

GT 4.2.1 Reliable File Transfer (RFT) Service: Developer's Guide

GT 4.2.1 Reliable File Transfer (RFT) Service: Developer's Guide

Introduction

RFT Service implementation in GT 4.2.1 uses standard SOAP messages over HTTP to submit and manage a set of 3rd party GridFTP transfers and to delete files using GridFTP. The user creates a RFT resource by submitting a list of URL pairs of files that need to be transferred/deleted to RFT Factory service. The user also specifies the time to live for the resource the user is creating to a GT 4.2.1 container in which RFT is deployed and configured. The resource is created after the user is properly authorized and authenticated. RFT service implementation exposes operations to control and manage the transfers (the resource). The operations exposed by both RFT factory and RFT service are briefly described below. The resource the user created also exposes the state of the transfer as a resource property to which the user can either subscribe for changes or poll for the changes in state periodically using standard command line clients.

Table of Contents

How-to Index	5
1. Before you begin	1
1. RFT Feature Summary	1
2. Tested platforms	2
3. Backward compatibility summary	2
4. Technology dependencies	2
5. Reliable Transfer Service (RFT) Security Considerations	3
2. Usage scenarios	4
1. Transferring large datasets using GridFTP	4
2. Deleting a set of files and directories using GridFTP	6
3. Tutorials	8
4. Architecture and design overview	9
5. APIs	10
1. Programming Model Overview	10
2. Component API	10
6. Services and WSDL	11
1. Protocol overview	11
2. Operations	11
3. RFT Resource Properties	12
4. Faults	13
5. WSDL and Schema Definition	13
I. RFT Commands	14
rft	15
globus-crft	17
rft-delete	19
7. RFT transfer request	20
1. Request Schema	20
2. Request Options	20
8. Configuring RFT	23
1. Configuration overview	23
2. Syntax of the interface	23
3. Optional configuration: configuring the PostgreSQL database	23
4. Optional configuration: configuring the MySQL database	24
5. RFT auto-registration with default WS MDS Index Service	25
6. Registering RFT manually with default WS MDS Index Service	26
9. Environment variable interface	27
1. Environmental variables for RFT	27
10. Debugging	28
1. Development Logging in Java WS Core	28
2. Enable Debug Logging in RFT	28
11. Troubleshooting	30
1. Errors	30
2. RFT fault-tolerance and recovery	30
12. Related Documentation	32
Index	33

List of Tables

11.1. Reliable File Transfer (RFT) Errors	30
---	----

How-to Index

A

api,
architecture,

C

commands
 globus-crft,
 rft,
 rft-delete,
compatibility,
configuration interface,
 auto-registration with WS MDS Index,
 overview,
 resource properties,
 settings,
configuring,
 auto-registration to WS MDS Index Service,
 MySQL database,
 PostgreSQL database,
 registering manually to WS MDS Index Service,
 resource properties,

D

debugging
 (for developers),
 logging,
deleting a set of files and directories (with GridFTP),
deleting files (from a list),
dependencies,

E

environmental variables,
errors,

F

fault tolerance,
features,

L

logging
 debugging,

P

papers,
platforms, tested,

R

recovery,

resource properties,

S

security considerations,
services,
submitting a third party GridFTP transfer,
submitting transfers (from a list)
 C client,

T

transfer request,
 options,
 schema,
transferring large datasets (with GridFTP),
troubleshooting
 (for developers),

U

usage scenarios,

W

wSDL,

Chapter 1. Before you begin

1. RFT Feature Summary

Features new in GT 4.2.1

Supported Features

- Delete files: Delete a set of files/directories on a GridFTP server.
- Exponential Backoff: Configurable exponential back off before a failed transfer is retried.
- Transfer All or None: If this option is set and one of the transfers in the request fails, RFT will stop transferring the remainder of the request and delete the files that were already transferred successfully.
- Transfer Permissions: File permissions are restored at the destination once the file is transferred successfully. This can be configured to throw a fatal error or a transient error depending on whether the GridFTP server supports the MLST command.
- Configurable number of concurrent transfers per container and per request.
- Better error reporting and faults.
- Database purge of the request and transfers after life time expiration.
- Cumulative (aggregate) Resource Properties on the factory provide some statistical information.
- One status Resource Property for the entire transfer.
- Recursive directory transfers and deletes.
- Parallel streams.
- TCP Buffer Size.
- Third-party directory transfers, file transfers and deletes.
- Data channel authentication (DCAU).
- NoTPT option.
- Different subject names for source and destination GridFTP servers for the authorization mechanism.
- Support for binary/ASCII type of transfers.
- Configurable number of retries for failed transfers per request.
- Block Size in bytes.

Deprecated Features

- None

2. Tested platforms

Tested platforms for RFT:

- Linux
 - Fedora Core 1 i686
 - Fedora Core 3 i686
 - RedHat 7.3 i686
 - RedHat 9 x86
 - Debian Sarge x86
 - Debian 3.1 i686
- Mac OS X
 - Mac OS X 10.3, 10.4

Tested containers for RFT:

- Java WS Core container
- Tomcat 5.0.30

3. Backward compatibility summary

Protocol changes since GT 4.0.x

- Added All or None option, maximum attempts, and finishBy to the transfer request
- Not backwards compatible with the OGSi version

API changes since GT 4.0.x

- None

Exception changes since GT 4.0.x

- None

Schema changes since GT 4.0.x

- WSDL changes to work with the new Java WS Core

4. Technology dependencies

RFT depends on the following GT components:

- Java WS Core
- WS Authentication and Authorization

- Delegation Service
- Service Groups
- MDS useful RP

RFT depends on the following 3rd party software:

- Optional : PostgreSQL 7.1 or later. Not tested with 8.0 yet.
- Optional : MySQL

5. Reliable Transfer Service (RFT) Security Considerations

5.1. Permissions of service configuration files

The service configuration files such as `jndi-config.xml` and `server-config.wsdd` (located under `etc/<gar>/` directory) contain private information such as database passwords and usernames. Ensure that these configuration files are only readable by the user that is running the container.

