

GT 4.2.0 RLS : Developer's Guide

DRAFT

GT 4.2.0 RLS : Developer's Guide

Introduction

This guide contains information of interest to developers working with the RLS. It provides reference information for application developers, including APIs, architecture, procedures for using the APIs and code samples.

DRAFT

Table of Contents

1. Before you begin	1
1. Feature summary	1
2. Tested platforms	1
3. Backward compatibility summary	1
4. Technology dependencies	1
5. Replica Location Service (RLS) Security Considerations	2
2. Usage scenarios	3
3. Tutorials	4
4. Architecture and design overview	5
5. APIs	6
1. Programming Model Overview	6
2. Component API	6
6. Command line tools	7
globus-rls-admin	8
globus-rls-cli	10
globus-rls-server	14
7. Configuring RLS	19
1. Configuration overview	19
2. Server configuration file (globus-rls-server.conf)	19
3. Basic configuration	19
4. Host key and certificate configuration	20
5. Configuring LRC to RLI updates	21
6. Configuring the RLS Server for the WS MDS Index Service	22
7. Configuring the RLS Server for the MDS2 GRIS	23
8. Complete RLS Server settings (globus-rls-server.conf)	23
8. Debugging	29
9. Troubleshooting	30
1. Errors	30
10. Related Documentation	31
Glossary	32
Index	33

List of Tables

6.1. Options for globus-rls-admin	9
6.2. Options for globus-rls-cli	10
6.3. Commands for globus-rls-cli	12
6.4. Options for globus-rls-server	17
7.1. Complete RLS Server settings (globus-rls-server.conf)	24
9.1. Replica Locator Service (RLS) Errors	30

DRAFT

Chapter 1. Before you begin

1. Feature summary

Features New in GT 4.2.0

- An embedded database (using SQLite) and ODBC libraries are now included in the RLS Server installation. A default installation of GT, will include a fully configured RLS Server that requires no database installation or configuration by the end user. Advanced users can still use their own database without changes.
- The RLS now includes a new Java client API written entirely in Java. It does not require the C client libraries and allows for Java client support on 64-bit platforms. The new API is fully backward compatible. Users need not make any changes to existing code to take advantage of the new library.

Other Supported Features

- Comprehensive C and Java library for replica registration, replica lookup, replica attributes, index queries, and administrative tasks.
- Command line (`globus-rls-cli`) tool for client operations on catalogs and indexes.
- Command line (`globus-rls-admin`) tool for administrative tasks.

Deprecated Features

- None

2. Tested platforms

Tested platforms for RLS include most 32-bit flavors of Linux and UNIX, including RedHat, Solaris, and others.

3. Backward compatibility summary

Protocol changes since GT 4.0.x

- None

API changes since GT 4.0.x

- None

Exception changes since GT 4.0.x

- None

Schema changes since GT 4.0.x

- None

4. Technology dependencies

RLS depends on the following GT components:

- `globus_core`
- `globus_common`
- `globus_io`
- `globus_gssapi_gsi`
- `globus_usage`

RLS depends on the following 3rd party software:

- RDBMS: MySQL, PostgreSQL, or Oracle
- ODBC manager: unixODBC, iODBC
- ODBC driver: MyODBC, psqLODBC, or Oracle

5. Replica Location Service (RLS) Security Considerations

Security recommendations include:

- *Dedicated User Account:* It is recommended that users create a dedicated user account for installing and running the RLS service (e.g., `globus` as recommended in the general GT installation instructions). This account may be used to install and run other services from the Globus Toolkit.
- *Key and Certificate:* It is recommended that users do not use their `hostkey` and `hostcert` for use by the RLS service. Create a `containerkey` and `containercert` with permissions 400 and 644 respectively and owned by the `globus` user. Change the `rlskeyfile` and `rlscertfile` settings in the RLS configuration file (`$GLOBUS_LOCATION/etc/globus-rls-server.conf`) to reflect the appropriate filenames.
- *LRC and RLI Databases:* Users must ensure security of the RLS data as maintained by their chosen database management system. Appropriate precautions should be made to protect the data and access to the database. Such precautions may include creating a user account specifically for RLS usage, encrypting database users' passwords, etc.
- *RLS Configuration:* It is recommended that the RLS configuration file (`$GLOBUS_LOCATION/etc/globus-rls-server.conf`) be owned by and accessible only by the dedicated user account for RLS (e.g., `globus` account per above recommendations). The file contains the database user account and password used to access the LRC and RLI databases along with important settings which, if tampered with, could adversely affect the RLS service.

Chapter 2. Usage scenarios

This section provides examples illustrating the basic usage of the client interfaces supported by the RLS. Using the client API, developers may create client applications that interact with the RLS server to perform replica location operations.

Developing in C

Client applications developed in C must do *both* of the following:

1. Include the client header file at `$GLOBUS_LOCATION/include/globus_rls_client.h`.
2. Link to the client shared library at `$GLOBUS_LOCATION/lib/libglobus_rls_client_gcc32dbgpthr`.

For [C language example code](#)¹, click [here](#)².

Developing in Java

Client applications developed in Java must do *all* of the following:

1. Include the RLS Jar, `$GLOBUS_LOCATION/lib/rls.jar`, in the CLASSPATH.
2. Import the RLS Package `org.globus.replica.rls.*`.
3. Depend on the client shared library via the Java Native Interface (JNI).

For [Java language example code](#)³, click [here](#)⁴.

¹ test.c

² test.c

³ testrls.java

⁴ testrls.java

Chapter 3. Tutorials

There are no tutorials available at this time.

DRAFT

Chapter 4. Architecture and design overview

The Replica Location Service design consists of two components. *Local Replica Catalogs (LRCs)* maintain consistent information about logical-to-physical mappings on a site or storage system. The *Replica Location Indexes (RLIs)* aggregate state information contained in one or more LRCs and build a global, hierarchical distributed index to support discovery of replicas at multiple sites. LRCs send summaries of their state to RLIs using soft state update protocols. The server consists of a multi-threaded front end server and a back-end relational database, such as MySQL or PostgreSQL. The front end server can be configured to act as an LRC server and/or an RLI server. Clients access the server via a simple string-based RPC protocol. The client APIs support C, Java and Python. The APIs contain operations to create and delete mappings, associate *attributes* with mappings, and perform queries.

