condor-G doc for using GRAM 2 and 5

¹ ../reports/coverage/index.html

² <http://jira.globus.org/browse/GRAM-108>

³ <http://jira.globus.org/browse/GRAM-106>

⁴ <http://jira.globus.org/browse/GRAM-105>

⁵ <http://jira.globus.org/browse/GRAM-104>

⁶ <http://jira.globus.org/browse/GRAM-103>

⁷ <http://jira.globus.org/browse/GRAM-102>

⁸ <http://jira.globus.org/browse/GRAM-100>

⁹ <http://jira.globus.org/browse/GRAM-99>

¹⁰ <http://jira.globus.org/browse/GRAM-98>

- [Bug 96](#):¹¹ GRAM-106 SGE LRM mishandles invalid environment definition
- [Bug 95](#):¹² GRAM-106 SGE LRM doesn't check for executable permissions
- [Bug 94](#):¹³ GRAM-106 SGE LRM doesn't check for executable existence
- [Bug 93](#):¹⁴ GRAM-106 SGE LRM script doesn't handle environment vars with whitespace
- [Bug 92](#):¹⁵ stdout to local file doesn't work if count >1
- [Bug 88](#):¹⁶ Missed two phase commit causes job to not be destroyed
- [Bug 86](#):¹⁷ Prioritize script invocations to improve throughput
- [Bug 80](#):¹⁸ GRAM 5 beta2 release
- [Bug 79](#):¹⁹ Add support for OSG's "NFS Lite" concept
- [Bug 77](#):²⁰ GRAM zombie
- [Bug 71](#):²¹ GRAM protocol test package contains expired test certificate
- [Bug 70](#):²² globus-job-status acts strange for completed jobs in GRAM5
- [Bug 69](#):²³ globus-job-get-output -f doesn't work in GRAM5
- [Bug 68](#):²⁴ Bad error when proxy is too short-lived
- [Bug 54](#):²⁵ make globus-job-manager-event-generator not require configuration by default
- [Bug 53](#):²⁶ Generalize log path configuration
- [Bug 51](#):²⁷ configurable control of number of perl scripts that can run simultaneously
- [Bug 47](#):²⁸ simplify the throughput tester program and use improved version as doc
- [Bug 24](#):²⁹ Debug/verbose flags for globusrun, globus-job-run
- [Bug 23](#):³⁰ Improved error codes and error reporting for users

¹¹ <http://jira.globus.org/browse/GRAM-96>

¹² <http://jira.globus.org/browse/GRAM-95>

¹³ <http://jira.globus.org/browse/GRAM-94>

¹⁴ <http://jira.globus.org/browse/GRAM-93>

¹⁵ <http://jira.globus.org/browse/GRAM-92>

¹⁶ <http://jira.globus.org/browse/GRAM-88>

¹⁷ <http://jira.globus.org/browse/GRAM-86>

¹⁸ <http://jira.globus.org/browse/GRAM-80>

¹⁹ <http://jira.globus.org/browse/GRAM-79>

²⁰ <http://jira.globus.org/browse/GRAM-77>

²¹ <http://jira.globus.org/browse/GRAM-71>

²² <http://jira.globus.org/browse/GRAM-70>

²³ <http://jira.globus.org/browse/GRAM-69>

²⁴ <http://jira.globus.org/browse/GRAM-68>

²⁵ <http://jira.globus.org/browse/GRAM-54>

²⁶ <http://jira.globus.org/browse/GRAM-53>

²⁷ <http://jira.globus.org/browse/GRAM-51>

²⁸ <http://jira.globus.org/browse/GRAM-47>

²⁹ <http://jira.globus.org/browse/GRAM-24>

³⁰ <http://jira.globus.org/browse/GRAM-23>

- [Bug 22:](#)³¹ client connections can't be timed out
- [Bug 15:](#)³² transition from httpg to https
- [Bug 14:](#)³³ increase availability of GRAM in linux distributions
- [Bug 12:](#)³⁴ Gatekeeper's syslog output cannot be controlled
- [Bug 5:](#)³⁵ Add gram-level prologue and epilogue script execution for mpi jobs
- [Bug 4:](#)³⁶ Add support for a "managed fork" service
- [Bug 2:](#)³⁷ Investigate how to setup GRAM5 services in a HA setup

4. Bug Fixes

- [All Fixes:](#)³⁸ This link lists all 67 of the improvements and bug fixes that were done for the GT 5.0.0 release