The deployment process automatically sets the permissions of `jndi-config.xml` and `server-config.wsdd` as user readable only. However, this might not work correctly on all platforms and this does not apply to any other configuration files.

5.2. Access of information stored in the database

RFT stores the transfer requests in a database. Proper security measures need to be taken to protect the access of the data by granting/revoking appropriate permissions on tables that are created for RFT use and other steps that are appropriate and consistent with site specific security measures.

5.3. Permissions of persistent data

RFT uses the subscription persistence API from the GT4 core to store all of its subscription data under the `~/ .globus/persisted` directory. Ensure that the entire `~/ .globus/persisted` directory is only readable by the user running the container.

Chapter 2. Usage scenarios

1. Transferring large datasets using GridFTP

RFT is primarily used to reliably transfer large datasets using GridFTP. If you are a developer and would like to use RFT, the following steps would help you to do that.

- Contact the Delegation Factory Service and get an EPR for the Delegation Resource that contains your delegated credential.

```
public static EndpointReferenceType
    delegateCredential(String host, String port) throws Exception {
    ClientSecurityDescriptor desc = new ClientSecurityDescriptor();
    // Credential to sign with, assuming default credential
    GlobusCredential credential = GlobusCredential.getDefaultCredential();
    desc.setGSITransport(Constants.GSI_TRANSPORT)
    Util.registerTransport();
    desc.setAuthz('host');

    String factoryUrl = PROTOCOL + "://" + host + ":"
        + port + SERVICE_URL_ROOT
        + DelegationConstants.FACTORY_PATH;

    // lifetime in seconds
    int lifetime = TERM_TIME * 60;

    // Get the public key to delegate on.
    EndpointReferenceType delegEpr = AddressingUtils
        .createEndpointReference(factoryUrl, null);
    X509Certificate[] certsToDelegateOn = DelegationUtil
        .getCertificateChainRP(delegEpr, desc);
    X509Certificate certToSign = certsToDelegateOn[0];
    return DelegationUtil.delegate(factoryUrl,
        credential, certToSign, lifetime, false,
        desc);
}
```

- Now construct a TransferRequestType Object:

```
TransferType[] transferArray = new TransferType[1];
transferArray[0] = new TransferType();
transferArray[0].setSourceUrl("gsiftp://foo/bar");
transferArray[0].setDestinationUrl("gsiftp://blah/");
RFTOptionsType rftOptions = new RFTOptionsType();
rftOptions.setBinary(true);
// You can set more options like parallel streams, buffer sizes etc
// Refer to Public Interface guide of RFT for more details
TransferRequestType request = new TransferRequestType();
```

```
request.setRftOptions(rftOptions);
request.setTransfer(transferArray);
request.setTransferCredentialEndpoint(delegateCredential(host, port));
```

- Now contact the RFT factory and create an RFT resource:

```
public static EndpointReferenceType createRFT(String rftFactoryAddress,
      BaseRequestType request)
throws Exception {
    endpoint = new URL(rftFactoryAddress);
    factoryPort = rftFactoryLocator
        .getReliableFileTransferFactoryPortTypePort(endpoint);
    CreateReliableFileTransferInputType input =
        new CreateReliableFileTransferInputType();
    //input.setTransferJob(transferType);
    if(request instanceof TransferRequestType) {
        input.setTransferRequest((TransferRequestType)request);
    } else {
        input.setDeleteRequest((DeleteRequestType)request);
    }
    Calendar termTime = Calendar.getInstance();
    termTime.add(Calendar.HOUR, 1);
    input.setInitialTerminationTime(termTime);
    setSecurity((Stub)factoryPort);
    CreateReliableFileTransferOutputType response = factoryPort
        .createReliableFileTransfer(input);

    return response.getReliableTransferEPR();
}
```

- Now contact the RFT service Implementation and call start to actually start the transfer:

```
ReliableFileTransferPortType rft = rftLocator
    .getReliableFileTransferPortTypePort(rftepr);
setSecurity((Stub)rft);

//For secure notifications
subscribe(rft);
System.out.println("Subscribed for overall status");
//End subscription code
Calendar termTime = Calendar.getInstance();
termTime.add(Calendar.MINUTE, TERM_TIME);
SetTerminationTime reqTermTime = new SetTerminationTime();
reqTermTime.setRequestedTerminationTime(termTime);
System.out.println("Termination time to set: " + TERM_TIME
    + " minutes");
SetTerminationTimeResponse termRes = rft
    .setTerminationTime(reqTermTime);
StartOutputType startresp = rft.start(new Start());
```

2. Deleting a set of files and directories using GridFTP

RFT can also be used to delete a set of files and directories using GridFTP server. The following steps depict how to:

- Contact the Delegation Factory Service and get an EPR for the Delegation Resource that contains your delegated credential.

```
public static EndpointReferenceType
    delegateCredential(String host, String port) throws Exception {
    ClientSecurityDescriptor desc = new ClientSecurityDescriptor();
    // Credential to sign with, assuming default credential
    GlobusCredential credential = GlobusCredential.getDefaultCredential();
    desc.setGSITransport(Constants.GSI_TRANSPORT)
    Util.registerTransport();
    desc.setAuthz('host');

    String factoryUrl = PROTOCOL + "://" + host + ":"
        + port + SERVICE_URL_ROOT
        + DelegationConstants.FACTORY_PATH;

    // lifetime in seconds
    int lifetime = TERM_TIME * 60;

    // Get the public key to delegate on.
    EndpointReferenceType delegEpr = AddressingUtils
        .createEndpointReference(factoryUrl, null);
    X509Certificate[] certsToDelegateOn = DelegationUtil
        .getCertificateChainRP(delegEpr, desc);
    X509Certificate certToSign = certsToDelegateOn[0];
    return DelegationUtil.delegate(factoryUrl,
        credential, certToSign, lifetime, false,
        desc);
}
```

