GT 4.2.1 RLS : Developer's Guide
GT 4.2.1 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.
# 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  
   1. Java Examples ......................................................................................................................... 3  
   2. Example Code ......................................................................................................................... 5  
3. Tutorials ......................................................................................................................................... 7  
4. Architecture and design overview ........................................................................................................ 8  
5. APIs .............................................................................................................................................. 9  
   1. Programming Model Overview ................................................................................................... 9  
   2. Component API ....................................................................................................................... 9  
6. Command line tools ........................................................................................................................ 10  
   globus-rls-admin ................................................................. 11  
   globus-rls-cli ............................................................................................................................. 13  
   globus-rls-server ........................................................................................................................ 17  
7. Configuring RLS ............................................................................................................................. 22  
   1. Configuration overview ........................................................................................................... 22  
   2. Server configuration file (globus-rls-server.conf) ...................................................................... 22  
   3. Basic configuration .................................................................................................................. 22  
   4. Host key and certificate configuration ...................................................................................... 23  
   5. Configuring LRC to RLI updates ............................................................................................... 24  
   6. Configuring the RLS Server for the WS MDS Index Service ..................................................... 25  
   7. Configuring the RLS Server for the MDS2 GRIS .......................................................................... 26  
   8. Complete RLS Server settings (globus-rls-server.conf) ............................................................. 26  
8. Debugging .................................................................................................................................... 32  
9. Troubleshooting ............................................................................................................................. 33  
   1. Errors ................................................................................................................................... 33  
10. Related Documentation .................................................................................................................. 34  
    Glossary ...................................................................................................................................... 35  
    Index ....................................................................................................................................... 36
List of Tables

6.1. Options for globus-rls-admin ................................................................. 12
6.2. Options for globus-rls-cli ................................................................. 13
6.3. Commands for globus-rls-cli ............................................................. 15
6.4. Options for globus-rls-server ........................................................... 20
7.1. Complete RLS Server settings (globus-rls-server.conf) ..................... 27
9.1. Replica Locator Service (RLS) Errors ................................................. 33
Chapter 1. Before you begin

1. Feature summary

Features New in GT 4.2.1

• None since GT 4.2.0.

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, Debian, CentOS, SUSE, Solaris/Sparc, Solaris/x86, 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
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

1. Java Examples

This section provides examples of a few basic operations using the new Java client API. Here is an outline of the typical steps used to resolve a replica location.

- Establish a connection to the Replica Location Index service
- Construct a list of the logical names to be used in the query
- Query the index service
- Inspect the return list and construct lists of logical names to be used in queries to the Local Replica Catalog services
- Query the catalog services
- Inspect the results returned by the catalog services

1.1. Create a connection source

A connection source is needed in order to establish connections to the RLS. The connection source may be shared and is a thread safe object. The SimpleRLSConnectionSource may be directly instantiated by the client or it may be used as a JNDI object and shared by multiple clients (e.g., in a container that supports JNDI). In this example, the client instantiates the connection source.

```java
RLSConnectionSource source = null;
try {
    source = new SimpleRLSConnectionSource();
} catch (RLSException e) {
    // handle exception
}
```

1.2. Create a connection

Use the connection source to establish a connection. If the source has defaults you may use its parameterless connect() method, otherwise you must supply the connection URL and credentials.

```java
RLSConnection connection = null;
try {
    connection = source.connect(url, credential);
} catch (RLSException e) {
    // handle exception
}
```
1.3. Create a simple index query

Query objects are used to represent different types of RLS queries. There are simple queries, batch queries, and attribute searches. This examples uses a simple query object. We begin by querying the RLS index service which tells us which catalog services to query for a given logical name.

```java
IndexQuery indexQuery = new SimpleIndexQuery(
    SimpleIndexQuery.queryMappingsByLogicalName,
    "my-logical-name-123",
    null);
```