Detailed information on the architecture and design can be found in [A Framework for Constructing Scalable Replica Location Services](#)¹ and [Performance and Scalability of a Replica Location Service](#)².

¹ <http://www.isi.edu/~annc/papers/chervenakFinalSC2002.pdf>

² <http://www.isi.edu/~annc/papers/chervenakhpd13.pdf>

Chapter 5. APIs

1. Programming Model Overview

The RLS provides a Client API for C and Java based clients. The RLS Client C API is provided in the form of a library (e.g., .so file). Any installation of RLS will include the shared library as part of the `$GLOBUS_LOCATION/include` and `$GLOBUS_LOCATION/lib` directories. The RLS Client Java API depends on the RLS Client JAR located in the `$GLOBUS_LOCATION/lib` directory.

2. Component API

- [RLS Client C API Documentation](#)¹
- [RLS Client Java API Documentation](#)²

¹ <http://www.isi.edu/%7Eannc/rls/doc/client/index.html>

² <http://www.isi.edu/~annc/rls/doc/client-java/index.html>

Chapter 6. Command line tools

DRAFT

Name

globus-rls-admin -- RLS administration tool

globus-rls-admin

Tool description

Performs administrative operations on an RLS server.

Synopsis

-A/-a/-C option value/-c option/-d/-e/-p/-q/-s/-t timeout/-u/-v [rli] [pattern] [server]

DRAFT

Options

Table 6.1. Options for globus-rls-admin

-A	<p>Adds rli to the list of <i>RLI</i> servers updated by an <i>LRC</i> server using <i>Bloom filters</i>.</p> <p><i>Note:</i> Partitions are not supported with Bloom filters. The LRC server maintains one Bloom filter for all <i>LFNs</i> in its database, which is sent to all RLI servers configured to receive Bloom filter updates with this option.</p>
-a	<p>Adds rli and optionally pattern to the list of RLI servers that the LRC server sends updates to (using a list of LFNs).</p> <p>If pattern is specified, then only LFNs matching it will be sent to rli.</p> <p>If rli is added with no patterns, then it is sent all updates. Pattern matching is done using standard Unix file globbing.</p>
-C option value	<p>Sets server option to value.</p> <p><i>Important:</i> This does <i>not</i> update the configuration file. The next time the server is restarted, the configuration change will be <i>lost</i>.</p>
-c option	<p>Retrieves the configuration value for the specified option from the server.</p> <p>If option is set to all, then all options are retrieved.</p>
-d	<p>Removes rli and pattern from the list of RLI servers that the LRC server sends updates to.</p> <p>If pattern is not specified, then all entries for rli are removed.</p> <p><i>Note:</i> If all patterns are removed separately, then rli is sent all updates. To stop any updates from being sent to rli, do <i>not</i> specify pattern.</p>
-e	<p>Clears the LRC database. Removes all lfn, <i>pfh</i> mappings.</p>
-p	<p>Verifies that the server is responding.</p>
-q	<p>Causes the RLS server to exit.</p>
-S	<p>Shows statistics and other information gathered by the RLS server.</p> <p>This is intended to be input into GRIS.</p>
-s	<p>Shows the list of RLI servers and patterns being sent updates by the LRC server.</p> <p>If rli or pattern are not specified, they are considered wildcards.</p>
-t timeout	<p>Sets timeout (in seconds) for RLS server requests.</p> <p>The default value is 30.</p>
-u	<p>Causes the LRC server to immediately start full soft state updates to any RLI servers previously added with the -a option.</p>
-v	<p>Shows the version and exits.</p>

Name

globus-rls-cli -- RLS client tool

globus-rls-cli

Tool description

Provides a command line interface to some of the functions supported by RLS. It also supports an interactive interface (if *command* is not specified). In interactive mode, double quotes may be used to encode an argument that contains white space.

Synopsis

command [*-c*] [*-h*] [*-l reslimit*] [*-s*] [*-t timeout*] [*-u*] [*command*] *rls-server*

Options

The client command tool uses **getopt** for command line parsing.

Note: Some versions will continue scanning for options (works that begin with a hyphen) for the entire command line, which makes it impossible to specify negative integer or floating point value for an *attribute*. The workaround for this problem is to tell **getopt()** that there are no more options by including 2 hyphens. For example, to specify the value **-2** you must enter **-- -2**.

Table 6.2. Options for globus-rls-cli

-c	Sets "clearvalues" flag when deleting an attribute (will remove any attribute value records when an attribute is deleted).
-h	Shows usage.
-l reslimit	Sets an incremental limit on the number of results returned by a wildcard query at a time. Note that <i>all results will be returned</i> by the client. This parameter only limits the number of results <i>incrementally retrieved</i> by the client during a single internal communication call. For instance, if the wildcard query produces 1000 results and the reslimit is set to 100, the client will internally make 10 calls to the server. From the user's perspective the client will simply return all 1000 results. Zero means no limit.
-s	Uses SQL style wildcards (% and _).
-t timeout	Sets timeout (in seconds) for RLS server requests. The default is 30 seconds.
-u	Uses Unix style wildcards (* and ?).
-v	Shows version.