- Now construct a DeleteRequestType object:

```
DeleteType[] deleteArray = new DeleteType[1];
deleteArray[0] = new DeleteType();
deleteArray[0].setFile("gsiftp://foo/bar");
DeleteOptionsType deleteOptions = new DeleteOptionsType();
deleteOptions.setSubjectName("SUBJECT-NAME");
DeleteRequestType request = new DeleteRequestType();
request.setDeleteOptions(deleteOptions);
request.setDeletion(deleteArray);
request.setTransferCredentialEndpoint(delegateCredential(host, port));
```

- Now contact the RFT factory and create an RFT resource:

```
public static EndpointReferenceType createRFT(String rftFactoryAddress,
      BaseRequestType request)
throws Exception {
    endpoint = new URL(rftFactoryAddress);
    factoryPort = rftFactoryLocator
        .getReliableFileTransferFactoryPortTypePort(endpoint);
    CreateReliableFileTransferInputType input =
        new CreateReliableFileTransferInputType();
    //input.setTransferJob(transferType);
    if(request instanceof TransferRequestType) {
        input.setTransferRequest((TransferRequestType)request);
    } else {
        input.setDeleteRequest((DeleteRequestType)request);
    }
    Calendar termTime = Calendar.getInstance();
    termTime.add(Calendar.HOUR, 1);
    input.setInitialTerminationTime(termTime);
    setSecurity((Stub)factoryPort);
    CreateReliableFileTransferOutputType response = factoryPort
        .createReliableFileTransfer(input);

    return response.getReliableTransferEPR();
}
```

- Now contact the RFT service Implementation and call start to actually start the transfer:

```
ReliableFileTransferPortType rft = rftLocator
    .getReliableFileTransferPortTypePort(rftepr);
setSecurity((Stub)rft);

//For secure notifications
subscribe(rft);
System.out.println("Subscribed for overall status");
//End subscription code
Calendar termTime = Calendar.getInstance();
termTime.add(Calendar.MINUTE, TERM_TIME);
SetTerminationTime reqTermTime = new SetTerminationTime();
reqTermTime.setRequestedTerminationTime(termTime);
System.out.println("Termination time to set: " + TERM_TIME
    + " minutes");
SetTerminationTimeResponse termRes = rft
    .setTerminationTime(reqTermTime);
StartOutputType startresp = rft.start(new Start());
```

Chapter 3. Tutorials

There are no tutorials available at this point.

Chapter 4. Architecture and design overview

A design doc can be found [here](#)¹.

¹ ../Protocol_overview.doc

Chapter 5. APIs

1. Programming Model Overview

The Reliable Transfer Service (RFT) is a WSRF based service that provides interfaces for controlling and monitoring third party file transfers using GridFTP servers. The client controlling the transfers (in this case RFT) is hosted inside of a Grid service so it can be managed using the soft state model. It is essentially a reliable and recoverable version of the GT2 `globus-url-copy` tool and more. In GT 4.2.1, RFT can also perform file deletion and recursive directory deletion operations. It is also used by GRAM to perform all the staging operations and cleanup operations.

2. Component API

Some relevant APIs:

- [Service API](#)¹
- [Common API](#)²
- [Client API](#)³

¹ http://www.globus.org/api/javadoc-4.2.1/globus_wsrf_rft_service_java/

² http://www.globus.org/api/javadoc-4.2.1/globus_wsrf_rft_common_java/

³ http://www.globus.org/api/javadoc-4.2.1/globus_wsrf_rft_client_java/

Chapter 6. Services and WSDL

1. Protocol overview

The RFT service implementation in GT 4.2.1 uses standard SOAP messages over HTTP to submit and manage a set of 3rd party GridFTP transfers and to delete files using GridFTP. The user creates an RFT resource by submitting a list of URL pairs of files that need to be transferred/deleted to the RFT Factory service. The user also specifies the time to live for the resource the user is creating to the GT 4.2.1 container in which RFT is deployed and configured. The resource is created after the user is properly authorized and authenticated. RFT service implementation exposes operations to control and manages the transfers (the resource). The operations exposed by both the RFT factory and the RFT service are briefly described below. The resource the user created also exposes the state of the transfer as a resource property to which the user can either subscribe for changes or poll for the changes in state periodically using standard command line clients.

2. Operations

Please find below operations of both RFT Factory and RFT Service Implementation.

2.1. RFT Factory Service

Used to create a Reliable File Transfer resource. The operations exposed by the factory are as follows:

- `createReliableFileTransfer`: Creates a Reliable File Transfer resource.
 - Input Parameters: Initial Termination time, Transfer Request or Delete Request.
 - Output parameters: Termination time, Current time, Endpoint reference of the Resource created. This should be stored by the user, as it is needed to query the status of the resource and to perform any further operations on the resource.
 - Fault: `createReliableFileTransferFault`.

2.2. RFT Service

Used to manage the Resource created using the RFT Factory Service. The operations exposed by the service are as follows:

- `start`: Starts executing the transfers/deletes.
 - Input Parameters: None
 - Output Parameters: None
 - Fault: `RepeatedlyStartedFault`
- `getStatus`: To get the status of a particular file.
 - Input Parameters: A source URL of the file that is part of the request.
 - Output Parameters: `Transfer Status Type`
 - Fault: `RFTDatabaseFault`

- `getStatusSet`: To get the status of a set of files in a request.
 - Input Parameters: `int` from (the relative position of the transfer in the request) and `int` offset (the number of files queried).
 - Output Parameters: An array of `TransferStatusType`.
 - Fault: `RFTDatabaseFault`
- `cancel`: To cancel a transfer that is part of a resource.
 - Input Parameters: `int` from (the relative position of the transfer in the request) and `int` to.
 - Output Parameters: None
 - Fault: `RFTDatabaseFault`

3. RFT Resource Properties

The resource properties of RFT Factory (which acts both as a resource and a service at the same time) and RFT Resource are found below:

3.1. RFT Factory Resource Properties

- `ActiveResourceInstances`: A dynamic resource property of the total number of active RFT resources in the container at a given point of time.
- `TotalNumberOfTransfers`: A dynamic resource property of the total number of transfers/deletes performed since the RFT service was deployed in this container.
- `TotalNumberOfActiveTransfers`: A dynamic resource property of the number of active transfers across all rft resources in a container at a given point of time.
- `TotalNumberOfBytesTransferred`: A dynamic resource property of the total number of bytes transferred by all RFT resources created since the deployment of the service.
- `RFTFactoryStartTime`: Time when the service was deployed in the container. Used to calculate uptime.
- `DelegationServiceEPR`: The end point reference of the Delegation resource that holds the delegated credential used in executing the resource.