5. GRAM5 Throughput Tests

5.1. Experiment Hardware and Software Configuration

The following experiments were run on the `nomer.mcs.anl.gov` virtual cluster. The cluster consists of 6 partitions, each having a single Intel(R) Xeon(R) CPU E5430 @ 2.66GHz core, and 2GB RAM. The virtual machines in the cluster each had a single virtual network interface. The cluster was configured as follows:

- 1 node: Master node
- 5 nodes: Test/execution nodes

All nodes ran an **apache2** http server, **gmond** (ganglia monitoring), and **pbs_mom** (torque LRM).

The master node also ran a **globus-gatekeeper**, **globus-gridftp-server**, **globus-job-manager-event-generator**, **gmetad** (Ganglia Meta Daemon), **pbs_sched** (Torque LRM scheduler), **pbs_server** (Torque LRM server), and **nfsd** linux kernel NFSv4 server for the execution nodes.

The test nodes (1 for single-client tests, 5 for multiple client tests) ran the **gram-throughput-tester** program. All 5 test/execution nodes executed the test job executables as scheduled by the LRM.

These tests were done with GT 5.0.0 beta1. The throughput tester was compiled from CVS trunk October 30, 2009.

5.2. Experiment Scenarios

The first set of tests submitted jobs to the GRAM5 service for one hour. After one hour elapsed, the client would terminate any jobs which were being managed by the GRAM5 service. The test client recorded the time of the experiment start, the time of the termination start, and the time after which all jobs had reached a DONE or FAILED state.

³¹ <http://jira.globus.org/browse/GRAM-22>

³² <http://jira.globus.org/browse/GRAM-15>

³³ <http://jira.globus.org/browse/GRAM-14>

³⁴ <http://jira.globus.org/browse/GRAM-12>

³⁵ <http://jira.globus.org/browse/GRAM-5>

³⁶ <http://jira.globus.org/browse/GRAM-4>

³⁷ <http://jira.globus.org/browse/GRAM-2>

³⁸ <http://tinyurl.com/yagefou>

The throughput test client would generate a maximum of 50 simultaneous job requests. For all but the *uncapped* test below, a maximum of 2000 jobs were managed by the job manager at any given time (pending or active). Once the client had submitted 2000 jobs, it would stop submissions until a job completed. A total of 10 execution slots were available for the LRM to schedule jobs into, so many were in the GRAM5 PENDING state during the duration of the test.

The different **gram-throughput-tester** experiments consisted of:

Table 1. GRAM5 Throughput Tester Experiments

Experiment Name	LRM monitoring method	Number of clients	Number of users	Maximum number of simultaneous jobs / client
1-client-poll	POLL	1	1	2000
1-client-seg	SEG	1	1	2000
1-client-seg-uncapped	SEG	1	1	unlimited
5-client-seg	SEG	5	1	2000
5-client-seg-diffusers	SEG	5	5	2000

5.3. Throughput Test Results

The following table contains a summary of the results of these experiments. The columns contain the following information:

Experiment	Experiment name, the same as in the previous section
Total Jobs	The total number of GRAM jobs that were <i>submitted</i> to the GRAM5 service by the throughput tester in one hour.
Termination Tasks After 1 Hour	The total number of jobs that were being managed by the GRAM5 service when the one hour test period elapsed. These jobs were terminated using the GRAM5 cancel protocol message by the throughput tester.
Termination Duration (hh:mm:ss)	The amount of time it took for the termination tasks to complete and all jobs to reach the DONE or FAILED state.
Master Node Max 1 min. Load Average	The maximum value of the one-minute load average on the master node, that is, the node running the GRAM5 and Torque service.
Master Node Average 1 min. Load Average	The average value of the one-minute load average on the master node over the duration of the test.
Errors	Description of any errors which occurred on the client side that prevented operations from completing as expected.