Commands

DRAFT

Table 6.3. Commands for globus-rls-cli

add <lfn> <pfn>	Adds <i>pfn</i> to mappings of <i>lfn</i> in an <i>LRC</i> catalog.
attribute add <object> <attr> <obj-type> <attr-type>	Adds an attribute to an object, where <i>object</i> should be the lfn or pfn name. <i>obj-type</i> should be one of lfn or pfn. <i>attr-type</i> should be one of date, float, int, or string. If <value> is of type date then it should be in the form "YYYY-MM-DD HH:MM:DD".
attribute bulk add <object> <attr> <obj-type>	Bulk adds attribute values.
attribute bulk delete <object> <attr> <obj-type>	Bulk deletes attributes.
attribute bulk query <attr> <obj-type> <object>	Bulk queries attributes.
attribute define <attr> <obj-type> <attr-type>	Defines a new attribute.
attribute delete <object> <attr> <obj-type>	Removes <i>attribute</i> from <i>object</i> .
attribute modify <object> <attr> <obj-type> <attr-type>	Modifies the value of an attribute.
attribute query <object> <attr> <obj-type>	Retrieves the value of the specified attribute for object .
attribute search <attr> <obj-type> <operator> <attr-type>	Searches for objects which have the specified attribute matching <i>operator</i> and <i>value</i> . <i>operator</i> should be one of =, !=, >, >=, <, or <=.
attribute show <attr> <obj-type>	Shows an attribute definition. If <i>attr</i> is a hyphen (-) then all attributes are shown.
attribute undefine <attr> <obj-type>	Deletes an attribute definition. Will return an error if any objects possess this attribute.
bulk add <lfn> <pfn> [<lfn> <pfn>]	Bulk adds lfn, pfn mappings.
bulk create <lfn> <pfn> [<lfn> <pfn>]	Bulk creates lfn, pfn mappings.
bulk delete <lfn> <pfn> [<lfn> <pfn>]	Bulk deletes lfn, pfn mappings.
bulk query lrc lfn [<lfn> ...]	Bulk queries the LRC for lfns.
bulk query lrc pfn [<pfn> ...]	Bulk queries the LRC for pfns.
bulk query rli lfn [<lfn> ...]	Bulk queries the <i>RLI</i> for lfns.
create <lfn> <pfn>	Creates a new <i>lfn</i> , <i>pfn</i> mapping in an LRC catalog.
delete <lfn> <pfn>	Deletes a <i>lfn</i> , <i>pfn</i> mapping from an LRC catalog.
exit	Exits the interactive session.
help	Prints a help message.
query lrc lfn <lfn>	Queries an LRC server for mappings of <i>lfn</i> .
query lrc pfn <pfn>	Queries an LRC server for mappings to <i>pfn</i> .
query rli lfn <lfn>	Queries an RLI server for mappings of <i>lfn</i> .

query wildcard lrc lfn <lfn-pattern>	Performs a wildcarded query of an LRC server for mappings of <i>lfn-pattern</i> . Patterns use the standard Unix wildcard characters: an asterisk (*) matches 0 or more characters, and a question mark (?) matches any single character.
query wildcard lrc pfn <pfn-pattern>	Queries an LRC server for mappings to <i>pfn-pattern</i> . Patterns use the standard Unix wildcard characters: an asterisk (*) matches 0 or more characters, and a question mark (?) matches any single character.
query wildcard rli lfn <lfn-pattern>	Queries an RLI server for mappings of <i>lfn-pattern</i> . Patterns use the standard Unix wildcard characters: an asterisk (*) matches 0 or more characters, and a question mark (?) matches any single character.
set reslimit <limit>	<p>Sets an incremental limit on the number of results returned by a wildcard query at a time.</p> <p>Note that <i>all results will be returned</i> by the client. This parameter only limits the number of results <i>incrementally retrieved</i> by the client during a single internal communication call. For instance, if the wildcard query produces 1000 results and the reslimit is set to 100, the client will internally make 10 calls to the server. From the user's perspective the client will simply return all 1000 results.</p>
set timeout <timeout>	<p>Sets the timeout (in seconds) on calls to the RLS server.</p> <p>The default value is 30.</p>
version	Shows the version and exits.

Name

globus-rls-server -- RLS server tool

globus-rls-server

Tool description

The RLS server (**globus-rls-server**) can be configured as either one or both of the following:

- *Location Replica Catalog (LRC)* server, which manages *Logical FileName (LFN)* to *Physical FileName (PFN)* mappings in a database. *Note:* If **globus-rls-server** is configured as an LRC server, the *RLI* servers that it sends updates to should be added to the database using **globus-rls-admin**.
- *Replica Location Index (RLI)* server, which manages mappings of LFNs to LRC servers.

Clients wishing to locate one or more physical filenames associated with a logical filename should first contact an RLI server, which will return a list of LRCs that may know about the LFN. The LRC servers are then contacted in turn to find the physical filenames.

Note: RLI information may be out of date, so clients should be prepared to get a negative response when contacting an LRC (or no response at all if the LRC server is unavailable).

Synopsis

```
[ -B lrc_update_bf ] [ -b maxbackoff ] [ -C rlscertfile ] [ -c conffile ] [ -d ] [ -e rli_expire_int ] [ -F lrc_update_factor ] [ -f maxfreethreads ] [ -I true/false ] [ -i idletimeout ] [ -K rlskeyfile ] [ -L loglevel ] [ -l true/false ] [ -M maxconnections ] [ -m maxthreads ] [ -N ] [ -o lrc_buffer_time ] [ -p pidfiledir ] [ -r true/false ] [ -S rli_expire_stale ] [ -s starthreads ] [ -t timeout ] [ -U myurl ] [ -u lrc_update_ll ] [ -v ]
```

LRC to RLI Updates

Two methods exist for LRC servers to inform RLI servers of their LFNs.

- By default, the LFNs are sent from the LRC to the RLI. This can be time consuming if the number of LFNs is large, but it does give the RLI an exact list of the LFNs known to the LRC, and it allows wildcard searching of the RLI.
- Alternatively, *Bloom filters* may be sent, which are highly compressed summaries of the LFNs. However, they do not allow wildcard searching and will generate more "false positives" when querying an RLI.

Please see below for more on Bloom filters.

globus-rls-admin can be used to manage the list of RLIs that an LRC server updates. This includes partitioning LFNs among multiple RLI servers.

A soft state algorithm is used in both update modes: periodically the LRC server sends its state (LFN information) to the RLI servers it updates. The RLI servers add these LFNs to their indexes or update timestamps if the LFNs were already known. RLI servers expire information about LFN, LRC mappings if they haven't been updated for a period longer than the soft state update interval.