3.2. RFT Resource Properties

- `OverallStatus`: This is a complex type providing the overall status of an RFT resource by providing the number of transfers pending, active, finished, retrying, failed, and cancelled. Each of these values can be obtained by invoking `getTransfers(Finished/Active/Failed/Restarted/Pending/Cancelled)` on `OverallStatus` Resource Property. Note that this Resource Property gets updated every time one of the transfers changes state, so there can be and will be more than one update in the life time of a RFT resource if you subscribe to this RP. This Resource Property also includes the last fault (if thrown) from a transfer and can be accessed by invoking `getFault` on `OverallStatus`. This will indicate why a transfer has failed.
- `RequestStatus`: This is a complex type resource property providing the status of an RFT resource in the form of `Pending/Active/Done/Failed`. The status can be obtained from `RequestStatusType` by invoking `getRequestStatus()`. This will result in one of four status strings (`Pending/Active/Done/Failed/Cancelled`). This RP also contains a fault

that denotes the last fault in a RFT resource and can be accessed by invoking `getFault()`. If a client is subscribed to this RP, there will be only be 2 updates in the life time of an RFT resource (Pending->Active->Done, Pending->Active->Failed, Pending->Active->Cancelled, and Pending->Cancelled).

- `TotalBytes`: This provides the total number of bytes transferred by the resource.
- `TotalTime`: This provides the total time taken to transfer the above-mentioned total bytes.

4. Faults

Faults from the RFT Factory Service and RFT Service can be found below:

4.1. RFT Factory Service

- `createReliableFileTransferFault`: All the errors encountered during the creation of the RFT resource are mapped to this fault. Any security related errors are caught before the factory and are thrown to the user/client.

4.2. RFT Service

- `RepeatedlyStartedFault`: This is raised if a client calls start more than once on a resource.
- `RFTDatabaseFault`: This is thrown when the service is unable to find the resource the user/client is querying for.

5. WSDL and Schema Definition

- [Reliable Transfer Factory Port Type](#)¹
- [Reliable Transfer Port Type](#)²

You can find links to all the RFT schemas [here](#)³.

¹ http://viewcvs.globus.org/viewcvs.cgi/ws-transfer/reliable/common/schema/transfer/reliable/reliable_transfer_factory_port_type.wsdl?rev=1.15&only_with_tag=globus_4_0_0&content-type=text/vnd.viewcvs-markup

² http://viewcvs.globus.org/viewcvs.cgi/ws-transfer/reliable/common/schema/transfer/reliable/reliable_transfer_port_type.wsdl?rev=1.14&only_with_tag=globus_4_0_0&content-type=text/vnd.viewcvs-markup

³ <http://viewcvs.globus.org/viewcvs.cgi/ws-transfer/reliable/common/schema/transfer/reliable/>

RFT Commands

Name

rft -- Submit and monitor a 3rd party GridFTP transfer

rft

Tool description

Submits a transfer to the Reliable File Transfer Service and prints out the status of the transfer on the console.

Command syntax and options

```
rft [-h <host-ip of the container defaults to localhost>
-r <port, defaults to 8080>
-l <lifetime for the resource default 60mins>
-m <security mechanism. 'msg' for secure message or 'conv' for
  secure conversation and 'trans' for transport. Defaults to
  secure transport.>
-p <protection type, 'sig' signature and 'enc' encryption,
  defaults to signature >
-z <authorization mechanism can be self or host. default self>
-file <file to write EPR of created Reliable File Transfer Resource]>
-f <path to the file that contains list of transfers>
```

This is a sample transfer file that the command-line client will be able to parse. It can also be found in **\$GLOBUS_LOCATION/share/globus_wsrf_rft_client/** along with other samples for directory transfers and deletes (lines starting with # are comments):

```
This option when it is set to true means to perform transfer in binary
form, if it is set to false transfer is done in ASCII. Default is binary.
true
```

```
#Block size in bytes that is transferred. Default is 16000 bytes.
16000
```

```
#TCP Buffer size in bytes
```

```
#Specifies the size (in bytes) of the TCP buffer to be used by the underlying
ftp data channels. This is critical to good performance over the WAN. Use the
bandwidth-delay product as your buffer size.
```

```
16000
```

```
#Notpt (No thirdPartyTransfer): turns third-party transfers off is this option
is set to false (on if set to true).
```

```
Site firewall and/or software configuration may prevent a connection
between the two servers (a third party transfer). If this is the case,
RFT will "relay" the data. It will do a GET from the source and a PUT to
the destination. This obviously causes a performance penalty, but will allow
you to complete a transfer you otherwise could not do.
```

```
false
```

```
#Number of parallel streams: Specifies the number of parallel data connections
that should be used.
```

1

#Data Channel Authentication (DCAU): Turns off data channel authentication for FTP transfers is set to false.(the default is true to authenticate the data channel).

true

Concurrency of the request: Number of files that you want to transfer at any given point. Default is set to one.