Table 2. GRAM5 Throughput Tester Results Summary

Experiment	Total Jobs	Termination Tasks After 1 Hour	Termination Duration (hh:mm:ss)	Master Node Max 1 min. Load Average	Master Node Average 1 min. Load Average	Errors
1-client-poll	2110	2000	00:14:18	11.46	7.96	None
1-client-seg	2110	2000	00:10:36	2.82	0.93	None
1-client-seg-un-capped	6664	6584	00:42:46	3.26	2.75	None
5-client-seg	6800	6434	00:57:20	3.19	2.49	Connection refused during termination
5-client-seg-diffusers	7226	6720	00:45:41	3.79	3.13	None

6. GRAM5 Condor-G Tests

6.1. Experiment Hardware and Software Configuration

The following experiments were run on the `nomer.mcs.anl.gov` virtual cluster. The cluster consists of 6 partitions, each having a single Intel(R) Xeon(R) CPU E5430 @ 2.66GHz core, and 2GB RAM. The virtual machines in the cluster each had a single virtual network interface. The cluster was configured as follows:

- 1 node: Master node
- 1 nodes: Test/execution nodes
- 4 nodes: execution nodes

All nodes ran an **apache2** http server, **gmond** (ganglia monitoring), and **pbs_mom** (torque LRM).

The master node also ran a **globus-gatekeeper**, **globus-gridftp-server**, **globus-job-manager-event-generator**, **gmetad** (Ganglia Meta Daemon), **pbs_sched** (Torque LRM scheduler), **pbs_server** (Torque LRM server), and **nfsd** linux kernel NFSv4 server for the execution nodes.

The test/execution node ran the condor-g daemons. The condor job classified ad included attributes to submit the jobs to the GRAM5 service running on the service node. The tests were done with Condor version 7.4.1. The Condor-G configuration parameters in the `condor_config` file were as follows:

Figure 1. Condor-G Experiment Configuration

```

GRIDMANAGER_MAX_PENDING_SUBMITS_PER_RESOURCE=50
GRIDMANAGER_MAX_SUBMITTED_JOBS_PER_RESOURCE=2000
GRIDMANAGER_MAX_PENDING_REQUESTS=50
GRIDMANAGER_JOB_PROBE_INTERVAL=300
GRIDMANAGER_MAX_JOBMANAGERS_PER_RESOURCE=0
ENABLE_GRID_MONITOR=FALSE
GRIDMANAGER_DEBUG= D_FULLLDEBUG
GRIDMANAGER_GLOBUS_COMMIT_TIMEOUT=12000

```

The execution nodes executed the test job executables as scheduled by the LRM.

For this test, the test/execution node and the execution nodes were configured to run up to 20 job processes each simultaneously.

6.2. Experiment Scenario

This test submitted a 2000 job condor job cluster, using the following classified ad:

Figure 2. Condor-G Classified Ad

```
Universe=grid
grid_resource = gt5 nomer1.mcs.anl.gov:2119/jobmanager-pbs
executable=/bin/sleep
arguments=300
transfer_executable=False
stream_output = False
stream_error = False
output = test.out.$(Process)
error = test.err.$(Process)
log = test.log
notification=Never
queue 2000
```

The configuration parameters are similar to the GRAM5 tests described in GRAM5 Throughput Tests section. The key difference being that this test runs until all 2000 jobs have completed and does not submit any jobs after the maximum of 2000 has been reached.

To provide a point of comparison, another test using similar parameters was run using the **gram-throughput-tester** program in place of Condor-G. Note that the Condor-G service provides file staging and a scratch directory beyond what the throughput tester job did.

The two experiments consist of:

Table 3. GRAM5 Condor-G Experiments

Experiment Name	LRM monitoring method	Number of clients	Number of users	Total number of jobs
Condor-G	SEG	1	1	2000
Throughput Tester	SEG	1	1	2000

6.3. Condor-G Test Results

The following table contains a summary of the results of these experiments. The columns contain the following information:

Experiment	Experiment name, the same as in the previous section
Time to Submit 2000 Jobs	The total number of GRAM jobs that were <i>submitted</i> to the GRAM5 service by the throughput tester in one hour.

Time For All Jobs To Complete	The amount of time it took for all 2000 jobs to complete. The theoretical minimum value for this is 50 minutes if all 2000 jobs were submitted instantaneously and there was no overhead for them to be deployed to the 200 execution nodes.
LRM Submit Rate (Jobs/Minute)	The total number of jobs that were being managed by the GRAM5 service when the one hour test period elapsed. These jobs were terminated using the GRAM5 cancel protocol message by the throughput tester.
Master Node Max 1 min. Load Average	The maximum value of the one-minute load average on the master node, that is, the node running the GRAM5 and Torque service.
Master Node Average 1 min. Load Average	The average value of the one-minute load average on the master node over the duration of the test.