The following options in the configuration file control the soft state algorithm when an LRC updates an RLI by sending LFNs:

- **rli_expire_int** (seconds)
- **rli_expire_stale** (seconds)
- **lrc_update_ll** (seconds)
- **lrc_update_bf** (seconds)

Updates to an LRC (new LFNs or deleted LFNs) normally don't propagate to RLI servers until the next soft state update (controlled by options **lrc_update_ll** and **lrc_update_bf**).

However, by enabling "immediate update" mode (set **lrc_update_immediate** to **true**), an LRC will send updates to an RLI within **lrc_buffer_time** seconds.

If updates are done with LFN lists then only the LFNs that have been added or deleted to the LRC are sent. If Bloom filters are used, then the entire Bloom filter is sent.

When immediate updates are enabled, the interval between soft state updates is multiplied by **lrc_update_factor** as long as no updates have failed (LRC and RLI are considered to be in sync). This can greatly reduce the number of soft state updates an LRC needs to send to an RLI.

Incremental updates are buffered by the LRC server until either 200 updates have accumulated (when LFN lists are used), or **lrc_buffer_time** seconds have passed since the last update.

Bloom filter updates

A Bloom filter is an array of bits. Each LFN is hashed multiple times and the corresponding bits in the Bloom filter are set.

Querying an RLI to verify if an LFN exists is done by performing the same hashes and checking if the bits in the filter are on. If not, then the LFN is known not to exist. If they're all on, then all that's known is that the LFN probably exists.

The size of the Bloom filter (as a multiple of the number of LFNs) and the number of hash functions control the false positive rate. The default values of 10 and 3 give a false positive rate of approximately 1%.

The advantage of Bloom filters is their efficiency. For example, if the LRC has 1,000,000 LFNs in its database, with an average length of 20 bytes, then 20,000,000 bytes must be sent to an RLI during a soft state update (assuming no partitioning). The RLI server must perform 1,000,000 updates to its database to create new LFN, LRC mappings or update timestamps on existing entries. With Bloom filters only 1,250,000 bytes are sent (10 x 1,000,000 bits / 8), and there are no database operations on the RLI (Bloom filters are maintained entirely in memory). A comparison of the time to perform a 1,000,000 LFN update: it took 20 minutes sending all the LFNs and less than 1 second using a Bloom filter. However as noted before, Bloom filters do *not* support wild card searches of an RLI.

Note: An LRC server can update some RLIs with Bloom filters and others with LFNs. However, an RLI server can only be updated using one method.

The following options in the [Configuration](#) file control Bloom filter updates:

- **rli_bloomfilter** true/false
- **rli_bloomfilter_dir** none/default/pathname
- **lrc_bloomfilter_numhash** N
- **lrc_bloomfilter_ratio** N

- `irc_update_bf` seconds

Log Messages

`globus-rls-server` uses syslog to log errors and other information (facility **LOG_DAEMON**) when it's running in normal (daemon) mode.

If the `-d` option (debug) is specified, then log messages are written to stdout.

Signals

The server will reread its configuration file if it receives a **HUP** signal. It will wait for all current requests to complete and shut down cleanly if sent any of the following signals: **INT**, **QUIT** or **TERM**.

Options (`globus-rls-server`)

The following table describes the command line options available for `globus-rls-server`:

DRAFT

Table 6.4. Options for globus-rls-server

-b maxbackoff	Maximum time (in seconds) that globus-rls-server will attempt to reopen the socket it listens on after an I/O error.
-C rls-certfile	Name of the X.509 certificate file that identifies the server; sets environment variable X509_USER_CERT .
-c conffile	Name of the configuration file for the server. The default is \$GLOBUS_LOCATION/etc/globus-rls-server.conf if the environment variable GLOBUS_LOCATION is set; else, /usr/local/etc/globus-rls-server.conf .
-d	Enables debugging. The server will not detach from the controlling terminal, and log messages will be written to stdout rather than syslog. For additional logging verbosity set the loglevel (see the -L option) to higher values.
-e rli_expire_int	Interval (seconds) at which an RLI server should expire stale entries.
-F lrc_update_factor	If lrc_update_immediate mode is on, and the LRC server is in sync with an RLI server (an LRC and RLI are synced if there have been no failed updates since the last full soft state update), then the interval between RLI updates for this server (lrc_update_ll) is multiplied by lrc_update_factor .
-f maxfreethreads	Maximum number of idle threads the server will leave running. Excess threads are terminated.
-I true false	Turns LRC to RLI immediate update mode on (true) or off (false). The default value is false .
-i idletimeout	Seconds after which idle client connections are timed out.
-K rls-keyfile	Name of the X.509 key file. Sets environment variable X509_USER_KEY .
-L loglevel	Sets the log level. By default this is 0 , which means only errors will be logged. Higher values mean more verbose logging.
-l true false	Configures whether the server is an LRC server. The default is false .
-M maxconnections	Maximum number of active connections. It should be small enough to prevent the server from running out of open file descriptors. The default value is 100 .
-m maxthreads	Maximum number of threads server will start up to support simultaneous requests.
-N	Disables authentication checking. This option is intended for debugging. Clients should use the URL RLSN://host to disable authentication on the client side.
-o lrc_buffer_time	LRC to RLI updates are buffered until either the buffer is full or this much time (in seconds) has elapsed since the last update. The default value is 30 .
-p pidfiledir	Directory where PID files should be written.

-r	Configures whether the server is an RLI server. The default value is false .
-S rli_expire_stale	Interval (in seconds) after which entries in the RLI database are considered stale (presumably because they were deleted in the LRC). Stale entries are not returned in queries.
-s startthreads	Number of threads to start up initially.
-t timeout	Timeout (in seconds) for calls to other RLS servers (in other words, for LRC calls to send an update to an RLI). A value of 0 disables timeouts. The default value is 30 .
-U myurl	URL for this server.
-u lrc_update_ll	Interval (in seconds) between lfn-list LRC to RLI updates.
-v	Shows version and exits.