1

#Grid Subject name of the source gridftp server. This is used for Authorization purposes. If the source gridftp server is running with host credentials you can specify "n /DC=org/DC=doe grids/OU=People/CN=Ravi Madduri 134710

#Grid Subject name of the destination gridftp server. This is used for Authorization purposes. If the destination gridftp server is running with host credentials you can specify "null" here. By default Host authorization is done. /DC=org/DC=doe grids/OU=People/CN=Ravi Madduri 134710

#Transfer all or none of the transfers: This option if set to true will make RFT to clean up (delete) all the transfers that have been done already if one of the transfers fails.

false

#Maximum number of retries: This is number of times RFT retries a transfer failed with a non-zero exit code.

10

#Source/Dest URL Pairs: gsiftp urls of source followed by destination.

If directory is to be recursively transferred the source gsiftp url and destination gsiftp url should end with "/". Currently RFT supports Directory - Directory, File - Directory, File - File transfers. There can be more URL pairs and all of them use the same options as above for performing the transfer.

gsiftp://localhost:5678/tmp/rftTest.tmp

gsiftp://localhost:5678/tmp/rftTest_Done.tmp

Limitations

This command line client is very simple and does not do any intelligent parsing of various command line options or of the options in the sample transfer file. It works fine if used in the way documented here. For more information on all these options please refer to the [documentation of globus-url-copy](#). Also, please note that the maximum number of transfers the command-line client can process before running out of memory is ~21K with the default JVM heap size, which was 64M in our tests. Please look at [Performance Reports](#)¹ for more details.

¹ ../rft_scalability_3_9_4.doc

Name

globus-crft -- Command-line client to transfer files using RFT

globus-crft

Tool description

This distribution contains a client to the RFT service written in C. RFT is the reliable transfer server. It allows clients to submit URL transfer requests to a persistent service which will perform the transfers on behalf of the client.

Options

- | | |
|---|--|
| <code>-a --all-or-none <on off></code> | Enable all or none transfer: default off. |
| <code>-con --concurrent <int></code> | The number of simultaneous transfers. |
| <code>-C --cancel</code> | Cancel a transfer. |
| <code>-c --create</code> | Create a new RFT service. |
| <code>-del --delete</code> | Delete a URL. |
| <code>-ds --destination-subject <subject></code> | The expected domain name of the destination GridFTP server. |
| <code>-d --destroy</code> | Destroy the server. If used with <code>-monitor</code> , wait until completion and then destroy. |
| <code>-D --done</code> | Return the current status of the transfer in the exit code: <ul style="list-style-type: none">• 0=Done• 1=Active• 2=Pending• 3=Cancelled• 4=Failed |
| <code>-ef --epr-file <path></code> | Path to the EPR file. If used with <code>--create</code> the EPR is written to this location. In all other cases the EPR is read from this location. |
| <code>-ez --easy</code> | Create, submit, and wait for the transfer to complete. The job is started with some standard options. |
| <code>-e --factory <contact></code> | The endpoint to contact when creating a server. Used with <code>--create</code> . |
| <code>-f --transfer-file <path></code> | A path to a file that contains the source destination URL pairs. |
| <code>-gS --getStatusSet <int> <int></code> | Get the status of all the transfer requests in the range. |
| <code>-g --getStatus <source url></code> | Get the status of the given source url. |

`-h` | `--help` Print usage information.

FIXME - finish converting to variable list:

`-ms` | `--message-security` <[sig] | [conv] | [trans]>
 Security mechanism. 'msg' for secure message,
 'conv' for secure conversation, 'trans' for
 transport. The default is trans.

`-m` | `--monitor` Wait for the service to complete, and receive
 status updates.

`-os` | `--getOverallStatus` Get the overall status.

`-p` | `--protection` <[sig] | [enc]>
 Protection type. 'sig' for signature, 'enc' for
 encryption. The default is 'sig'.

`-P` | `--parallel` <int> The number of parallel sockets to use with each
 transfer.

`-q` | `--quiet` Write no output.

`-rs` | `--getRequestStatus` Get the request status.

`-r` | `--retries` Number of retries

`-S` | `--subject` <subject> The expected domain name of both the source and
 destination GridFTP servers.

`-ss` | `--source-subject` <subject>
 The expected domain name of the source GridFTP
 server.

`-s` | `--submit` Start the RFT service

`-tb` | `--tcp-bs` <int> The TCP buffer size to use with each transfer.

`-ttl` | `--termination-time` <int>
 Set the lifetime of the service.

`-v` | `--version` Print version information.

`-vb` | `--verbose` Display much more output.

`-xi` | `--xml-input` <path> Read the request description from the given xml
 description.

`-xo` | `--xml-output` <path> Write the request description to the given file
 location in xml format.

`-z` | `--authz` <[self] | [host] | [id <subject>]>
 Authorization. 'self', 'host', or 'id <DN>'.

Limitations

No limitations exist with this command line tool.

Name

rft-delete -- Command-line client to delete files using RFT

rft-delete

Tool description

This command-line tool is used to submit a list of files to be deleted.

Command and options

```
rft-delete [-h <host-ip of the container default localhost>
-r <port, defaults to 8080>
-l <lifetime for the resource default 60mins>
-m <security mechanism. 'msg' for secure message or 'conv' for
  secure conversation and 'trans' for transport. Defaults to
  secure transport.>
-p <protection type, 'sig' signature and 'enc' encryption,
  defaults to signature >
-z <authorization mechanism can be self or host. default self>
-file <file to write EPR of created Reliable File Transfer Resource]>
-f <path to the file that contains list of transfers>
```

This is a sample file that the command line client will be able to parse, and it can also be found in **\$GLOBUS_LOCATION/share/globus_wsrf_rft_client/** along with other samples for directory transfers and deletes (lines starting with # are comments):

```
# Subject name (defaults to host subject)
  /DC=org/DC=doegrids/OU=People/CN=Ravi Madduri 134710
  gsiftp://localhost:5678/tmp/rftTest_Done.tmp
  gsiftp://localhost:5678/tmp/rftTest_Done1.tmp
```

Limitations

No limitations exist with this command line tool.

Chapter 7. RFT transfer request

1. Request Schema

Please go [here](#)¹ to view the entire RFT transfer request schema documentation.