1.4. Query the index service

RLS index services keep an index of logical names for each catalog service that sends its index to the given index service. By querying an index service, the client can find out which catalog services may have replica locations for a desired logical name.

```java
try {
    ReplicaLocationIndex index = connection.index();
    Results results = index.query(indexQuery);
    if (results.getRC() == RLSStatusCode.RLS_SUCCESS) {
        List batch = results.getBatch();
        Iterator i = batch.iterator();
        while (i.hasNext()) {
            IndexMappingResult result = (IndexMappingResult) i.next();
            if (result.getRC() != RLSStatusCode.RLS_SUCCESS)
                continue;
            String logicalName = result.getLogical();
            String catalogURL = result.getCatalog();
            // At this point, the client will need to create
            // a CatalogQuery object for each distinct catalog
            // URL returned in the results. These URLs indicate
            // the catalog services which have replica locations
            // for the given logical name.
        }
    }
} catch (RLSException e) {
    // handle exception
}
```

1.5. Create a simple catalog query

Based on the results of the index query, a client will create catalog queries.

```java
CatalogQuery catalogQuery = new SimpleCatalogQuery(
    SimpleCatalogQuery.queryMappingsByLogicalName,
    "my-logical-name-123",
```
1.6. Query the catalog service

RLS catalog services keep a catalog of logical names mapped to target names. The target names are typically used to indicate the URL for a data object (e.g., a gsiftp, http, etc. URL).

```java
try {
    LocalReplicaCatalog catalog = connection.catalog();
    Results results = catalog.query(catalogQuery);
    if (results.getRC() == RLSStatusCode.RLS_SUCCESS) {
        List batch = results.getBatch();
        Iterator i = batch.iterator();
        while (i.hasNext()) {
            MappingResult result = (MappingResult) i.next();
            if (result.getRC() != RLSStatusCode.RLS_SUCCESS)
                continue;
            String logicalName = result.getLogical();
            String targetName = result.getTarget();
            // At this point, the client has resolved the
            // target name for the given logical name. Keep in
            // mind that in the RLS, a logical name may be
            // mapped to multiple target names.
        }
    }
} catch (RLSException e) {
    // handle exception
}
```

2. Example Code

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_gcc32dbgthr`.

For C language example code\(^1\), click here\(^2\).

Developing in Java

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

---

\(^1\) test.c
\(^2\) test.c
1. Include the RLS Jar, $GLOBUS_LOCATION/lib/globus_rls_client.jar, in the CLASSPATH.

2. Import the RLS Package org.globus.replica.rls.*.

For Java language example code³, click here⁴. Note that the examples in this section use the older, deprecated API.
Chapter 3. Tutorials

There are no tutorials available at this time.
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.

---

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 commons-logging and cog-jglobus libraries which typically located in the $GLOBUS_LOCATION/lib/common folder. The RLS Java Client jar is named globus_rls_client.jar and is typically installed in the $GLOBUS_LOCATION/lib folder.

2. Component API

- RLS Client C API Documentation
- RLS Client Java API Documentation

1 http://www.isi.edu/~annr/rls/doc/client/index.html
2 http://www.isi.edu/~schuler/globus/4.2/doc/index.html
Chapter 6. Command line tools
Command line tools

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 ]
## Options

### Table 6.1. Options for globus-rls-admin

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>Sets &quot;clearvalues&quot; flag when deleting an attribute (will remove any attribute value records when an attribute is deleted).</td>
</tr>
<tr>
<td>-h</td>
<td>Shows usage.</td>
</tr>
<tr>
<td>-l reslimit</td>
<td>Sets an incremental limit on the number of results returned by a wildcard query at a time. Note that all results will be returned by the client. This parameter only limits the number of results incrementally retrieved 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.</td>
</tr>
<tr>
<td>-s</td>
<td>Uses SQL style wildcards (% and _).</td>
</tr>
<tr>
<td>-t timeout</td>
<td>Sets timeout (in seconds) for RLS server requests. The default is 30 seconds.</td>
</tr>
<tr>
<td>-u</td>
<td>Uses Unix style wildcards (* and ?).</td>
</tr>
<tr>
<td>-v</td>
<td>Shows version.</td>
</tr>
</tbody>
</table>
Commands
### Table 6.3. Commands for globus-rls-cli