Chapter 7. Configuring RLS

1. Configuration overview

RLS configuration involves statically-defined, system settings as defined in the RLS configuration file (see `$GLOBUS_LOCATION/etc/globus-rls-server.conf`), settings changed temporarily at run-time using the RLS Admin tool (see `globus-rls-admin(1) -C option value` command), and finally LRC-to-RLI and RLI-to-RLI updates configured using the RLS Admin tool (see `globus-rls-admin(1) -a, -A, -d` commands).

2. Server configuration file (`globus-rls-server.conf`)

Configuration settings for the RLS are specified in the `globus-rls-server.conf` file. If the configuration file is not specified on the command line (see the `-c` option) then it is looked for in both:

- `$GLOBUS_LOCATION/etc/globus-rls-server.conf`
- `/usr/local/etc/globus-rls-server.conf` if `GLOBUS_LOCATION` is not set



Note

Command line options always override items found in the configuration file.

The configuration file is a sequence of lines consisting of a keyword, whitespace, and a value. Comments begin with `#` and end with a newline.

3. Basic configuration

Review the server configuration file `$GLOBUS_LOCATION/etc/globus-rls-server.conf` and change any options you want. The server man page `globus-rls-server(8)` has complete details on all options. The complete details are also provided later in this section.

A minimal configuration file for both an LRC and RLI server would be:

```
# Configure the database connection info
db_user      dbuser
db_pwd       dbpassword

# If the server is an LRC server
lrc_server   true
lrc_dbname   lrc1000

# If the server is an RLI server
rli_server   true
rli_dbname   rli1000 # Not needed if updated by Bloom filters

# Configure who can make requests of the server
acl .*: all
```

```
# RE matching grid-mapfile users or DNs from x509 certs
...
```

4. Host key and certificate configuration

The server uses a host certificate to identify itself to clients. By default this certificate is located in the files `/etc/grid-security/hostcert.pem` and `/etc/grid-security/hostkey.pem`. Host certificates have a distinguished name of the form `/CN=host/FQDN`. If the host you plan to run the RLS server on does not have a host certificate, you must obtain one from your Certificate Authority. The RLS server must be run as the same user who owns the host certificate files (typically root). The location of the host certificate files may be specified in `$GLOBUS_LOCATION/etc/globus-rls-server.conf`:

```
rlscertfile      path-to-cert-file    # default /etc/grid-security/hostcert.pem
rlskeyfile       path-to-key-file     # default /etc/grid-security/hostkey.pem
```

It is possible to run the RLS server without authentication, by starting it with the `-N` option, and using URL's of the form `rlsn://server` to connect to it. Notice that the URL scheme is `rlsn` as opposed to `rls`.

It is generally recommended to run the server with a user account other than root for added security. In order to do so, you will need to create complimentary key and certificate files owned by a designated user account, `globus` for instance.

1. Begin by copying the `/etc/grid-security/hostcert.pem` and `/etc/grid-security/hostkey.pem` to `/etc/grid-security/containercert.pem` and `/etc/grid-security/containerkey.pem`. Note that we use the prefix "container" to conform with the recommended naming scheme for other services distributed with the Globus Toolkit.

```
% cp /etc/grid-security/hostcert.pem /etc/grid-security/containercert.pem
% cp /etc/grid-security/hostkey.pem /etc/grid-security/containerkey.pem
```

2. Then change ownership of the files to the designated user account, `globus` in our example.

```
% chown globus /etc/grid-security/containercert.pem
% chown globus /etc/grid-security/containerkey.pem
```

3. Change the `rlskeyfile` and `rlscertfile` settings in the RLS configuration file (`$GLOBUS_LOCATION/etc/globus-rls-server.conf`) to reflect the appropriate filenames.

```
rlscertfile      /etc/grid-security/containercert.pem
rlskeyfile       /etc/grid-security/containerkey.pem
```

4. Finally, bear in mind that your certificate and key files must always have file permissions 644 and 400 respectively.

```
% ls -l /etc/grid-security/*.pem
-rw-r--r--  1 globus  gridstaff    818 Dec  8  2005 /etc/grid-security/containercer
-r-----   1 globus  gridstaff    887 Dec  8  2005 /etc/grid-security/containerkey
-rw-r--r--  1 root    root        818 Dec  8  2005 /etc/grid-security/hostcert.pem
-r-----   1 root    root        887 Dec  8  2005 /etc/grid-security/hostkey.pem
```

If authentication is enabled, RLI servers must include `acl` configuration options that match the identities of LRC servers that update it and that grant the `rli_update` permission to the LRCs.

5. Configuring LRC to RLI updates

One of the key benefits to using the RLS for managing replica location information is its distributed architecture. In a distributed deployment, one or more Local Replica Catalog (LRC) services will send updates of its contents to one or more Replica Location Index (RLI) services.

By default the installed LRC is not configured to send updates to any RLI, even the local RLI co-located with the local LRC. Use the `globus-rls-admin(1)` tool to configure the LRC to send updates to one or more RLI services.

- To configure the LRC to send uncompressed lists of its logical names to a RLI, use the following command:

```
% $GLOBUS_LOCATION/bin/globus-rls-admin -a rls://rli_host rls://lrc_host
```

- To configure the LRC to send compressed bitmaps (using Bloom filters) of its logical names to a RLI, use the following command:

```
% $GLOBUS_LOCATION/bin/globus-rls-admin -A rls://rli_host rls://lrc_host
```

- To configure the LRC to stop sending updates to a RLI, use the following command:

```
% $GLOBUS_LOCATION/bin/globus-rls-admin -d rls://rli_host rls://lrc_host
```

Note

While any given LRC is capable of sending uncompressed or compressed updates to any RLI. The RLI service must be configured to accept either uncompressed or compressed updates but not both. See the `rli_bloom-filter` setting of the RLS configuration file for more details.

There are tradeoffs between using uncompressed and compressed updates in your configuration. The advantage of using compressed updates, not surprisingly, is a significant reduction in network overhead and memory usage. As replica location mappings grow into the 10's of millions or more, the savings of using compressed updates becomes important. On the other hand, due to the compressed nature of the Bloom filter bitmap used to represent the logical names in the LRC, the wildcard query at the RLI cannot be supported when update compression is used.