2. Request Options

2.1. General Options

These options are set in the `transferRequest`² and `deleteRequest`³ elements and apply similarly for each.

- *concurrency*

This denotes number of files in the request that needs to be transferred at one time.

- *maxAttempts*

Maximum number of attempts after transient errors to execute the transfer or deletion before giving up and raising an error.

- *finishBy*

(Not Implemented) In future versions of RFT this will be used to enforce time constraints on a transfer.

2.2. Transfer Options

These options are set in the `rftOptions` element (see `RFTOptionsType`⁴ for more details) and are specific to file transfers. They can be specified as defaults for all transfers under the `transferRequest`⁵ element, and/or individually under each `transfer` element (see `TransferType`⁶ for more details):

```
<transferRequest>
  <transfer>...</transfer>
  <rftOptions>
    <!-- option elements here -->
  </rftOptions>
</transferRequest>
```

AND/OR

```
<transferRequest>
  <transfer>
```

¹ ../rft_job_description.html

² ../rft_job_description.html#element_transferRequest

³ ../rft_job_description.html#element_deleteRequest

⁴ ../rft_job_description.html#type_RFTOptionsType

⁵ ../rft_job_description.html#element_transferRequest

⁶ ../rft_job_description.html#type_TransferType

```
    ...
    <sourceUrl>
    <destinationUrl>
    ...
    <rftOptions>
      <!-- option elements here -->
    </rftOptions>
  </transfer>
</transferRequest>
```

- *binary*
Transfer as a binary file. Default is "true".
- *blockSize*
Specifies the size of the data blocks to use in the transfer.
- *tcpBufferSize*
Specifies the TCP buffer size used for the transfer.
- *notpt*
If set to "true", third-party transfer mode will not be use. Instead, a client thread will be started that will GET data from the source server and and PUT data to the destination server. Default is "false".
- *parallelStreams*
Specifies the number of parallel streams to use during the transfer. Default is 1.
- *dcau*
Specifies whether or not to use data channel authentication. Default is true.
- *subjectName*
Specifies the credential subject to use for authenticating both the source and destination servers.
- *destinationSubjectName*
Specifies the credential subject to use for authenticating the destination server.
- *sourceSubjectName*
Specifies the credential subject to use for authenticating the source server.
- *userName*
Specifies the username to be used to perform the transfer which sometimes may not be the same as transfer requester.

2.3. Deletion Options

These options are set in the `deleteOptions` element (see [DeleteOptionsType](#)⁷ for more details), and are specific to file deletions. They can be specified as defaults for all deletions under the `deleteRequest`⁸ element, and/or individually under each `deletion` element (see [DeleteType](#)⁹ for more details):

```
<deleteRequest>
  <deletion>...</deletion>
  <deleteOptions>
    <!-- option elements here -->
  </deleteOptions>
</deleteRequest>
```

AND/OR

```
<deleteRequest>
  <deletion>
    ...
    <file>
      <deleteOptions>
        <!-- option elements here -->
      </deleteOptions>
    </deletion>
</deleteRequest>
```

- *subjectName*

Specifies the credential subject to use for authenticating the target server.

- *userName*

Specifies the username to be used to perform the deletion.

⁷ ../rft_job_description.html#type_DeleteOptionsType

⁸ ../rft_job_description.html#element_deleteRequest

⁹ ../rft_job_description.html#type_DeleteType

Chapter 8. Configuring RFT

1. Configuration overview

RFT has the following prerequisites:

- [Java WS Core](#) - This is built and installed in a [Installing GT 4.2.1](#).
- A host certificate (see [Installing GT 4.2.1](#)).
- [GridFTP](#) - GridFTP performs the actual file transfer and is built and installed in a [Installing GT 4.2.1](#).
- PostgreSQL - PostgreSQL is used to store the state of the transfer to allow for restart after failures. The interface to PostgreSQL is JDBC, so any DBMS that supports JDBC can be used, although no others have been tested. For instructions on configuring the PostgreSQL database for RFT, see below. .

2. Syntax of the interface

The security of the service can be configured by modifying the [security descriptor](#). It allows for configuring the credentials that will be used by the service, type of authentication and authorization that needs to be enforced. By default, the following security configuration is installed:

- Credentials set for use by the container are used. If they aren't specified, default credentials are used.
- GSI Secure conversation authentication is enforced for all methods.

Note: Changing the required authentication and authorization method will require suitable changes to the clients that contact this service.

To alter the security descriptor configuration, refer to [security descriptor](#). The file to be altered is `$GLOBUS_LOCATION/etc/globus_wsrf_rft/security-config.xml`.

3. Optional configuration: configuring the PostgreSQL database

PostgreSQL (version 7.1 or greater) can be used with RFT but is no longer a requirement. You can either use the packages which came with your operating system (RPMs, DEBs, ...) or build from source. We used PostgreSQL version 7.3.2 for our testing and the following instructions are good for the same.

1. Install PostgreSQL. Instructions on how to install/configure PostgreSQL can be found [here](#)¹.
2. Configure the postmaster daemon so that it accepts TCP connections. This can be done by adding the "-o -i" switch to the postmaster script (This is either the init.d script found in `/etc/init.d/postgresql` or `/var/lib/`, depending on how you installed PostgreSQL). Follow the instructions [here](#)² to start the postmaster with the -i option.
3. You will now need to create a PostgreSQL user that will connect to the database. This is usually the account under which the container is running. You can create a PostgreSQL user by running the following command: `su postgres; createuser globus`. If you get the following error: `psql: could not connect to`

¹ <http://www.postgresql.org/docs/manuals/>

² <http://www.postgresql.org/docs/7.4/static/postmaster-start.html>

server: No such file or directory Is the server running locally and accepting connections on Unix domain socket "/tmp/.s.PGSQL.5432"? this generally means that either your postmaster is not started with the -i option or you didn't restart the postmaster after the above mentioned step.

