

GT 4.2.1 WS MDS Aggregator Framework: System Administrator's Guide

GT 4.2.1 WS MDS Aggregator Framework: System Administrator's Guide

Introduction

This guide contains advanced configuration and other sysadmin information specific to the WS MDS Aggregator Framework.

Important

The Aggregator Framework is built, installed, and deployed as part of the standard Globus Toolkit installation procedure: [Installing GT 4.2.1](#) and [WS MDS System Administrator's Guide](#).

Table of Contents

Aggregator How-to	5
1. Configuring the Aggregator Framework	1
1. Configuration overview	1
2. Syntax of the interface	1
2. Testing	3
3. Security Considerations	4
1. WS MDS Aggregator Services (Index Service and Trigger Service) Security Considerations	4
4. Debugging	5
1. Logging in Java WS Core	5
5. Troubleshooting	7
1. Error Messages	7
2. Aggregator isn't collecting data	7
3. Aggregator entry disappears	7
Glossary	8
Index	9

List of Tables

5.1. WS MDS Aggregator Error Messages	7
---	---

Aggregator How-to

C

- configuration interface,
 - aggregator sinks,
 - aggregator sources,
 - disable publishing,
- overview,
- configuring,
 - aggregator sinks,
 - aggregator sources,
 - disabling publishing,
- overview,

D

- debugging
 - logging,

E

- errors,

L

- logging
 - CEDPS-compliant,
 - debugging,

S

- security considerations,

Chapter 1. Configuring the Aggregator Framework

WS MDS aggregator services (such as MDS Index, MDS Trigger and MDS Archive Tech Preview) inherit their configuration system from the *Aggregator Framework* module.

The Aggregator Framework does not have its own service -side configuration, although services which are based on the framework have their own service-side configuration options (such as *MDS Index* and *MDS Trigger*) which are documented in the per-service documentation.

Registrations to a working Aggregator Framework are configured for the `mds-servicegroup-add(1)` tool. This tool takes an XML configuration file listing registrations, and causes those registrations to be made.

In general, configuration of aggregator services involves configuring the service to get information from one or more sources in a Grid. The mechanism for doing this is defined by (inherited from) the Aggregator Framework and described in this section.

1. Configuration overview

Configuring an Aggregating Service Group to perform a data aggregation is performed by specifying an AggregatorContent object as the content parameter of a ServiceGroup `add` method invocation. An AggregatorContent object is composed of two `xsd:any` arrays: AggregatorConfig and AggregatorData:

- The AggregatorConfig `xsd:any` array is used to specify parameters that are to be passed to the underlying AggregatorSource when the ServiceGroup `add` method is invoked. These parameters are generally type-specific to the implementation of the AggregatorSource and/or AggregatorSink being used.
- The AggregatorData `xsd:any` array is used as the storage location for aggregated data that is the result of message deliveries to the AggregatorSink. Generally, the AggregatorData parameter of the AggregatorContent is not populated when the ServiceGroup `add` method is invoked, but rather is populated by message delivery from the AggregatorSource.

2. Syntax of the interface

2.1. Configuring the Aggregator Sources

For detailed information on configuring the three types of aggregator sources provided by the Globus Toolkit, see [Aggregator Sources Reference](#).

- [Configuring Execution Aggregator Source](#)
- [Configuration file: parameters for the query aggregator source](#)
- [Configuration file: parameters for the subscription aggregator source](#)

2.2. Configuring the Aggregator Sink

An aggregator sink may require sink-specific configuration (for example, the MDS *Trigger Service* requires sink-specific configuration; the MDS *Index Service* does not). See the documentation for the specific *aggregator service* being used for details on sink-specific documentation.

2.2.1. Disabling the publishing of the aggregator configuration on the server side

It is now possible to disable the publishing of the aggregator configuration along with the aggregated data. The following optional parameter can be added to the *AggregatorConfiguration* section of the service `jndi-config.xml` file:

```
<parameter>
  <name>publishAggregatorConfiguration</name>
  <value>false</value> </parameter>
```

By default, this option is disabled and the aggregator configuration information is published.

Chapter 2. Testing

The Aggregator Framework is a software framework used to create services. To test that the Aggregator Framework is working, deploy and test a service (such as [Index Service](#)).

Chapter 3. Security Considerations

1. WS MDS Aggregator Services (Index Service and Trigger Service) Security Considerations

By default, the *aggregator sources* do not use authentication credentials -- they retrieve information using anonymous SSL authentication or no authentication at all, and thus retrieve only publicly-available information. If a user or administrator changes that configuration so that a service's aggregator source uses credentials to acquire non-privileged data, then that user or administrator must configure the service's aggregator sink to limit access to authorized users.

Chapter 4. Debugging

1. Logging in Java WS Core

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

Java WS Core server side has two types of loggers. One logger is used for development logging and by default writes to standard out. The other logger includes system administration information and is [CEDPs best practices](#)¹ compliant.

On client side, only developer logging is available and is configured using `log4j.properties`.

1.1. 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](#)² 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](#)³ as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the [configuration file format](#)⁴.

1.1.1. 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](#)⁵. 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
```

1.1.2. 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.

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

² <http://jakarta.apache.org/commons/logging/>

³ <http://logging.apache.org/log4j/>

⁴ [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))

⁵ <http://tomcat.apache.org/tomcat-5.5-doc/logging.html>

1.2. Configuring system administration logs

The specific logger to edit will be `log4j.logger.sysadmin` in `$GLOBUS_LOCATION/container-log4j.properties`. There you can configure the following properties:

```
log4j.appender.infoCategory=org.apache.log4j.RollingFileAppender
log4j.appender.infoCategory.Threshold=INFO
log4j.appender.infoCategory.File=var/containerLog
log4j.appender.infoCategory.MaxFileSize=10MB
log4j.appender.infoCategory.MaxBackupIndex=2
```

Above implies the logging file is rolling with each file size limited to 10MB and the logging information is stored in `$GLOBUS_LOCATION/var/containerLog`.

1.3. Sample log file

The [sample log file](#)⁶ contains many log entries for various scenarios in the Java WS container.

⁶ <http://www.globus.org/toolkit/docs/4.2/4.2.1/common/javawscore/sample-container-log.txt>

Chapter 5. Troubleshooting

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

1. Error Messages

Table 5.1. WS MDS Aggregator Error Messages

Error Code	Definition	Possible Solutions
error	what causes this	possible solutions
WS MDS is built on Java WS Core, please see Java WS Core Error Codes for more error code documentation.		

2. Aggregator isn't collecting data

Problem: I was able to successfully register an aggregator entry with `mds-servicegroup-add`, but the aggregator isn't collecting data for the registration.

Solution: The fact that the registration was successful does not mean that there are no errors in the registration parameters. Verify that details such as resource EPRs, resource property names, and queries are accurate, and check the container logs for the [aggregator service](#) and (if applicable) the remote service for more information.

3. Aggregator entry disappears

Problem: I was able to successfully register an aggregator entry with `mds-servicegroup-add`, and the aggregator collected information for this entry for a while, but then the entry disappeared.

Solution: make sure that `mds-servicegroup-add` is still running. Registrations time out; `mds-servicegroup-add` refreshes them periodically.

Glossary

A

Aggregator Framework	A software framework used to build services that collect and aggregate data. WS MDS Services (such as the Index and Trigger services) are built on the Aggregator Framework, and are sometimes called Aggregator Services.
aggregator services	Services that are built on the Aggregator Framework, such as the WS MDS Index Service and Trigger Service.
aggregator source	A Java class that implements an interface (defined as part of the Aggregator Framework) to collect XML-formatted data. WS MDS contains three aggregator sources: the query aggregator source, the subscription aggregator source, and the execution aggregator source.

I

Index Service	An aggregator service in WS MDS that serves as a registry similar to UDDI, but much more flexible. Indexes collect information and publish that information as WSRF resource properties.
---------------	--

T

Trigger Service	An aggregator service (in WS MDS) that collects information and compares that data against a set of conditions defined in a configuration file. When a condition is met, or triggered, the specified action takes place (for example, an email is sent to a system administrator when the disk space on a server reaches a threshold).
-----------------	--

Index

C

- configuration interface, 1
 - aggregator sinks, 1
 - aggregator sources, 1
 - disable publishing, 2
 - overview, 1
- configuring, 1
 - aggregator sinks, 1
 - aggregator sources, 1
 - disabling publishing, 2
 - overview, 1

D

- debugging
 - logging, 5

E

- errors, 7

L

- logging
 - CEDPS-compliant, 5
 - debugging, 5

S

- security considerations, 4

GT 4.2.1 WS MDS Aggregator Framework: Developer's Guide

GT 4.2.1 WS MDS Aggregator Framework: Developer's Guide

Introduction

The Aggregator Framework allows pluggable data sources and sinks to be written and connected together. Generally a source collects data from or about a particular grid resource, and passes it to a sink which does something interesting with it.

The aggregator sinks supplied with the toolkit implement the Index Service and Trigger Service. The *aggregator sources* supplied with the toolkit collect information using resource property queries, subscription/notification, and execution of external programs.

This document describes the programmatic interfaces to the Aggregator Framework. See also general Globus Toolkit coding guidelines¹ and GT 4.2.1 best practices.

¹ http://www.globus.org/toolkit/docs/development/coding_guidelines.html

Table of Contents

Aggregator How-to	6
1. Before you begin	1
1. Feature summary	1
2. Tested platforms	1
3. Backward compatibility summary	1
4. Technology dependencies	2
5. WS MDS Aggregator Services (Index Service and Trigger Service) Security Considerations	2
2. Usage scenarios	3
1. Creating WS MDS services	3
3. Tutorials	4
4. Architecture and design overview for the WS MDS Aggregator Framework	5
1. Standard aggregator sinks	6
2. Standard aggregator sources	6
5. APIs	8
1. Programming Model Overview	8
2. The Aggregating ServiceGroup framework	8
3. The standard plugins	8
4. Component API	8
6. WS and WSDL	9
1. Protocol overview	9
2. Operations	9
3. WS MDS Aggregator Framework Resource Properties	10
4. Faults	10
5. WSDL and Schema Definition	10
I. WS MDS Commands	11
mds-servicegroup-add	12
mds-set-multiple-termination-time	15
7. Configuring execution aggregator source	16
1. Introduction	16
2. Registering	16
3. Configuration file: parameters for the execution aggregator source	18
4. Configuring the executable	19
5. Ping test example	20
6. Troubleshooting	21
8. Configuring the Aggregator Framework	22
1. Configuration overview	22
2. Syntax of the interface	22
9. Overview of Graphical User Interface	24
10. Debugging	25
1. Development Logging in Java WS Core	25
2. Enable Debug Logging for the Aggregator Framework	25
11. Troubleshooting	26
1. Java WS Core Errors	27
2. General troubleshooting information	29
12. Related Documentation	30
Glossary	31
Index	32

List of Figures

4.1. Graphic of Information Services Flow	5
---	---

List of Tables

4.1. Standard aggregator sinks	6
4.2. Standard aggregator sources	7
7.1. Aggregator configuration parameters	18
11.1. Java WS Core Errors	28

Aggregator How-to

A

- aggregator sources,
 - execution,
 - configuring,
 - troubleshooting,
 - query,
 - GetMultipleResourcePropertiesPollType,
 - GetResourcePropertyPollType,
 - QueryResourcePropertiesPollType,
 - subscription,
- AggregatorServiceGroup resource properties,
- apis,
- architecture,

C

- compatibility,
- configuration file, registering
 - example-aggregator-registration.xml,
 - parameters,
 - execution aggregator source,
- configuration interface,
 - aggregator sinks,
 - aggregator sources,
 - disable publishing,
 - overview,
- configuring,
 - aggregator sinks,
 - aggregator sources,
 - disabling publishing,
 - execution aggregator source,
 - executable,
 - overview,
- configuring termination time of resources registered to MDS aggregator services,
- creating WS MDS services,

D

- debugging,
 - logging,
- dependencies,

E

- EPR,
- errors,

F

- features,

L

- logging
 - debugging,

M

- mds-servicegroup-add,
- mds-set-multiple-termination-time,

O

- org.globus.mds.aggregator.impl,

P

- platforms,

R

- registering
 - ping test example,
- registering resources to MDS aggregator services,
- registering with the default Index Service,
- related documentation,
- resource properties, AggregatorServiceGroup,

S

- security considerations,
- services,

T

- troubleshooting
 - execution aggregator source,
 - for developers,
- tutorials
 - Build a Grid Service,

U

- usage scenarios,

W

- WSDL,

Chapter 1. Before you begin

1. Feature summary

Features new in release GT 4.2.1

- The `mds-servicegroup-add` command no longer requires the `-s` or `-e` arguments
- The `mds-set-multiple-termination-time` command has been created to aid in lifetime management of service group entry resources created via `mds-servicegroup-add`

Other Supported Features

- Collects information from grid resources using pluggable aggregation sources which collect information by polling, subscription, and by execution of local scripts.
- Delivers collected information to pluggable information sinks.
- Management of individual aggregations is now performed over the wire through WS ServiceGroup APIs.
- The `QueryAggregatorSource` and `SubscriptionAggregatorSource` now attempt to detect when the data source EPR is local to the current container instance, and if so set the connection properties to use local transport.
- Added a service level configuration option for suppressing the publication of aggregator configuration elements in aggregator service group registry entries. In other words, the Service Group "Content" Resource Property will contain only the aggregated data.

2. Tested platforms

Tested Platforms for WS MDS Aggregator Framework

- Linux on i386
- Windows XP

3. Backward compatibility summary

The Aggregator framework is completely backward compatible with the version included in GT 4.0.x.

Protocol changes since GT version 4.0

- The Aggregator Framework is affected by the Java WS Core protocol changes (see the [Java WS Core Release Notes](#) for details)

API changes since GT version 4.0

- None. Aggregator sources and execution information providers written for GT version 4.0 should continue to work in this version.

Schema changes since GT version 4.0

- See the [Java WS Core Release Notes](#).

4. Technology dependencies

Aggregator Framework depends on the following GT components:

- Java WS Core

Aggregator Framework depends on the following 3rd party software:

- None

5. WS MDS Aggregator Services (Index Service and Trigger Service) Security Considerations

By default, the *aggregator sources* do not use authentication credentials -- they retrieve information using anonymous SSL authentication or no authentication at all, and thus retrieve only publicly-available information. If a user or administrator changes that configuration so that a service's aggregator source uses credentials to acquire non-privileged data, then that user or administrator must configure the service's aggregator sink to limit access to authorized users.

Chapter 2. Usage scenarios

1. Creating WS MDS services

The Aggregator Framework is used to create MDS services by linking an *aggregator source* (a java class that implements the AggregatorSource interface to collect data) to an *aggregator sink* (a java class that implements the AggregatorSink interface to process data, e.g., by providing a service interface for it). The AggregatorSource and AggregatorSink interfaces are documented in [Chapter 5, APIs](#).

[provide more concrete info/example?]

Chapter 3. Tutorials

Use of the index service (based on the WS MDS Aggregator Framework) is covered in the [Build a Grid Service Tutorial \(GlobusWORLD 2005\)](#)¹.

¹ <http://www.globus.org/toolkit/tutorials/BAS/>

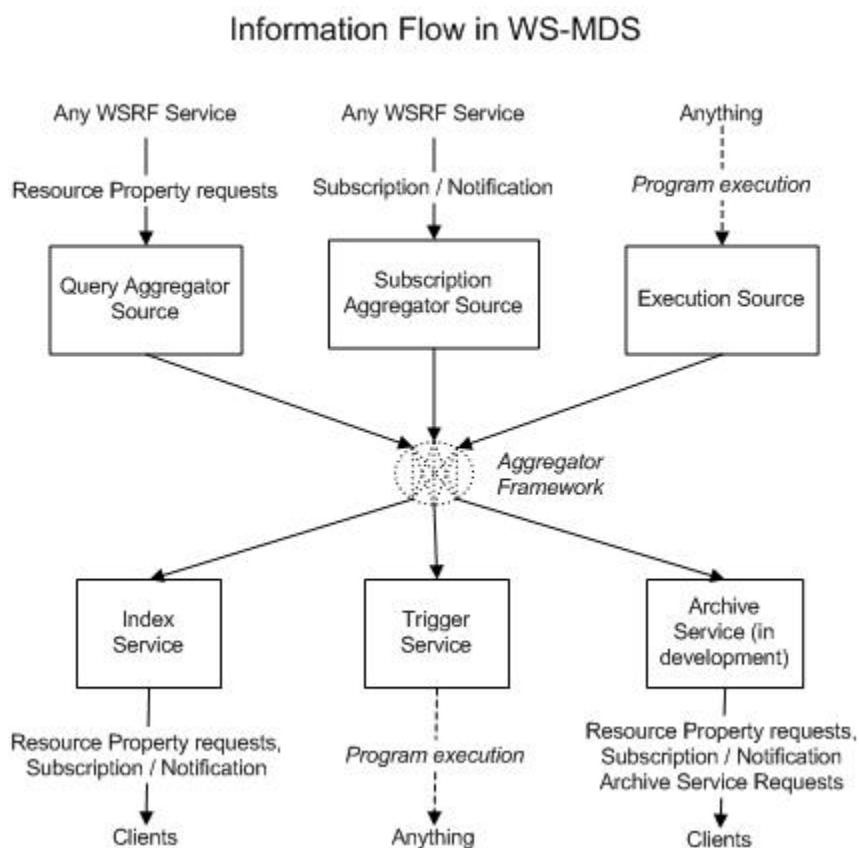
Chapter 4. Architecture and design overview for the WS MDS Aggregator Framework

The WS MDS Aggregator Framework is the software framework on which WS MDS aggregator services are built. The Aggregator Framework collects data from an **aggregator source** and sends that data to an **aggregator sink** for processing.

Aggregator sources distributed with the Globus Toolkit include modules that query resource properties, acquire data through subscription/notification, and execute programs to generate data.