Table 4. GRAM5 Throughput Tester Results Summary

Experiment	Time to Submit 2000 Jobs (hh:mm:ss)	Time For All Jobs To Complete (hh:mm:ss)	LRM Submit Rate (Jobs/Minute)	Master Node Max 1 min. Load Average	Master Node Average 1 min. Load Average
Condor-G	00:32:34	02:00:56	94	6.56	1.64
Throughput Tester	00:17:58	01:54:49	111	2.43	0.53

GT 5.0.0 Release Notes: GRAM5

Table of Contents

1. Component Overview	1
2. Feature summary	1
3. Summary of Changes in GRAM5	2
4. Bug Fixes	2
5. Known Problems	2
6. Technology dependencies	4
7. Tested platforms	4
8. Backward compatibility summary	5
9. Associated Standards	5
10. For More Information	5
Glossary	5

[Release Notes](#)

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 *job scheduler*, 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 4.2.x

- Server-side architectural changes to improve scalability, performance, and reliability
- Improved error notification protocol compared to GRAM2
- Teragrid Gateway Identity support for job auditing
- Usage stats messages
- Added support for Sun Grid Engine (SGE)

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

- The GRAM5 client tools have dropped support for the Duroc API for task coallocation
- The GRAM5 service no longer streams output and error during job execution; instead this data is send after the job terminates
- The GRAM5 service no longer provides intra-job communication via the DUCT API
- The GRAM5 does not rely on XML schemas and WSDL service definitions

3. Summary of Changes in GRAM5

GRAM5 represents a significant improvement from GRAM2 and GRAM4 service implementations. GRAM2's limitation is scalability. GRAM4's is reliability. GRAM5 is both reliable AND scalable. It is important to note that GRAM5 is GRAM2 compatible. There are other improvements as well, like completely rewritten service logging based on the [CEDPS logging best practices](#)¹, Teragrid Gateway Identity support for job auditing, support for job exit codes, and [usage stat support](#)².

We have been very encouraged by our [performance results](#)³, which shows greater than 10x scalability than GRAM2 and roughly 10x reduction in resource consumption on the service host. We welcome your feedback as you integrate GRAM5 into your production grids.

4. Bug Fixes

- [All Fixes](#):⁴ This link lists all 67 of the improvements and bug fixes that were done for the GT 5.0.0 release

5. Known Problems

The following problems and limitations are known to exist for GRAM5 at the time of the 5.0.0 release:

5.1. Limitations

- [list limitations]

5.2. Outstanding bugs

- [Bug 108](#):⁵ Fork perl zombies
- [Bug 106](#):⁶ Fix test failures with SGE LRM adapter
- [Bug 105](#):⁷ Held Condor jobs should be reported as SUSPENDED
- [Bug 104](#):⁸ globus-job-manager-event-generator loads all historical events the first time run

¹ <http://www.cedps.net/index.php/LoggingBestPractices>

² <http://www.globus.org/toolkit/docs/5.0/5.0.0/execution/gram5/user/#gram5-usage-stats>

³ <http://www.globus.org/toolkit/docs/5.0/5.0.0/execution/gram5/qp/#gram5-reports-throughput>

⁴ <http://tinyurl.com/yagefou>

⁵ <http://jira.globus.org/browse/GRAM-108>

⁶ <http://jira.globus.org/browse/GRAM-106>

⁷ <http://jira.globus.org/browse/GRAM-105>

⁸ <http://jira.globus.org/browse/GRAM-104>

- [Bug 103](#):⁹ Ease two phase end commit timeout
- [Bug 102](#):¹⁰ Fix Two Phase Commit Semantics for Failed Jobs
- [Bug 100](#):¹¹ one of the RSL parameters is not supported error doesn't indicate which it is
- [Bug 99](#):¹² Add a high-level diagram for the approach doc
- [Bug 98](#):¹³ Add Condor-G doc for using GRAM 2 and 5
- [Bug 96](#):¹⁴ GRAM-106 SGE LRM mishandles invalid environment definition
- [Bug 95](#):¹⁵ GRAM-106 SGE LRM doesn't check for executable permissions
- [Bug 94](#):¹⁶ GRAM-106 SGE LRM doesn't check for executable existence
- [Bug 93](#):¹⁷ GRAM-106 SGE LRM script doesn't handle environment vars with whitespace
- [Bug 92](#):¹⁸ stdout to local file doesn't work if count >1
- [Bug 88](#):¹⁹ Missed two phase commit causes job to not be destroyed
- [Bug 86](#):²⁰ Prioritize script invocations to improve throughput
- [Bug 80](#):²¹ GRAM 5 beta2 release
- [Bug 79](#):²² Add support for OSG's "NFS Lite" concept
- [Bug 77](#):²³ GRAM zombie
- [Bug 71](#):²⁴ GRAM protocol test package contains expired test certificate
- [Bug 70](#):²⁵ globus-job-status acts strange for completed jobs in GRAM5
- [Bug 69](#):²⁶ globus-job-get-output -f doesn't work in GRAM5
- [Bug 68](#):²⁷ Bad error when proxy is too short-lived
- [Bug 54](#):²⁸ make globus-job-manager-event-generator not require configuration by default