4. Now you need to set security on the database you are about to create. You can do it by following the steps below:

```
sudo vi /var/lib/pgsql/data/pg_hba.conf and append the following line to the file:
```

```
host rftDatabase "username" "host-ip" 255.255.255.255 md5 Note: use crypt instead of md5 if you are using PostgreSQL 7.3 or earlier.
```

```
sudo /etc/init.d/postgresql restart
```

5. To create the database that is used for RFT run (as user globus): `createdb rftDatabase`.
6. To populate the RFT database with the appropriate schemas run: `psql -d rftDatabase -f $GLOBUS_LOCATION/share/globus_wsrft_rft/rft_schema.sql`. Now that you have created a database to store RFT's state, the following steps configure RFT to find the database:
7. Open `$GLOBUS_LOCATION/etc/globus_wsrft_rft/jndi-config.xml`.
8. Find the `dbConfiguration` section under the `ReliableFileTransferService` `<service>` section.
9. Change the `connectionString` to point to the machine on which you installed PostgreSQL and to the name of the database you used in step 2. If you installed PostgreSQL on the same machine as your Globus install, the default should work fine for you.
10. Change the `userName` to the name of the user who owns/created the database and do the same for the password (it also depends on how you configured your database).
11. Don't worry about the other parameters in the section. The defaults should work fine for now.
12. Edit the configuration section under `ReliableFileTransferService`. There are two values that can be edited in this section:
13.
 - `backOff`: Time in seconds you want RFT to backoff before a failed transfer is retried by RFT. The default should work fine for now.
 - `maxActiveAllowed`: This is the number of transfers the container can do at given point. The default should be fine for now.

4. Optional configuration: configuring the MySQL database

If you prefer MySQL to Postgres or derby you can use it with RFT instead. Instructions on how to this can be found at [here](http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/admin/rft-admin-mysql.html).³

³ <http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/admin/rft-admin-mysql.html>

5. RFT auto-registration with default WS MDS Index Service

With a default GT 4.2.1 installation, the RFT service is automatically registered with the default WS MDS Index Service running in the same container for monitoring and discovery purposes.

There is a jndi resource defined in `$GLOBUS_LOCATION/etc/globus_wsrf_rft/jndi-config.xml` as follows :

```
<resource name="mdsConfiguration"

  type="org.globus.wsrfl.impl.servicegroup.client.MDSConfiguration">
  <resourceParams>
    <parameter>
      <name>reg</name>
      <value>>true</value>
    </parameter>
    <parameter>
      <name>factory</name>
      <value>org.globus.wsrfl.jndi.BeanFactory</value>
    </parameter>
  </resourceParams>
</resource>
```

To configure the automatic registration of RFT to the default WS MDS Index Service, change the value of the parameter `<reg>` as follows:

- `true` turns on auto-registration; this is the default in GT 4.2.1.
- `false` turns off auto-registration.

5.1. Configuring resource properties

By default, the following resource properties (from the RFT Factory Resource) are sent to the default Index Service:

- `ActiveResourceInstances`: A dynamic resource property of the total number of active RFT resources in the container at a given point of time.
- `TotalNumberOfTransfers`: A dynamic resource property of the total number of transfers/deletes performed since the RFT service was deployed in this container.
- `TotalNumberOfActiveTransfers`: A dynamic resource property of the number of active transfers across all rft resources in a container at a given point of time.
- `TotalNumberOfBytesTransferred`: A dynamic resource property of the total number of bytes transferred by all RFT resources created since the deployment of the service.
- `RFTFactoryStartTime`: Time when the service was deployed in the container. Used to calculate uptime.
- `DelegationServiceEPR`: The end point reference of the Delegation resource that holds the delegated credential used in executing the resource.

You can configure which resource properties are sent in RFT's registration.xml file, \$GLOBUS_LOCATION/etc/globus_wsrft_rft/registration.xml. The following is the relevant section of the file:

```
<Content xsi:type="agg:AggregatorContent"
  xmlns:agg="http://mds.globus.org/aggregator/types">

  <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">

    <agg:GetMultipleResourcePropertiesPollType
      xmlns:rft="http://www.globus.org/namespaces/2004/10/rft">
      <!-- Specifies that the index should refresh information
        every 60000 milliseconds (once per minute) -->
      <agg:PollIntervalMillis>60000</agg:PollIntervalMillis>

      <!-- specifies that all Resource Properties should be
        collected from the RFT factory -->

      <agg:ResourcePropertyNames>rft:TotalNumberOfBytesTransferred</agg:ResourcePropertyNames>
      <agg:ResourcePropertyNames>rft:TotalNumberOfActiveTransfers</agg:ResourcePropertyNames>
      <agg:ResourcePropertyNames>rft:RFTFactoryStartTime</agg:ResourcePropertyNames>
      <agg:ResourcePropertyNames>rft:ActiveResourceInstances</agg:ResourcePropertyNames>

      <agg:ResourcePropertyNames>rft:TotalNumberOfTransfers</agg:ResourcePropertyNames>

    </agg:GetMultipleResourcePropertiesPollType>
  </agg:AggregatorConfig>
</agg:AggregatorData/>
</Content>
```

6. Registering RFT manually with default WS MDS Index Service

If a third party needs to register an RFT service manually, see [Registering with mds-servicegroup-add](#) in the WS MDS Aggregator Framework documentation.

Chapter 9. Environment variable interface

1. Environmental variables for RFT

The only environment variable that needs to be set for RFT is `GLOBUS_LOCATION`, in order to run the command line clients, which should be set to the location of the globus installation.

Chapter 10. Debugging

Log output from RFT is a useful tool for debugging issues. Because RFT is built on top of Java WS Core, developer debugging is the same as described in [Chapter 10, Debugging](#). For information about sys admin logs, see [Chapter 7, Debugging](#).

1. Development Logging in Java WS Core

The following information applies to Java WS Core and those services built on it.

Logging in the Java WS Core is based on the [Jakarta Commons Logging](#)¹ API. Commons Logging provides a consistent interface for instrumenting source code while at the same time allowing the user to plug-in a different logging implementation. Currently we use [Log4j](#)² as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the [configuration file format](#)³.