6. Configuring the RLS Server for the WS MDS Index Service

The server package includes a script `$GLOBUS_LOCATION/libexec/aggrexec/globus-rls-aggregator-source.pl` that may be used as an Execution Aggregator Source by WS MDS. See [GT 4.2.0 Index Services](#) for more information on setting up and using the Execution Aggregator Source scripts in WS MDS. The script may be invoked as follows and will generate output in the format as depicted.

```
% $GLOBUS_LOCATION/libexec/aggrexec/globus-rls-aggregator-source.pl rls://mysite
<?xml version="1.0" encoding="UTF-8"?>
<rlsStats>
  <site>rls://mysite</site>
  <version>4.0</version>
  <uptime>03:08:15</uptime>
  <serviceList>
    <service>lrc</service>
    <service>rli</service>
  </serviceList>
  <lrc>
    <updateMethodList>
      <updateMethod>lfnlist</updateMethod>
      <updateMethod>bloomfilter</updateMethod>
    </updateMethodList>
    <updatesList>
      <updates>
        <site>rls://myothersite:39281</site>
        <method>bloomfilter</method>
        <date>08/01/05</date>
        <time>16:16:38</time>
      </updates>
    </updatesList>
    <numlfn>283902</numlfn>
    <numpfn>593022</numpfn>
    <nummap>593022</nummap>
  </lrc>
  <rli>
    <updatedViaList>
      <updatedVia>bloomfilters</updatedVia>
    </updatedViaList>
    <updatedByList>
      <updatedBy>
        <site>rls://myothersite:39281</site>
        <date>08/01/05</date>
        <time>10:03:21</time>
      </updatedBy>
    </updatedByList>
  </rli>
</rlsStats>
```

Important

Be sure to configure the security context of the container running the MDS, and be sure that the security configuration on the RLS host recognizes the MDS security context.

When following the instructions provided by the [GT 4.2.0 Index Services](#), you will need to consider the security context used by the MDS to invoke the Execution Aggregator Source script provided by RLS. Most deployments of RLS run the service with security enabled. Therefore any client connections, including administrative status operations, require authentication and authorization. In order for MDS to use the provided script to check RLS status, it must invoke the script with a valid user proxy or user certificate and key. The RLS must recognize the DN from the user certificate (i.e., the DN should be in the gridmap file).

One way to configure the MDS security context for use with RLS monitoring is to set the environment variables `X509_USER_CERT` and `X509_USER_KEY` to point to the container certificate and key. Run the MDS with these environment settings. Also, add the DN from the container certificate to the gridmap file on the host running the RLS.

Alternatively, you could modify the provided script so that it sets the environment variables to another user certificate and key (or proxy) as desired before calling the RLS.

7. Configuring the RLS Server for the MDS2 GRIS

The server package includes a program called `globus-rls-reporter` that will report information about an RLS server to the MDS2 GRIS. Use this procedure to enable this program:

1. To enable Index Service reporting, add the contents of the file `$GLOBUS_LOCATION/setup/globus/rls-ldif.conf` to the MDS2 GRIS configuration file `$GLOBUS_LOCATION/etc/grid-info-resource-ldif.conf`.
2. If necessary, set your virtual organization (VO) name in `$GLOBUS_LOCATION/setup/globus/rls-ldif.conf`. The default value is `local`. The VO name is referenced twice, on the lines beginning `dn:` and `args:`.
3. You must restart your MDS (GRIS) server after modifying `$GLOBUS_LOCATION/etc/grid-info-resource-ldif.conf`. You can use the following commands to do so:

```
$GLOBUS_LOCATION/sbin/SXXgris stop
$GLOBUS_LOCATION/sbin/SXXgris start
```

8. Complete RLS Server settings (globus-rls-server.conf)

This section describes the complete details of the RLS Server configuration settings.

Table 7.1. Complete RLS Server settings (globus-rls-server.conf)

<pre>acl user: permission [permission]</pre>	<p>acl entries may be a combination of DNs and local usernames. If a DN is not found in the gridmap file then the file is used to search the acl list.</p> <p>A gridmap file may also be used to map DNs to local usernames, which in turn are matched against the regular expressions in the acl list to determine the user's permissions.</p> <p>user is a regular expression matching distinguished names (or local usernames if a gridmap file is used) of users allowed to make calls to the server.</p> <p>There may be multiple acl entries, with the first match found used to determine a user's privileges.</p> <p>[permission] is one or more of the following values:</p> <ul style="list-style-type: none"> • lrc_read Allows client to read an <i>LRC</i>. • lrc_update Allows client to update an LRC. • rli_read Allows client to read an <i>RLI</i>. • rli_update Allows client to update an RLI. • admin Allows client to update an LRC's list of RLIs to send updates to. • stats Allows client to read performance statistics. • all Allows client to do all of the above.
<pre>authentication true false</pre>	<p>Enable or disable GSI authentication.</p> <p>The default value is true.</p> <p>If authentication is enabled (true), clients should use the URL schema rls: to connect to the server.</p> <p>If authentication is <i>not</i> enabled (false), clients should use the URL schema rlsn:.</p>
<pre>db_pwd password</pre>	<p>Password to use to connect to the database server.</p> <p>The default value is changethis.</p>
<pre>db_user data-baseuser</pre>	<p>Username to use to connect to database server.</p> <p>The default value is dbperson.</p>
<pre>idletimeout seconds</pre>	<p>Seconds after which idle connections close.</p> <p>The default value is 900.</p>
<pre>loglevel N</pre>	<p>Sets loglevel to N (default is 0). Higher levels mean more verbosity.</p>