⁹ <http://jira.globus.org/browse/GRAM-103>

¹⁰ <http://jira.globus.org/browse/GRAM-102>

¹¹ <http://jira.globus.org/browse/GRAM-100>

¹² <http://jira.globus.org/browse/GRAM-99>

¹³ <http://jira.globus.org/browse/GRAM-98>

¹⁴ <http://jira.globus.org/browse/GRAM-96>

¹⁵ <http://jira.globus.org/browse/GRAM-95>

¹⁶ <http://jira.globus.org/browse/GRAM-94>

¹⁷ <http://jira.globus.org/browse/GRAM-93>

¹⁸ <http://jira.globus.org/browse/GRAM-92>

¹⁹ <http://jira.globus.org/browse/GRAM-88>

²⁰ <http://jira.globus.org/browse/GRAM-86>

²¹ <http://jira.globus.org/browse/GRAM-80>

²² <http://jira.globus.org/browse/GRAM-79>

²³ <http://jira.globus.org/browse/GRAM-77>

²⁴ <http://jira.globus.org/browse/GRAM-71>

²⁵ <http://jira.globus.org/browse/GRAM-70>

²⁶ <http://jira.globus.org/browse/GRAM-69>

²⁷ <http://jira.globus.org/browse/GRAM-68>

²⁸ <http://jira.globus.org/browse/GRAM-54>

- [Bug 53](#):²⁹ Generalize log path configuration
- [Bug 51](#):³⁰ configurable control of number of perl scripts that can run simultaneously
- [Bug 47](#):³¹ simplify the throughput tester program and use improved version as doc
- [Bug 24](#):³² Debug/verbose flags for globusrun, globus-job-run
- [Bug 23](#):³³ Improved error codes and error reporting for users
- [Bug 22](#):³⁴ client connections can't be timed out
- [Bug 15](#):³⁵ transition from httpg to https
- [Bug 14](#):³⁶ increase availability of GRAM in linux distributions
- [Bug 12](#):³⁷ Gatekeeper's syslog output cannot be controlled
- [Bug 5](#):³⁸ Add gram-level prologue and epilogue script execution for mpi jobs
- [Bug 4](#):³⁹ Add support for a "managed fork" service
- [Bug 2](#):⁴⁰ Investigate how to setup GRAM5 services in a HA setup

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 5.3 x86_64
 - Debain 4.0 x86_64

²⁹ <http://jira.globus.org/browse/GRAM-53>

³⁰ <http://jira.globus.org/browse/GRAM-51>

³¹ <http://jira.globus.org/browse/GRAM-47>

³² <http://jira.globus.org/browse/GRAM-24>

³³ <http://jira.globus.org/browse/GRAM-23>

³⁴ <http://jira.globus.org/browse/GRAM-22>

³⁵ <http://jira.globus.org/browse/GRAM-15>

³⁶ <http://jira.globus.org/browse/GRAM-14>

³⁷ <http://jira.globus.org/browse/GRAM-12>

³⁸ <http://jira.globus.org/browse/GRAM-5>

³⁹ <http://jira.globus.org/browse/GRAM-4>

⁴⁰ <http://jira.globus.org/browse/GRAM-2>

- Mac OS X
 - Mac OS X 10.5.8

8. Backward compatibility summary

Protocol changes in GRAM since GT4.2.x 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 coallocation 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.

9. Associated Standards

None

10. For More Information

See [GRAM5](#) for more information about this component.

Glossary

J

job scheduler

See the term [scheduler](#)¹.

¹ #scheduler