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>query wildcard lrc lfn &lt;lfn-pattern&gt;</code></td>
<td>Performs a wildcarded query of an LRC server for mappings of <code>lfn-pattern</code>. Patterns use the standard Unix wildcard characters: an asterisk (*) matches 0 or more characters, and a question mark (?) matches any single character.</td>
</tr>
<tr>
<td><code>query wildcard lrc pf n &lt;pf n-pattern&gt;</code></td>
<td>Queries an LRC server for mappings to <code>pf n-pattern</code>. Patterns use the standard Unix wildcard characters: an asterisk (*) matches 0 or more characters, and a question mark (?) matches any single character.</td>
</tr>
<tr>
<td><code>query wildcard rli lfn &lt;lfn-pattern&gt;</code></td>
<td>Queries an RLI server for mappings of <code>lfn-pattern</code>. Patterns use the standard Unix wildcard characters: an asterisk (*) matches 0 or more characters, and a question mark (?) matches any single character.</td>
</tr>
<tr>
<td><code>set reslimit &lt;limit&gt;</code></td>
<td>Sets an incremental limit on the number of results returned by a wildcard query at a time. Note that all results will be returned by the client. This parameter only limits the number of results incrementally retrieved 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.</td>
</tr>
<tr>
<td><code>set timeout &lt;timeout&gt;</code></td>
<td>Sets the timeout (in seconds) on calls to the RLS server. The default value is 30.</td>
</tr>
<tr>
<td><code>version</code></td>
<td>Shows the version and exits.</td>
</tr>
</tbody>
</table>
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 File Name (LFN) to Physical File Name (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


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:
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`
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:
### Table 6.4. Options for globus-rls-server

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-b maxbackoff</code></td>
<td>Maximum time (in seconds) that <code>globus-rls-server</code> will attempt to reopen the socket it listens on after an I/O error.</td>
</tr>
<tr>
<td><code>-C rlscertfile</code></td>
<td>Name of the X.509 certificate file that identifies the server; sets environment variable <code>X509_USER_CERT</code>.</td>
</tr>
<tr>
<td><code>-c conffile</code></td>
<td>Name of the configuration file for the server. The default is <code>$GLOBUS_LOCATION/etc/globus-rls-server.conf</code> if the environment variable <code>GLOBUS_LOCATION</code> is set; else, <code>/usr/local/etc/globus-rls-server.conf</code>.</td>
</tr>
<tr>
<td><code>-d</code></td>
<td>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 <code>-L</code> option) to higher values.</td>
</tr>
<tr>
<td><code>-e rli_expire_int</code></td>
<td>Interval (seconds) at which an RLI server should expire stale entries.</td>
</tr>
<tr>
<td><code>-F lrc_update_factor</code></td>
<td>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 <code>lrc_update_factor</code>.</td>
</tr>
<tr>
<td><code>-f maxfreethreads</code></td>
<td>Maximum number of idle threads the server will leave running. Excess threads are terminated.</td>
</tr>
<tr>
<td>`-I true</td>
<td>false`</td>
</tr>
<tr>
<td><code>-i idletimeout</code></td>
<td>Seconds after which idle client connections are timed out.</td>
</tr>
<tr>
<td><code>-K rlskeyfile</code></td>
<td>Name of the X.509 key file. Sets environment variable <code>X509_USER_KEY</code>.</td>
</tr>
<tr>
<td><code>-L loglevel</code></td>
<td>Sets the log level. By default this is 0, which means only errors will be logged. Higher values mean more verbose logging.</td>
</tr>
<tr>
<td>`-l true</td>
<td>false`</td>
</tr>
<tr>
<td><code>-M maxconnections</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>-m maxthreads</code></td>
<td>Maximum number of threads server will start up to support simultaneous requests.</td>
</tr>
<tr>
<td><code>-N</code></td>
<td>Disables authentication checking. This option is intended for debugging. Clients should use the URL <code>RLSN://host</code> to disable authentication on the client side.</td>
</tr>
<tr>
<td><code>-o lrc_buffer_time</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>-p pidfiledir</code></td>
<td>Directory where PID files should be written.</td>
</tr>
</tbody>
</table>
### Command line tools