<code>lrc_bloomfilter_numhash N</code>	<p>Number of hash functions to use in <i>Bloom filters</i>.</p> <p>The default value is 3.</p> <p>Possible values are 1 through 8.</p> <p>This value, in conjunction with <code>lrc_bloomfilter_ratio</code>, will determine the number of false positives that may be expected when querying an RLI that is updated via Bloom filters.</p> <p><i>Note:</i> The default values of 3 and 10 give a false positive rate of approximately 1%.</p>
<code>lrc_bloomfilter_ratio N</code>	<p>Sets ratio of bloom filter size (in bits) to number of <i>LFNs</i> in the LRC catalog (in other words, size of the Bloom filter as a multiple of the number of LFNs in the LRC database.) This is only meaningful if Bloom filters are used to update an RLI. Too small a value will generate too many false positives, while too large a value wastes memory and network bandwidth.</p> <p>The default value is 10.</p> <p><i>Note:</i> The default values of 3 and 10 give a false positive rate of approximately 1%.</p>
<code>lrc_buffer_time N</code>	<p>LRC to RLI updates are buffered until either the buffer is full or this much time in seconds has elapsed since the last update.</p> <p>The default value is 30.</p>
<code>lrc_dbname</code>	<p>Name of LRC database.</p> <p>The default value is <code>lrcdb</code>.</p>
<code>lrc_server true false</code>	<p>If LRC server, the value should be <code>true</code>.</p> <p>The default value is <code>false</code>.</p>
<code>lrc_update_bf seconds</code>	<p>Interval in seconds between LRC to RLI updates when the RLI is updated by Bloom filters. In other words, how often an LRC server does a Bloom filter soft state update.</p> <p>This can be much smaller than the interval between updates without using Bloom filters (<code>lrc_update_ll</code>).</p> <p>The default value is 300.</p>
<code>lrc_update_factor N</code>	<p>If <code>lrc_update_immediate</code> mode is on, and the LRC server is in sync with an RLI server (an LRC and RLI are synced if there have been no failed updates since the last full soft state update), then the interval between RLI updates for this server (<code>lrc_update_ll</code>) is multiplied by the value of this option.</p>
<code>lrc_update_immediate true false</code>	<p>Turns LRC to RLI immediate mode updates on (<code>true</code>) or off (<code>false</code>).</p> <p>The default value is <code>false</code>.</p>
<code>lrc_update_ll seconds</code>	<p>Number of seconds before an LRC server does an LFN list soft state update.</p> <p>The default value is 86400.</p>

<code>lrc_update_retry seconds</code>	Seconds to wait before an LRC server will retry to connect to an RLI server that it needs to update. The default value is 300.
<code>maxbackoff seconds</code>	Maximum seconds to wait before re-trying listen in the event of an I/O error. The default value is 300 .
<code>maxfreethreads N</code>	Maximum number of idle threads. Excess threads are killed. The default value is 5 .
<code>maxconnections N</code>	Maximum number of simultaneous connections. The default value is 100.
<code>maxthreads N</code>	Maximum number of threads running at one time. The default value is 30.
<code>myurl URL</code>	URL of server. The default value is <code>rls://<hostname>:port</code>
<code>odbcini filename</code>	Sets environment variable ODBCINI. If not specified, and ODBCINI is not already set, then the default value is <code>\$GLOBUS_LOCATION/var/odbc.ini</code> .
<code>pidfile filename</code>	Filename where pid file should be written. The default value is <code>\$GLOBUS_LOCATION/var/<programname>.pid</code> .
<code>port N</code>	Port the server listens on. The default value is 39281 .
<code>result_limit limit</code>	Sets the maximum number of results returned by a query. The default value is 0 (zero), which means no limit. If a query request includes a limit greater than this value, an error (GLOBUS_RLS_BADARG) is returned. If the query request has no limit specified, then at most <code>result_limit</code> records are returned by a query.
<code>rli_bloomfilter true false</code>	RLI servers must have this set to accept Bloom filter updates. If <code>true</code> , then only Bloom filter updates are accepted from LRCs. If <code>false</code> , full LFN lists are accepted. <i>Note:</i> If Bloom filters are enabled, then the RLI does <i>not</i> support wildcarded queries.

<code>rli_bloomfilter_dir</code> <code>none default path-</code> <code>name</code>	<p>If an RLI is configured to accept bloom filters (<code>rli_bloomfilter true</code>), then Bloom filters may be saved to this directory after updates.</p> <p>This directory is scanned when an RLI server starts up and is used to initialize Bloom filters for each LRC that updated the RLI.</p> <p>This option is useful when you want the RLI to recover its data immediately after a restart rather than wait for LRCs to send another update.</p> <p>If the LRCs are updating frequently, this option is unnecessary and may be wasteful in that each Bloom filter is written to disk after each update.</p> <ul style="list-style-type: none"> • <code>none</code> <p>Bloom filters are not saved to disk.</p> <p>This is the default.</p> • <code>default</code> <p>Bloom filters are saved to the default directory:</p> <ul style="list-style-type: none"> • <code>\$GLOBUS_LOCATION/var/rls-bloomfilters</code> if <code>GLOBUS_LOCATION</code> is set • <code>else, /tmp/rls-bloomfilters</code> • <code>pathname</code> <p>Bloom filters are saved to the named directory.</p> <p>Any other string is used as the directory name unchanged.</p> <p>The Bloom filter files in this directory have the name of the URL of the LRC that sent the Bloom filter, with slashes(/) changed to percent signs (%) and ".bf" appended.</p>
<code>rli_dbname</code> <code>database</code>	<p>Name of the RLI database.</p> <p>The default value is <code>rlidb</code>.</p>
<code>rli_expire_int</code> <code>seconds</code>	<p>Interval (in seconds) between RLI expirations of stale entries. In other words, how often an RLI server will check for stale entries in its database.</p> <p>The default value is <code>28800</code>.</p>
<code>rli_expire_stale</code> <code>seconds</code>	<p>Interval (in seconds) after which entries in the RLI database are considered stale (presumably because they were deleted in the LRC).</p> <p>The default value is <code>86400</code>.</p> <p>This value should be no smaller than <code>lrc_update_ll</code>.</p> <p>Stale RLI entries are not returned in queries.</p> <p><i>Note:</i> If the LRC server is responding, this value is not used. Instead the value of <code>lrc_update_ll</code> or <code>lrc_update_bf</code> is retrieved from the LRC server, multiplied by 1.2, and used as the value for this option.</p>

<code>rli_server</code> <code>true false</code>	If an RLI server, the value should be <code>true</code> . The default value is <code>false</code> .
<code>rlscertfile filename</code>	Name of the X.509 certificate file identifying the server. This value is set by setting environment variable <code>X509_USER_CERT</code> .
<code>rlskeyfile filename</code>	Name of the X.509 key file for the server. This value is set by setting environment variable <code>X509_USER_KEY</code> .
<code>startthreads N</code>	Number of threads to start initially. The default value is 3.
<code>timeout seconds</code>	Timeout (in seconds) for calls to other RLS servers (e.g., for LRC calls to send an update to an RLI).