1.1. Configuring server side developer logs

Server side logging can be configured in `$GLOBUS_LOCATION/container-log4j.properties`, when the container is stand alone container. For tomcat level logging, refer to [Logging for Tomcat](#)⁴, . The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

Additional logging can be enabled for a package by adding a new line to the configuration file. Example:

```
#for debug level logging from org.globus.package.FooClass
log4j.category.org.globus.package.name.FooClass=DEBUG
#for warnings from org.some.warn.package
log4j.category.org.some.warn.package=WARN
```

1.2. Configuring client side developer logs

Client side logging can be configured in `$GLOBUS_LOCATION/log4j.properties`. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

2. Enable Debug Logging in RFT

A standard way to debug RFT is to make the container print out more verbose error messages. You can do this with the following steps:

Edit `$GLOBUS_LOCATION/container-log4j.properties` and add following line to it:

```
log4j.category.org.globus.transfer=DEBUG
```

¹ <http://jakarta.apache.org/commons/logging/>

² <http://logging.apache.org/log4j/>

³ [http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure\(java.lang.String,org.apache.log4j.spi.LoggerRepository\)](http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure(java.lang.String,org.apache.log4j.spi.LoggerRepository))

⁴ <http://tomcat.apache.org/tomcat-5.5-doc/logging.html>

. For more verbosity add

```
log4j.category.org.globus.ftp=DEBUG
```

, which will print out Gridftp messages too.

Chapter 11. Troubleshooting

For a list of common errors in GT, see [Error Codes](#).

1. Errors

Table 11.1. Reliable File Transfer (RFT) Errors

Error Code	Definition	Possible Solutions
Error creating RFT Home: Failed to connect to database ... Until this is corrected all RFT request will fail and all GRAM jobs that require staging will fail	This occurs when you start the container if RFT is not configured properly to talk to a PostgreSQL database.	The usual cause is that Postmaster is not accepting TCP connections, which means that you must restart Postmaster with the <code>-i</code> option (see Configuring RFT).
ERROR service.RFTResourceManager [Thread-13,transferCompleted:517] Unable to update on finished org.globus.transfer.reliable.service.database.RftDBException: RFT database update error [Caused by: Syntax error: Encountered ")" at line 1, column 47.]	This error occurs as a result of a dynamically built SQL update string. The update occurs when a transfer completes. It is used to notify transfer requests using the same hosts that resources on that host have been freed. The error message occurs when no rows in the database match that host.	Users of RFT may safely ignore this error. The message is harmless to the functionality of RFT and will not affect the results of a transfer in any way. The exception is safely caught. Future versions of RFT will have optimizations to avoid this step.

2. RFT fault-tolerance and recovery

RFT uses PostgreSQL to check-point transfer state in the form of restart markers and recover from transient transfer failures, using retry mechanism with exponential backoff, during a transfer. RFT has been tested to recover from source and/or destination server crashes during a transfer, network failures, container failures (when the machine running the container goes down), file system failures, etc. RFT Resource is implemented as a `PersistentResource`, so `ReliableFileTransferHome` gets initialized every time a container gets restarted. Please find a more detailed description of fault-tolerance and recovery in RFT below:

- **Source and/or destination GridFTP failures:** In this case RFT retries the transfer for a configurable number of maximum attempts with exponential backoff for each retry (the backoff time period is configurable also). If a failure happens in the midst of a transfer, RFT uses the last restart marker that is stored in the database for that transfer and uses it to resume the transfer from the point where it failed, instead of restarting the whole file. This failure is treated as a container-wide backoff for the server in question. What this means is that all other transfers going to/from that server, across all the requests in a container, will be backed off and retried. This is done in order to prevent further failures of the transfers by using knowledge available in the database.
- **Network failures:** Sometimes this happens due to heavy load on a network or for any other reason packets are lost or connections get timed out. This failure is considered a transient failure and RFT retries the transfer with exponential backoff for that particular transfer (and not the whole container, as with the source and/or destination GridFTP failures).

- **Container failures:** These type of failures occur when the machine running the container goes down or if the container is restarted with active transfers. When the container is restarted, it restarts ReliableTransferHome, which looks at the database for any active RFT resources and restarts them.

2.1. Failure modes that are not addressed:

- Running out of disk space for the database.

Chapter 12. Related Documentation

- [Lessons learned producing an OGSi compliant Reliable File Transfer Service¹](#) (pdf)
- [Reliable Data Transport: A Critical Service for the Grid²](#) (pdf)

¹ http://www-unix.mcs.anl.gov/%7Ekeahey/DBGS/DBGS_files/dbgs_papers/allcock.pdf

² <http://www.doc.ic.ac.uk/%7Eesn5/GGF/GGF11/BGBS-Allcock.pdf>

Index

A

api, 10
architecture, 9

C

commands
 globus-crft, 17
 rft, 15
 rft-delete, 19
compatibility, 2
configuration interface, 23
 auto-registration with WS MDS Index, 25
 overview, 23
 resource properties, 25
 settings, 23
configuring, 23
 auto-registration to WS MDS Index Service, 25
 MySQL database, 24
 PostgreSQL database, 23
 registering manually to WS MDS Index Service, 26
 resource properties, 25

D

debugging
 (for developers), 28
 logging, 28
deleting a set of files and directories (with GridFTP), 6
deleting files (from a list), 19
dependencies, 2

E

environmental variables, 27
errors, 30

F

fault tolerance, 30
features, 1

L

logging
 debugging, 28

P

papers, 32
platforms, tested, 2

R

recovery, 30

resource properties, 12

S

security considerations, 3
services, 11
submitting a third party GridFTP transfer, 15
submitting transfers (from a list)
 C client, 17

T

transfer request, 20
 options, 20
 schema, 20
transferring large datasets (with GridFTP), 4
troubleshooting
 (for developers), 30

U

usage scenarios, 4

W

wSDL, 11