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-r</code></td>
<td>Configures whether the server is an RLI server.</td>
</tr>
<tr>
<td><code>-S rli_expire_stale</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>-s startthreads</code></td>
<td>Number of threads to start up initially.</td>
</tr>
<tr>
<td><code>-t timeout</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>-U myurl</code></td>
<td>URL for this server.</td>
</tr>
<tr>
<td><code>-u lrc_update_ll</code></td>
<td>Interval (in seconds) between lfn-list LRC to RLI updates.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Shows version and exits.</td>
</tr>
</tbody>
</table>
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.
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-aggregatorsource.pl` that may be used as an Execution Aggregator Source by WS MDS. See GT 4.2.1 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-aggregatorsource.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 4.2.1 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)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| acl user: permission [permission] | 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. 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. 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. There may be multiple acl entries, with the first match found used to determine a user's privileges. [permission] is one or more of the following values:  
• lrc_read Allows client to read an LRC.  
• lrc_update Allows client to update an LRC.  
• rli_read Allows client to read an RLI.  
• 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. |
| authentication true|false | Enable or disable GSI authentication.  
The default value is true.  
If authentication is enabled (true), clients should use the URL schema rls: to connect to the server.  
If authentication is not enabled (false), clients should use the URL schema rlsn:.
| db_pwd password | Password to use to connect to the database server.  
The default value is changethis. |
| db_user databaseuser | Username to use to connect to database server.  
The default value is dbperson. |
| idletimeout seconds | Seconds after which idle connections close.  
The default value is 900. |
| loglevel N | Sets loglevel to N (default is 0). Higher levels mean more verbosity. |
### Configuring RLS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lrc_bloomfilter_num-hash N</code></td>
<td>Number of hash functions to use in Bloom filters. The default value is 3. Possible values are 1 through 8. 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. Note: The default values of 3 and 10 give a false positive rate of approximately 1%.</td>
</tr>
<tr>
<td><code>lrc_bloomfilter_ratio N</code></td>
<td>Sets ratio of bloom filter size (in bits) to number of LFNs 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. The default value is 10. Note: The default values of 3 and 10 give a false positive rate of approximately 1%.</td>
</tr>
<tr>
<td><code>lrc_buffer_time N</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>lrc_dbname</code></td>
<td>Name of LRC database. The default value is <code>lrcdb</code>.</td>
</tr>
<tr>
<td>`lrc_server true</td>
<td>false`</td>
</tr>
<tr>
<td><code>lrc_update_bf seconds</code></td>
<td>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. This can be much smaller than the interval between updates without using Bloom filters (<code>lrc_update_ll</code>). The default value is 300.</td>
</tr>
<tr>
<td><code>lrc_update_factor N</code></td>
<td>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.</td>
</tr>
<tr>
<td>`lrc_updateImmediate true</td>
<td>false`</td>
</tr>
<tr>
<td><code>lrc_update_ll seconds</code></td>
<td>Number of seconds before an LRC server does an LFN list soft state update. The default value is 86400.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrc_update_retry seconds</td>
<td>Maximum seconds to wait before re-trying listen in the event of an I/O error.</td>
</tr>
<tr>
<td></td>
<td>The default value is 300.</td>
</tr>
<tr>
<td>maxbackoff seconds</td>
<td>Maximum number of idle threads. Excess threads are killed.</td>
</tr>
<tr>
<td></td>
<td>The default value is 5.</td>
</tr>
<tr>
<td>maxfreethreads N</td>
<td>Maximum number of simultaneous connections.</td>
</tr>
<tr>
<td></td>
<td>The default value is 100.</td>
</tr>
<tr>
<td>maxconnections N</td>
<td>Maximum number of simultaneous connections.</td>
</tr>
<tr>
<td></td>
<td>The default value is 100.</td>
</tr>
<tr>
<td>maxthreads N</td>