Chapter 8. Debugging

To run the RLS server in debug mode, use the `-d` option along with the `-L num` option (e.g., `$GLOBUS_LOCATION/bin/globus-rls-server -d -L 3`). The `-d` option instructs the RLS server to direct log output to `stdout`, while the `-L num` option sets the log level where a higher `num` results in more detailed output.

DRAFT

Chapter 9. Troubleshooting

Information on troubleshooting can be found in the [FAQ](#)¹. For a list of common errors in GT, see [Error Codes](#).

1. Errors

Table 9.1. Replica Locator Service (RLS) Errors

Error Code	Definition	Possible Solutions
Error with credential: The proxy credential: <credential> with subject: <subject> expired <minutes> minutes ago	Expired proxy credential	Create a new proxy with grid-proxy-init .
Unable to connect to localhost:xxxx	Unable to connect to the local host. This can be due to a variety of reasons, including a wrong address or port number in the RLS connection URL or an issue with a firewall configuration.	<ul style="list-style-type: none"> • Double-check the address and port number in the RLS connection URL. parameters are correct. • If a firewall configuration is preventing connections to the target host for a particular port, you may need to consult the system administrator.
"connection timeout"	<p>At times, a client may experience a connection timeout when interacting with the RLS server due to a variety of reasons:</p> <ul style="list-style-type: none"> • One reason could simply be due to wide-area network latency or congestion. • Another situation that users eventually encounter is due to scaling of the system. As the RLS server's database of replica location mappings grows in size, some query operations, such as bulk queries involving large quantities of mappings or wildcard queries that result in a large subset of mappings, will begin to take more time both to process the query and to return the large results set to the client over the network. 	<p>If timeouts are experienced with increasing frequency, increase the RLS server's timeout configuration parameter found in the <code>\$GLOBUS_LOCATION/var/globus-rls-server.conf</code> file. You may also use the <code>-t</code> timeout option of the globus-rls-cli tool.</p>

¹ http://www.globus.org/toolkit/data/rls/rls_faq.html

Chapter 10. Related Documentation

For additional details, see the [RPC Protocol Description](#)¹.

DRAFT

¹ [rpcprotocol.pdf](#)

Glossary

B

Bloom filter Compression scheme used by the Replica Location Service (RLS) that is intended to reduce the size of soft state updates between Local Replica Catalogs (LRCs) and Replica Location Index (RLI) servers. A Bloom filter is a bit map that summarizes the contents of a Local Replica Catalog (LRC). An LRC constructs the bit map by applying a series of hash functions to each logical name registered in the LRC and setting the corresponding bits.

L

Local Replica Catalog (LRC) Stores mappings between logical names for data items and the target names (often the physical locations) of replicas of those items. Clients query the LRC to discover replicas associated with a logical name. Also may associate attributes with logical or target names. Each LRC periodically sends information about its logical name mappings to one or more RLIs.

See also [RLI](#)⁶.

logical file name A unique identifier for the contents of a file.

P

physical file name The address or the location of a copy of a file on a storage system.

R

Replica Location Index (RLI) Collects information about the logical name mappings stored in one or more Local Replica Catalogs (LRCs) and answers queries about those mappings. Each RLI periodically receives updates from one or more LRCs that summarize their contents.

RLS attribute Descriptive information that may be associated with a logical or target name mapping registered in a Local Replica Catalog (LRC). Clients can query the LRC to discover logical names or target names that have specified RLS attributes.

⁶ #rli

Index

A

- administrative operations
 - configuring LRC server to stop updating RLI, 8
 - configuring LRC-to-RLI updates
 - compressed (Bloom filters), 8
 - uncompressed, 8
 - configuring settings (runtime only), 8
 - pinging the server, 8
 - stopping RLS server, 8
- architecture
 - for admin, 5

C

- client operations
 - attributes
 - adding an attribute to an lfn or pfn, 10
 - bulk adding attribute values, 10
 - bulk deleting attribute values, 10
 - bulk querying attributes, 10
 - defining a new attribute, 10
 - deleting an attribute definition, 10
 - modifying the value of an attribute, 10
 - removing an attribute, 10
 - retrieving the value of the specified attribute for an lfn or pfn, 10
 - searching for lfns or pfns which have the specified matching operator and value, 10
 - showing an attribute definition, 10
 - basic
 - adding physical filenames to mappings of logical filenames in a LocalReplicaCatalog, 10
 - creating a new lfn, pfn mapping in an LRC catalog, 10
 - deleting a lfn, pfn mapping from an LRC catalog, 10
 - exiting the interactive session, 10
 - bulk
 - bulk adding lfn, pfn mappings, 10
 - bulk creating lfn, pfn mappings, 10
 - bulk deleting lfn, pfn mappings, 10
 - bulk querying the LRC for lfns, 10
 - bulk querying the LRC for pfns, 10
 - bulk querying the RLI for lfns, 10
 - querying
 - performing a wildcarded query of an LRC server for mappings of lfn-pattern, 10
 - querying an LRC server for mappings of lfn, 10
 - querying an LRC server for mappings of pfn, 10

- querying an LRC server for mappings to pfn-pattern, 10
- querying an RLI server for mappings of lfn, 10
- querying an RLI server for mappings to lfn-pattern, 10
- using interactive mode, 10

D

- debugging, 29

E

- errors, 30

S

- server operations, 14
 - configuring RLS server as Location Replica Catalog (LRC), 14
 - configuring RLS server as Replica Location Index (RLI), 14