<td>Maximum number of threads running at one time.</td>
</tr>
<tr>
<td></td>
<td>The default value is 30.</td>
</tr>
<tr>
<td>myurl URL</td>
<td>URL of server.</td>
</tr>
<tr>
<td></td>
<td>The default value is rls://&lt;hostname&gt;:port</td>
</tr>
<tr>
<td>odbcini filename</td>
<td>Sets environment variable ODBCINI.</td>
</tr>
<tr>
<td></td>
<td>If not specified, and ODBCINI is not already set, then the default value is</td>
</tr>
<tr>
<td></td>
<td>$GLOBUS_LOCATION/var/odbc.ini.</td>
</tr>
<tr>
<td>pidfile filename</td>
<td>Filename where pid file should be written.</td>
</tr>
<tr>
<td></td>
<td>The default value is $GLOBUS_LOCATION/var/&lt;programname&gt;.pid.</td>
</tr>
<tr>
<td>port N</td>
<td>Port the server listens on.</td>
</tr>
<tr>
<td></td>
<td>The default value is 39281.</td>
</tr>
<tr>
<td>result_limit limit</td>
<td>Sets the maximum number of results returned by a query.</td>
</tr>
<tr>
<td></td>
<td>The default value is 0 (zero), which means no limit.</td>
</tr>
<tr>
<td></td>
<td>If a query request includes a limit greater than this value, an error (GLO-</td>
</tr>
<tr>
<td></td>
<td>BUS_RLS_BADARG) is returned.</td>
</tr>
<tr>
<td></td>
<td>If the query request has no limit specified, then at most result_limit re-</td>
</tr>
<tr>
<td></td>
<td>cords are returned by a query.</td>
</tr>
<tr>
<td>rli_bloomfilter true</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>If true, then only Bloom filter updates are accepted from LRCs.</td>
</tr>
<tr>
<td></td>
<td>If false, full LFN lists are accepted.</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> If Bloom filters are enabled, then the RLI does <em>not</em> support wildcarded queries.</td>
</tr>
</tbody>
</table>
Confusing RLS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rli_bloomfilter_dir</td>
<td>If an RLI is configured to accept bloom filters (rli_bloomfilter true), then Bloom filters may be saved to this directory after updates. This directory is scanned when an RLI server starts up and is used to initialize Bloom filters for each LRC that updated the RLI. 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. 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. Any other string is used as the directory name unchanged. 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 &quot;bf&quot; appended.</td>
</tr>
<tr>
<td></td>
<td>• none</td>
</tr>
<tr>
<td></td>
<td>Bloom filters are not saved to disk. This is the default.</td>
</tr>
<tr>
<td></td>
<td>• default</td>
</tr>
<tr>
<td></td>
<td>Bloom filters are saved to the default directory:</td>
</tr>
<tr>
<td></td>
<td>• $GLOBUS_LOCATION/var/rls-bloomfilters if GLOBUS_LOCATION is set</td>
</tr>
<tr>
<td></td>
<td>• else, /tmp/rls-bloomfilters</td>
</tr>
<tr>
<td></td>
<td>• pathname</td>
</tr>
<tr>
<td></td>
<td>Bloom filters are saved to the named directory. Any other string is used as the directory name unchanged.</td>
</tr>
<tr>
<td></td>
<td>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 &quot;bf&quot; appended.</td>
</tr>
<tr>
<td>rli dbname</td>
<td>Name of the RLI database. The default value is rlidb.</td>
</tr>
<tr>
<td>rli_expire_int</td>
<td>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. The default value is 28800.</td>
</tr>
<tr>
<td>rli_expire_stale</td>
<td>Interval (in seconds) after which entries in the RLI database are considered stale (presumably because they were deleted in the LRC). The default value is 86400. This value should be no smaller than lrc_update_11. Stale RLI entries are not returned in queries. Note: If the LRC server is responding, this value is not used. Instead the value of lrc_update_11 or lrc_update_bf is retrieved from the LRC server, multiplied by 1.2, and used as the value for this option.</td>
</tr>
</tbody>
</table>
### Configuring RLS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rli_server</td>
<td>If an RLI server, the value should be true. The default value is false.</td>
</tr>
<tr>
<td>rlscertfile</td>
<td>Name of the X.509 certificate file identifying the server. This value is set by setting environment variable X509_USER_CERT.</td>
</tr>
<tr>
<td>rlskeyfile</td>
<td>Name of the X.509 key file for the server. This value is set by setting environment variable X509_USER_KEY.</td>
</tr>
<tr>
<td>startthreads</td>
<td>Number of threads to start initially. The default value is 3.</td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout (in seconds) for calls to other RLS servers (e.g., for LRC calls to send an update to an RLI).</td>
</tr>
</tbody>
</table>
Chapter 8. Debugging

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Definition</th>
<th>Possible Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error with credential: The proxy credential: &lt;credential&gt; with subject: &lt;subject&gt; expired &lt;minutes&gt; minutes ago</td>
<td>Expired proxy credential</td>
<td>Create a new proxy with grid-proxy-init.</td>
</tr>
</tbody>
</table>
| Unable to connect to local-host: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. | • 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” | At times, a client may experience a connection timeout when interacting with the RLS server due to a variety of reasons:  
• 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. | If timeouts are experienced with increasing frequency, increase the RLS server’s timeout configuration parameter found in the $GLOBUS_LOCATION/var/globus-rls-server.conf file. You may also use the –t timeout option of the globus-rls-cli tool. |

\(^1\) [http://www.globus.org/toolkit/data/rls/rls_faq.html](http://www.globus.org/toolkit/data/rls/rls_faq.html)
Chapter 10. Related Documentation

For additional details, see the RPC Protocol Description¹.

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.
Index

A
administrative operations
configuring LRC server to stop updating RLI, 11
configuring LRC-to-RLI updates
  compressed (Bloom filters), 11
  uncompressed, 11
configuring settings (runtime only), 11
pinging the server, 11
stopping RLS server, 11
architecture
  for admin, 8

C
client operations
attributes
  adding an attribute to an lfn or pfn, 13
  bulk adding attribute values, 13
  bulk deleting attribute values, 13
  bulk querying attributes, 13
  defining a new attribute, 13
  deleting an attribute definition, 13
  modifying the value of an attribute, 13
  removing an attribute, 13
  retrieving the value of the specified attribute for an
    lfn or pfn, 13
  searching for lfns or pfns which have the specified
    matching operator and value, 13
  showing an attribute definition, 13
basic
  adding physical filenames to mappings of logical
    filenames in a LocalReplicaCatalog, 13
  creating a new lfn, pfn mapping in an LRC catalog,
    13
  deleting a lfn, pfn mapping from an LRC catalog,
    13
  exiting the interactive session, 13
bulk
  bulk adding lfn, pfn mappings, 13
  bulk creating lfn, pfn mappings, 13
  bulk deleting lfn, pfn mappings, 13
  bulk querying the LRC for lfns, 13
  bulk querying the LRC for pfns, 13
  bulk querying the RLI for lfns, 13
querying
  performing a wildcased query of an LRC server
    for mappings of lfn-pattern, 13
  querying an LRC server for mappings of lfn, 13
  querying an LRC server for mappings to lfn-pattern,
    13
  querying an RLI server for mappings of lfn, 13
  querying an RLI server for mappings to lfn-pattern,
    13
  using interactive mode, 13

D
debugging, 32

E
errors, 33

S
server operations, 17
  configuring RLS server as Location Replica Catalog
    (LRC), 17
  configuring RLS server as Replica Location Index
    (RLI), 17