Another way of describing the Aggregator Framework is that it is designed to facilitate the collecting of information from or about WS-Resources via plugin aggregator sources and the feeding of that information to plugin aggregator sinks, which can then perform actions such as re-publishing, logging, or archiving the information.

Figure 4.1. Graphic of Information Services Flow



Aggregators work on a type of service group called an **AggregatorServiceGroupRP**. Resources may be *registered* to an AggregatorServiceGroupRP using the service group add operation, which will cause an entry to be added to the service group. The entry will include configuration parameters for the aggregator source; when the registration is made, the appropriate aggregation source and sinks will be informed; the aggregator source will begin collecting data and

inserting it into the corresponding service group entry, and the aggregator sink will begin processing the information in the service group entries.

The method of collection by source and processing by the sink is dependent on the particular instantiation of the aggregator framework.

1. Standard aggregator sinks

The aggregator sinks distributed with the toolkit (`org.globus.mds.aggregator.impl.ServiceGroupEntryAggregatorSink` and `org.globus.mds.trigger.impl.TriggerResource`) are described in the following table.

Table 4.1. Standard aggregator sinks

Aggregator Sink	Service Implemented	Description
<code>ServiceGroupEntryAggregatorSink</code>	Index Service	The servicegroup sink (used by the <i>Index Service</i>) publishes received data as content in the <code>AggregatingServiceGroup</code> entry used to manage the registration. This data can therefore be retrieved by querying the index for its 'entries' resource property.
<code>TriggerResource</code>	Trigger Service	The <i>Trigger Service</i> provides an aggregator sink which receives data, applies tests to that data, and if the tests match, runs a specified executable. See the <i>Trigger Service</i> documentation for more information.

2. Standard aggregator sources

The aggregator sources supplied with the toolkit collect information using resource property queries (query sources), subscription/notification (subscription sources), and execution of external programs (execution sources).

The aggregator sources supplied with the Globus Toolkit are listed in the following table.



Note

All aggregator sources listed in this table are in the `org.globus.mds.aggregator.impl` package, so for example the aggregator source listed as `QueryAggregatorSource` is actually `org.globus.mds.aggregator.impl.QueryAggregatorSource`

Table 4.2. Standard aggregator sources

Aggregator Source	Description
QueryAggregatorSource	<p>The query source collects information from a registered resource by using WS-Resource Properties polling mechanisms:</p> <ul style="list-style-type: none">• <code>GetResourcePropertyPollType</code>; requests a single Resource Property from the remote resource.• <code>GetMultipleResourcePropertiesPollType</code>; requests multiple Resource Properties from the remote resource.• <code>QueryResourcePropertiesPollType</code>; requests a query be executed against the Resource Property Set of the remote resource. <p>Polls are made periodically, with both the period and target Resource Properties specified in the registration message.</p>
SubscriptionAggregator-Source	<p>The subscription source collects information from a registered resource using WS-Notification mechanisms. Data is delivered when property values change, rather than periodically.</p>
ExecutionAggregator-Source	<p>The execution source collects information about (not necessarily from) a registered resource by execution of a local executable, which is passed as input the identity of the registered resource. Details of the interface between the execution source and local executables are in Configuring Execution Aggregator Source.</p>

Chapter 5. APIs

1. Programming Model Overview

The Aggregator Framework module consists of an Aggregating ServiceGroup framework which supports plugins as detailed below, as well as a number of standard plugins.

2. The Aggregating ServiceGroup framework

The aggregating servicegroup framework is designed to facilitate the collecting of information from or about WS-Resources (via plugin *aggregator sources*) and the feeding of that information to plugin aggregator sinks.

The framework provides for over-the-wire management of the list of registered resources (through a WS-ServiceGroup interface) and a Java API for connecting sources and sinks together.

In general (although this is not a hard requirement), aggregator sinks will be tied into a specific service implementation, while aggregator sources are more independent. (For example, the trigger and index services act as sinks)

3. The standard plugins

A number of standard aggregator sources are provided, which implement the aggregator source API. These provide for collecting information from/about a WS-Resource by:

- WS-ResourceProperties poll operations
- WS-Notification subscription
- Execution of arbitrary executables

See [Aggregator Sources Reference](#) for more information about standard aggregator sources for GT 4.2.1.

4. Component API

There are two main Java interfaces in the aggregator framework.

- [AggregatorSink](#)¹ - which is implemented by sinks that can receive data from the Aggregator Framework.
- [AggregatorSource](#)² - which is implemented by sources that can feed data into the Aggregator Framework.

In addition, the AggregatorContent class is used when configuring an aggregator service programmatically, and to represent the data published in the aggregator's `Entry` resource property. All aggregator classes and interfaces are documented in the [aggregator Java API documentation](#)³

¹ http://www.globus.org/api/javadoc-4.2.1/globus_wsrf_mds_aggregator/org/globus/mds/aggregator/impl/AggregatorSink.html

² http://www.globus.org/api/javadoc-4.2.1/globus_wsrf_mds_aggregator/org/globus/mds/aggregator/impl/AggregatorSource.html

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

Chapter 6. WS and WSDL

1. Protocol overview

The Aggregator Framework builds on the [WS-ServiceGroup](#)¹ and [WS-ResourceLifetime](#)² specifications. Those specifications should be consulted for details on the syntax of each operation.

Each Aggregator Framework is represented as a WS-ServiceGroup (specifically, an AggregatorServiceGroup).

Resources may be registered to an AggregatorServiceGroup using the AggregatorServiceGroup Add operation. Each registration will be represented as a ServiceGroupEntry resource. Resources may be *registered* to an AggregatorServiceGroup using the service group add operation, which will cause an entry to be added to the service group.

The entry will include configuration parameters for the *aggregator source*; when the registration is made, the following will happen:

1. The appropriate aggregation source and sinks will be informed,
2. the aggregator source will begin collecting data and inserting it into the corresponding service group entry,
3. and the aggregator sink will begin processing the information in the service group entries.

The method of collection by source and processing by the sink is dependent on the particular instantiation of the aggregator framework (see [per-source documentation](#) for source information and [per-service documentation](#) for sink information - for example the [Index Service](#) and the [Trigger Service](#).)

2. Operations

2.1. AggregatorServiceGroup

- `add`: This operation is used to register a specified resource with the Aggregator Framework. In addition to the requirements made by the WS-ServiceGroup specification, the Content element of each registration must be an AggregatorContent type, with the AggregatorConfig element containing configuration information specific to each source and sink (documented in the [System Administrator's Guide](#)).

2.2. AggregatorServiceGroupEntry

- `setTerminationTime`: This operation can be used to set the termination time of the registration, as detailed in WS-ResourceLifetime.

¹ http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/servicegroup/sgw-2.wsdl?revision=1.2&view=markup&pathrev=globus_4_2_branch

² http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/lifetime/rlw-2.wsdl?revision=1.2&view=markup&pathrev=globus_4_2_branch

3. WS MDS Aggregator Framework Resource Properties

3.1. AggregatorServiceGroup Resource Properties

- `Entry`: This resource property publishes details of each registered resource, including both an EPR to the resource, the Aggregator Framework configuration information, and data from the sink.
- `RegistrationCount`: This resource property publishes registration load information (the total number of registrations since service startup and decaying averages)

4. Faults

The Aggregator Framework throws standard WS-ServiceGroup, WS-ResourceLifetime, and WS-ResourceProperties faults and does not define any new faults of its own.

5. WSDL and Schema Definition

- [AggregatorServiceGroup](#)³
- [AggregatorServiceGroupEntry](#)⁴
- [common types used by AggregatorServiceGroup and AggregatorServiceGroupEntry](#)⁵

Other relevant source files are the:

- [WSRF service group schema](#)⁶
- [WSRF resource lifetime schema](#)⁷
- MDS Usefulrp schema.

³ http://viewcvs.globus.org/viewcvs.cgi/ws-mds/aggregator/schema/mds/aggregator/aggregator_service_group_port_type.wsdl?revision=1.5&view=markup&pathrev=globus_4_2_branch

⁴ http://viewcvs.globus.org/viewcvs.cgi/ws-mds/aggregator/schema/mds/aggregator/aggregator_service_group_entry_port_type.wsdl?revision=1.6&view=markup&pathrev=globus_4_2_branch

⁵ http://viewcvs.globus.org/viewcvs.cgi/ws-mds/aggregator/schema/mds/aggregator/aggregator-types.xsd?revision=1.6&view=markup&pathrev=globus_4_2_branch

⁶ http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/servicegroup/sgw-2.wsdl?revision=1.2&view=markup&pathrev=globus_4_2_branch

⁷ http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/lifetime/rlw-2.wsdl?revision=1.2&view=markup&pathrev=globus_4_2_branch

WS MDS Commands

Name

`mds-servicegroup-add` -- Registering grid resources to aggregating MDS services such as the Index, Archive and Trigger services

`mds-servicegroup-add`

Tool description

mds-servicegroup-add creates a set of registrations to a WS-ServiceGroup and periodically renews those registrations. It is intended primarily for registering grid resources to aggregating MDS services such as the Index and Trigger services.

The tool can be deployed at the aggregating service, at resource services, or at any other location.

This allows registrations to be configured by the administrator of the aggregating service, or by the administrator of resources, by a third party, or by some combination of those.

Registrations are defined in an XML configuration file, which is documented here: [Registering Aggregator Sources](#).

For an example using an Index Service, see [Simple usage for the Index Service](#).

And remember to note the section on [Limitations](#).

Command syntax

The basic syntax for **mds-servicegroup-add** is:

```
mds-servicegroup-add [options] config.xml
```

where:

<code>-s ht-tp(s)://host:port/service-group-address</code>	A URL to the service group against which the <code>mds-servicegroup-add</code> request will be executed. This command line argument is an optional argument, it is only necessary that this URL argument be specified in the case that there are no suitable target service group EPRs present in the configuration file . Any end point references found in the configuration file will automatically override the EPR specified by this argument on the command-line. If this argument is not specified and no suitable service group EPR is present in the configuration file, the target EPR defaults to the <code>DefaultIndexService</code> on the local host using the default TLS port of 8443.
<code>-o outputFile</code>	If this argument is specified, mds-servicegroup-add will write the EPRs of all successfully created service group entries from the target resource to this file. This file can then be used as input to the mds-set-multiple-termination-time command.
<code>-q seconds</code>	By default, mds-servicegroup-add will continue to run, refreshing the lifetimes for the service group entry resources it creates. Use this option to cause mds-servicegroup-add to terminate itself after the specified number of seconds has elapsed. This can be helpful when using long-lifetime registrations or when updating entry lifetimes via a different mechanism.
<code>-a</code>	By default, mds-servicegroup-add will attempt to make an authenticated connection to each service group. This option is used to specify anonymous connections (and to prevent mds-servicegroup-add from failing if you don't have a valid Grid credential).
<code>-z auth_type</code>	Specify an authorization type:

	<p><code>self</code> Fail if the server's identity is different from the user's identity.</p> <p><code>host</code> Fail if the server does not have a valid server certificate.</p> <p><code>none</code> Continue regardless of the server's identity.</p>
<code>config.xml</code>	<p>Path to the registration configuration file (see Registering Aggregator Sources).</p> <p>The Globus Toolkit distribution includes an example configuration file: <code>\$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml</code>.</p>

The [common java client options](#) are also supported.

Registering a resource manually

Prerequisites

You need the following before you register a resource with an Index Service:

Note

Beginning with GT 4.0.1, the CAS, RFT and GRAM4 services are automatically registered with the default Index Service.

- Have a working Index Service, as documented in the [Index Service System Administrator's Guide](#).
- Know the EPR to the resource.
- Know the EPR to the Index Service. This can be seen in the container output at startup of the container on the index host, and will look something like this: `https://myhost:8443/wsrf/services/DefaultIndexService`

Simple usage for the Index Service

The simplest way to register resources to an index is to:

1. Edit the example configuration file (`$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml`), replacing the EPRs in that file with the EPRs for your resources
2. Run **mds-servicegroup-add** to perform the registrations specified in that file.

For example, to register to the `DefaultIndexService` with a modified `example-aggregator-registration.xml` file, you could run a command similar to the following:

```
$GLOBUS_LOCATION/bin/mds-servicegroup-add -s \
https://127.0.0.1:8443/wsrf/services/DefaultIndexService \
$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml
```

Limitations

It may be necessary for the tool to continue to run in order for the registrations that it maintains to be kept alive, as registrations will otherwise time out.

Name

`mds-set-multiple-termination-time` -- Administering the termination time of grid resources created by aggregating MDS services such as the Index and Trigger services

`mds-set-multiple-termination-time`

Tool description

mds-set-multiple-termination-time sets the termination time of multiple service group entries. It is intended primarily for working with groups of service group entry resources created by aggregating MDS services such as the Index and Trigger services.

The tool can be deployed at the aggregating service, at resource services, or at any other location.

This allows the lifetime of registrations to be configured by the administrator of the aggregating service, or by the administrator of resources, by a third party, or by some combination of those.

Command syntax

The basic syntax for **mds-set-multiple-termination-time** is:

`mds-set-multiple-termination-time [options]`

where:

<code>-i inputFile</code>	file containing an XML array of Endpoint References, such as one output by the mds-servicegroup-add command when used with the <code>-o</code> option.
<code>-w delay</code>	integer wait delay in seconds that will be added to the current time at the remote resource to generate the resource termination time. If not specified the termination time by defaults is set to the current time at the remote resource.
<code>-n date string</code>	ISO-8601 formatted date string representing an exact date and time, e.g. 2016-06-28T01:06:430Z If not specified the termination time by default is set to the current time at the remote resource.

The [common java client options](#) are also supported.

Chapter 7. Configuring execution aggregator source

1. Introduction

The execution aggregation source provides a way to aggregate data (arbitrary XML information) about a registered resource using an arbitrary local executable (such as an external script). The executable will be passed registration information as parameters and is expected to output the gathered data, as detailed below.

A basic example of the use of this API is described in the [fixme ping test example] for the aggregator execution source.

The execution aggregation source will periodically execute an identified executable. The identity of the executable and the frequency with which it is to run are specified in the registration message.

2. Registering

2.1. Registering resources (general)

To register resources with the Index Service:

1. Create a configuration file in XML that specifies registrations. See `$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml` for several specific examples. The configuration file is described in more detail below.
2. Run `mds-servicegroup-add(1)` to perform the registrations specified in that configuration file. For example, to register to the DefaultIndexService with a modified `example-aggregator-registration.xml` file, you could run a command similar to the following:

```
$GLOBUS_LOCATION/bin/mds-servicegroup-add -s \  
https://127.0.0.1:8443/wsrp/services/DefaultIndexService \  
$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml
```

- Each registration has a limited lifetime; **mds-servicegroup-add** should be left running in the background so that it can continue to refresh registrations.
- Depending on administration preference, it may be run on the same host as the aggregator service, on the same host as a member resource, or on any other host(s).

The configuration file consists of an optional `defaultServiceGroupEPR`, an optional `defaultRegistrantEPR`, and then one or more `ServiceGroupRegistrationParameters` blocks, each of which represents one registration.

You can use the example configuration at `$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml`¹, replacing the EPRs in that file with the EPRs for your resources. It includes many examples of configurations for GRAM, RFT and other situations.

The general syntax of the configuration file is:

¹ http://viewcvs.globus.org/viewcvs.cgi/*checkout*/ws-mds/aggregator/source/etc/example-aggregator-registration.xml?revision=1.13

```
<?xml version="1.0" encoding="UTF-8" ?>
<ServiceGroupRegistrations
  xmlns="http://mds.globus.org/servicegroup/client">

  // An optional default service group EPR.
  <defaultServiceGroupEPR>
    // Default service group EPR
  </defaultServiceGroupEPR>

  // An optional default registrant EPR.
  <defaultRegistrantEPR>
    // Default registrant EPR
  </defaultRegistrantEPR>

  // An optional default security descriptor file.
  <defaultSecurityDescriptorFile>
    // Path name of default security descriptor file
  </defaultSecurityDescriptorFile>

  // One or more service group registration blocks:

  <ServiceGroupRegistrationParameters>
    <ServiceGroupEPR>
      // EPR of the service group to register to
    </ServiceGroupEPR>
    <RegistrantEPR>
      // EPR of the entity to be monitored.
    </RegistrantEPR>
    <InitialTerminationTime>
      // Initial termination time
    </InitialTerminationTime>
    <RefreshIntervalSecs>
      // Refresh interval, in seconds
    </RefreshIntervalSecs>
    <Content type="agg:AggregatorContent">
      // Aggregator-source-specific configuration parameters
    </Content>
  </ServiceGroupRegistrationParameters>

</ServiceGroupRegistrations>
```

The following table describes the different blocks of the file and any parameters:

Table 7.1. Aggregator configuration parameters

defaultService-GroupEPR block	Provides a convenient way to register a number of resources to a single service group -- for example, if you wish to register several resources to your default VO index, you can specify that index as the default service group and omit the ServiceGroupEPR blocks from each ServiceGroupRegistrationParameters block.
defaultRegistrantEPR	Provides a convenient way to register a single resource to several service groups -- for example, if you wish to register your local GRAM server to several index servers, you can specify your GRAM server as the default registrant and omit the RegistrantEPR blocks from each ServiceGroupRegistrationParameters block.
defaultSecurityDescriptorFile	Simply the path to the security descriptor file .
ServiceGroupRegistrationParameters	Each ServiceGroupRegistrationParameters block specifies the parameters used to register a resource to a service group. The parameters specified in this block are:
ServiceGroupEPR	The EPR of the service group to register to. This parameter may be omitted if a defaultServiceGroupEPR block is specified; in this case, the value of defaultServiceGroupEPR will be used instead.
RegistrantEPR	The EPR of the resource to register. This parameter may be omitted if a defaultRegistrantEPR block is specified; in this case, the value of defaultRegistrantEPR will be used instead.
InitialTerminationTime	The initial termination time of this registration (this may be omitted). If the initial termination time is omitted, then the mds-servicegroup-add sets the initial termination time to the current wall time plus 2 times that of the specified RefreshIntervalSecs parameter.
RefreshIntervalSecs	The refresh interval of the registration, in seconds. The mds-servicegroup-add(1) will attempt to refresh the registration according to this interval, by default incrementing the termination time of the registration by 2 times this interval for every successful refresh. If at any point the termination time for the registration expires the registration will be subject to removal within a maximum of 5 minutes.
Content	Aggregator-source-specific registration parameters. The content blocks for the various aggregator sources are described in detail in the following sections.

3. Configuration file: parameters for the execution aggregator source

The configuration block for ExecutionAggregatorSource (inside the Content block) looks like this:

```
<Content xsi:type="agg:AggregatorContent"
  xmlns:agg="http://mds.globus.org/aggregator/types">
  <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
    <agg:ExecutionPollType>
      <agg:PollIntervalMillis>interval_in_ms</agg:PollIntervalMillis>
      <agg:ProbeName>dummy_namespace:probe_name</agg:ProbeName>
    </agg:ExecutionPollType>
  </agg:AggregatorConfig>
```

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

where:

`PollIntervalMillis` This parameter is the poll refresh period in milliseconds.

`ProbeName` This parameter specifies the name of the probe to run. To configure this name, go to the `jndi-config.xml` file for the service being configured and make sure an `executableMappings` parameter maps probe names to executable names.

Important

All executables must be in the directory `$GLOBUS_LOCATION/libexec/aggrexec` and you should only specify the name of the script/program, without any qualification or path; GT automatically expands the name to `$GLOBUS_LOCATION/libexec/aggrexec/probename`.

For example, the file for the MDS Index Service is `$GLOBUS_LOCATION/etc/globus_wsrif_mds_index_jndi-config.xml`, and the following code within that file maps the probe name `aggr-test` to the executable called `aggregator-exec-test.sh` and maps the probe name `pingexec` to the executable `example-ping-exec`.

```
<resource name="configuration"
  type="org.globus.mds.aggregator.impl.AggregatorConfiguration">
  <resourceParams>
    // ...
    <parameter>
      <name>executableMappings</name>
      <value>aggr-test=aggregator-exec-test.sh, pingexec=example-ping-exec</value>
    </parameter>
  </resourceParams>
</resource>
```

4. Configuring the executable

4.1. Name of executable

The executable to run will be `$GLOBUS_LOCATION/libexec/aggrexec/<scriptname>` with `scriptname` supplied by the `ProbeName` parameter in the configuration file.

4.2. Input to executable

Information about the registration will be supplied as command line parameters and on `stdin`.

A single command line parameter will be supplied to the executable. This will be the URL from the EPR of the registered service.

Two XML documents will be sent to `stdin`, in sequence:

1. The first document will be the full EPR to the registered service.
2. The second document will be the AggregatorConfig block from the registration message (configuration file).

4.3. Output from executable

The executable must output a well-formed XML document to stdout. This output document will be delivered into the Aggregator Framework.

5. Ping test example

5.1. Introduction

This file describes an example of using the Execution Aggregator Source API.

The example provides basic ping information about a registrant. It is intended for illustrative purposes, rather than real deployment use.

The aggregator framework is used by other services to collect information. In this example, it will be shown how to configure the index service to use the execution aggregator source.

5.2. Deploying the example

5.2.1. Install the example script in the correct location.

The example script is installed as: `$GLOBUS_LOCATION/etc/globus_wsrfd_mds_aggregator/example-ping-exec`. It is necessary to copy this to the directory where the execution source will look for executables, and ensure that it's executable:

```
$ cp $GLOBUS_LOCATION/etc/globus_wsrfd_mds_aggregator/example-ping-exec
$GLOBUS_LOCATION/libexec/aggrexec/. $ chmod a+x
$GLOBUS_LOCATION/libexec/aggrexec/example-ping-exec
```

5.2.2. Configure the index to use the execution source.

By default, the index is configured to use a WS-Resource Properties polling mechanism. It is necessary for this example to change the index configuration to use the execution source instead.

5.2.3. Register some resources.

This can be achieved using the [mds-servicegroup-add tool](#).

Rather than configuring the client to register with parameters for the Resource Property polling source, parameters for the execution source should be supplied in each registration.

Register both resources that you know exist, and also some non-existent resources.

5.2.4. Observe the results.

Start the container hosting the index, start the `mds-servicegroup-add` tool, then query the contents of the index with:

```
$ wsrf-query -s http://localhost:8080/wsrf/services/DefaultIndexService '/*'
```

Each registration should be represented as an Entry resource property value in the output of wsrf-query; embedded in each entry should be an `$examplePingResult` element with the results of ping testing.

6. Troubleshooting

If you've properly configured and registered your script for execution but are getting errors from the container because it cannot find the specified script, there are two likely causes.

First, make sure that your script/program is executable and is located in the `$GLOBUS_LOCATION/libexec/aggrexec` directory. When it's specified in the configuration mentioned above, only specify the name of the script/program, without any qualification or path. For example, using a `ProbeName` of `test-script` would specify the file `$GLOBUS_LOCATION/libexec/aggrexec/test-script`.

Next, make sure that you have correctly created an `executableMappings` definition in the appropriate `jndi-config.xml` file.

Chapter 8. Configuring the Aggregator Framework

WS MDS aggregator services (such as MDS Index, MDS Trigger and MDS Archive Tech Preview) inherit their configuration system from the *Aggregator Framework* module.

The Aggregator Framework does not have its own service -side configuration, although services which are based on the framework have their own service-side configuration options (such as [MDS Index](#) and [MDS Trigger](#)) which are documented in the per-service documentation.

Registrations to a working Aggregator Framework are configured for the `mds-servicegroup-add(1)` tool. This tool takes an XML configuration file listing registrations, and causes those registrations to be made.

In general, configuration of aggregator services involves configuring the service to get information from one or more sources in a Grid. The mechanism for doing this is defined by (inherited from) the Aggregator Framework and described in this section.

1. Configuration overview

Configuring an Aggregating Service Group to perform a data aggregation is performed by specifying an AggregatorContent object as the content parameter of a ServiceGroup `add` method invocation. An AggregatorContent object is composed of two `xsd:any` arrays: AggregatorConfig and AggregatorData:

- The AggregatorConfig `xsd:any` array is used to specify parameters that are to be passed to the underlying AggregatorSource when the ServiceGroup`add` method is invoked. These parameters are generally type-specific to the implementation of the AggregatorSource and/or AggregatorSink being used.
- The AggregatorData `xsd:any` array is used as the storage location for aggregated data that is the result of message deliveries to the AggregatorSink. Generally, the AggregatorData parameter of the AggregatorContent is not populated when the ServiceGroup `add` method is invoked, but rather is populated by message delivery from the AggregatorSource.

2. Syntax of the interface

2.1. Configuring the Aggregator Sources

For detailed information on configuring the three types of aggregator sources provided by the Globus Toolkit, see [Aggregator Sources Reference](#).

- [Configuring Execution Aggregator Source](#)
- [Configuration file: parameters for the query aggregator source](#)
- [Configuration file: parameters for the subscription aggregator source](#)

2.2. Configuring the Aggregator Sink

An aggregator sink may require sink-specific configuration (for example, the MDS *Trigger Service* requires sink-specific configuration; the MDS *Index Service* does not). See the documentation for the specific *aggregator service* being used for details on sink-specific documentation.

2.2.1. Disabling the publishing of the aggregator configuration on the server side

It is now possible to disable the publishing of the aggregator configuration along with the aggregated data. The following optional parameter can be added to the *AggregatorConfiguration* section of the service `jndi-config.xml` file:

```
<parameter>
  <name>publishAggregatorConfiguration</name>
  <value>false</value> </parameter>
```

By default, this option is disabled and the aggregator configuration information is published.

Chapter 9. Overview of Graphical User Interface

There is no GUI specifically for the Aggregator Framework. The release contains [WebMDS](#) which can be used to display monitoring information in a web browser. Specifically, it can be directed at services based on the Aggregator Framework to display information about resources registered to the Aggregator Framework.

Chapter 10. Debugging

Log output from WS MDS is a useful tool for debugging issues. Because WS MDS is built on top of Java WS Core, developer debugging is the same as described in [Chapter 10, Debugging](#). For information on sys admin logs, see [Chapter 4, 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 for the Aggregator Framework

To turn on debug logging for the Aggregator Framework, add the line:

```
log4j.category.org.globus.mds.aggregator=DEBUG
```

to the appropriate properties file.

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>

Chapter 11. Troubleshooting

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

1. Java WS Core Errors

Table 11.1. Java WS Core Errors

Error Code	Definition
Failed to acquire notification consumer home instance from registry	Caused by <code>javax.naming.NameNotFoundException</code> : Name <code>services</code> is not bound in
The WS-Addressing 'To' request header is missing	This warning is logged by the container if the request did not contain the necessary <i>WS-Addressing</i> headers, those headers at all or is somehow misconfigured.
java.io.IOException: Token length X > 33554432	If you see this error in the container log, it usually means you are trying to connect to HTTPS server using <code>https</code> specifies 8443 as a port number and <code>http</code> as the protocol name.
java.lang.NoSuchFieldError: DOCUMENT	This error usually indicates a mismatch between the version of Apache Axis that the code was compiled with and the version currently running with.
org.globus.wsrfl.InvalidResourceKeyException: Argument key is null / Resource key is missing	These errors usually indicate that a resource key was not passed with the request or that an invalid resource key was used (the element QName of the resource key did not match what the service expected).
Unable to connect to localhost:xxx	Cannot resolve localhost. The machine's <code>/etc/hosts</code> isn't set up correctly and/or you do not have DNS for
org.globus.common.ChainedIOException: Failed to initialize security context	This may indicate that the user's proxy is invalid.
Error: org.xml.sax.SAXException: Unregistered type: class xxx	This may indicate that an Axis generated XML type, defined by the WS RLS XSD, was not properly registered upon deployment without intervention by the user, sometimes they do not.
No socket factory for 'https' protocol	<p>When a client fails with the following exception:</p> <pre>java.io.IOException: No socket factory for 'https' protocol at org.apache.axis.transport.http.HTTPSender.getSocket(HTTPSender.java:100) org.apache.axis.transport.http.HTTPSender.writeToSocket(HTTPSender.java:110) org.apache.axis.transport.http.HTTPSender.invoke(HTTPSender.java:120)</pre> <p>FIXME - it may have happened because...</p>

Error Code	Definition
No client transport named 'https' found	<p>When a client fails with the following exception:</p> <pre>No client transport named 'https' found at org.apache.axis.client.AxisClient.invoke(AxisClient.java:170) at org.apache.axis.client.Call.invokeEngine(Call.java:2726)</pre> <p>The client is most likely loading an incorrect <code>client-config.wsdd</code> configuration file.</p>
ConcurrentModificationException in Tomcat 5.0.x	<p>If the following exception is visible in the Tomcat logs at startup, it might cause the HTTPSValve to fail:</p> <pre>java.util.ConcurrentModificationException at java.util.HashMap\$HashIterator.nextEntry(HashMap.java:782) at java.util.HashMap\$EntryIterator.next(HashMap.java:824) at java.util.HashMap.putAllForCreate(HashMap.java:424) at java.util.HashMap.clone(HashMap.java:656) at mx4j.server.DefaultMBeanRepository.clone(DefaultMBeanRepository.</pre> <p>The HTTPSValve might fail with the following exception:</p> <pre>java.lang.NullPointerException at org.apache.coyote.tomcat5.CoyoteRequest.setAttribute(CoyoteRequestFacade.java:100) at org.apache.coyote.tomcat5.CoyoteRequestFacade.setAttribute(CoyoteRequestFacade.java:100) at org.globus.tomcat.coyote.valves.HTTPSValve.expose(HTTPSVAlve.java:100)</pre> <p>These exceptions will prevent the transport security from working properly in Tomcat.</p>
java.net.SocketException: Invalid argument or cannot assign requested address	<p>FIXME - what causes this?</p>
GAR deploy/undeploy fails with container is running error	<p>A GAR file can only be deployed or undeployed locally while the container is off. However, GAR deployment fails with this error even if the container is off. This usually happens if the container has crashed or was stopped from cleaning up its state files.</p>

2. General troubleshooting information

- In general, if you want to investigate a problem on your own please see [Chapter 10, Debugging](#) for details on how to turn on debugging.
- Most of the command line clients have a `-debug` option that will display more detailed error messages, including the error stack traces.
- [Search the mailing lists](#)¹ such as gt-user@globus.org² or jwscore-user@globus.org³ (before posting a message).
- If you think you have found a bug please report it in our [Bugzilla](#)⁴ system. Please include as much as detail about the problem as possible.

¹ <http://www.globus.org/email-archive-search.php>

² <mailto:gt-user@globus.org>

³ <mailto:jwscore-user@globus.org>

⁴ <http://bugzilla.globus.org/bugzilla/>

Chapter 12. Related Documentation

Specifications for resource properties, service groups, and subscription/notification are available at <http://www.globus.org/wsrfl/>.

Glossary

A

Aggregator Framework	A software framework used to build services that collect and aggregate data. WS MDS Services (such as the Index and Trigger services) are built on the Aggregator Framework, and are sometimes called Aggregator Services.
aggregator services	Services that are built on the Aggregator Framework, such as the WS MDS Index Service and Trigger Service.
aggregator source	A Java class that implements an interface (defined as part of the Aggregator Framework) to collect XML-formatted data. WS MDS contains three aggregator sources: the query aggregator source, the subscription aggregator source, and the execution aggregator source.

I

Index Service	An aggregator service in WS MDS that serves as a registry similar to UDDI, but much more flexible. Indexes collect information and publish that information as WSRF resource properties.
---------------	--

T

Trigger Service	An aggregator service (in WS MDS) that collects information and compares that data against a set of conditions defined in a configuration file. When a condition is met, or triggered, the specified action takes place (for example, an email is sent to a system administrator when the disk space on a server reaches a threshold).
-----------------	--

W

Web Services Addressing (WSA)	The WS-Addressing specification defines transport-neutral mechanisms to address web services and messages. Specifically, it defines XML elements to identify web service endpoints and to secure end-to-end endpoint identification in messages. See the W3C WS Addressing Working Group ¹⁴ for details.
-------------------------------	---

¹⁴ <http://www.w3.org/2002/ws/addr/>

Index

A

- aggregator sources, 7
 - execution, 7
 - configuring, 16
 - troubleshooting, 21
 - query, 7
 - GetMultipleResourcePropertiesPollType, 7
 - GetResourcePropertyPollType, 7
 - QueryResourcePropertiesPollType, 7
 - subscription, 7
- AggregatorServiceGroup resource properties, 10
- apis, 8
- architecture, 5

C

- compatibility, 1
- configuration file, registering
 - example-aggregator-registration.xml, 16
 - parameters, 17
 - execution aggregator source, 18
- configuration interface, 22
 - aggregator sinks, 22
 - aggregator sources, 22
 - disable publishing, 23
 - overview, 22
- configuring, 22
 - aggregator sinks, 22
 - aggregator sources, 22
 - disabling publishing, 23
 - execution aggregator source, 16
 - executable, 19
 - overview, 22
- configuring termination time of resources registered to MDS aggregator services, 15
- creating WS MDS services, 3

D

- debugging, 25
 - logging, 25
- dependencies, 2

E

- EPR, 18
- errors, 27

F

- features, 1

L

- logging
 - debugging, 25

M

- mds-servicegroup-add, 12, 16
- mds-set-multiple-termination-time, 15

O

- org.globus.mds.aggregator.impl, 6

P

- platforms, 1

R

- registering
 - ping test example, 20
- registering resources to MDS aggregator services, 12
- registering with the default Index Service, 16
- related documentation, 30
- resource properties, AggregatorServiceGroup, 10

S

- security considerations, 2
- services, 9

T

- troubleshooting
 - execution aggregator source, 21
 - for developers, 26
- tutorials
 - Build a Grid Service, 4

U

- usage scenarios, 3

W

- WSDL, 9

GT 4.2.1 WS MDS Aggregator Sources Reference

GT 4.2.1 WS MDS Aggregator Sources Reference

Abstract

This reference describes the aggregator sources provided with the GT 4.2.1, how to register them [fixme with what specifically?], and configure each type.

Table of Contents

Aggregator Sources Howtos	6
1. Introduction	1
2. Types of Aggregator Sources in WS MDS	2
1. QueryAggregatorSource	2
2. SubscriptionAggregatorSource	2
3. ExecutionAggregatorSource	2
3. Registering Aggregator Sources	4
1. Registering resources (general)	4
2. Further configuration	6
4. Configuration file: parameters for the query aggregator source	7
1. GetResourcePropertyPollType	7
2. GetMultipleResourcePropertiesPollType	7
3. QueryResourcePropertiesPollType	8
5. Configuration file: parameters for the subscription aggregator source	9
6. Configuring Execution Aggregator Source	10
1. Configuration file: parameters for the execution aggregator source	10
2. Troubleshooting	11
3. Configuring the executable	11
4. Ping test example	11
Glossary	13
Index	14

List of Figures

1.1. Graphic of Information Services Flow 1

List of Tables

3.1. Aggregator configuration parameters	6
--	---

Aggregator Sources Howtos

A

- aggregator sources,
 - configuring
 - executable for the execution aggregator source,
 - execution ,
 - query ,
 - subscription ,
 - execution,
 - query,
 - registering,
 - subscription,
 - troubleshooting,

C

- configuration file, registering
 - example-aggregator-registration.xml,
 - parameters,

E

- EPR,

I

- information flow (diagram),

M

- mds-servicegroup-add,

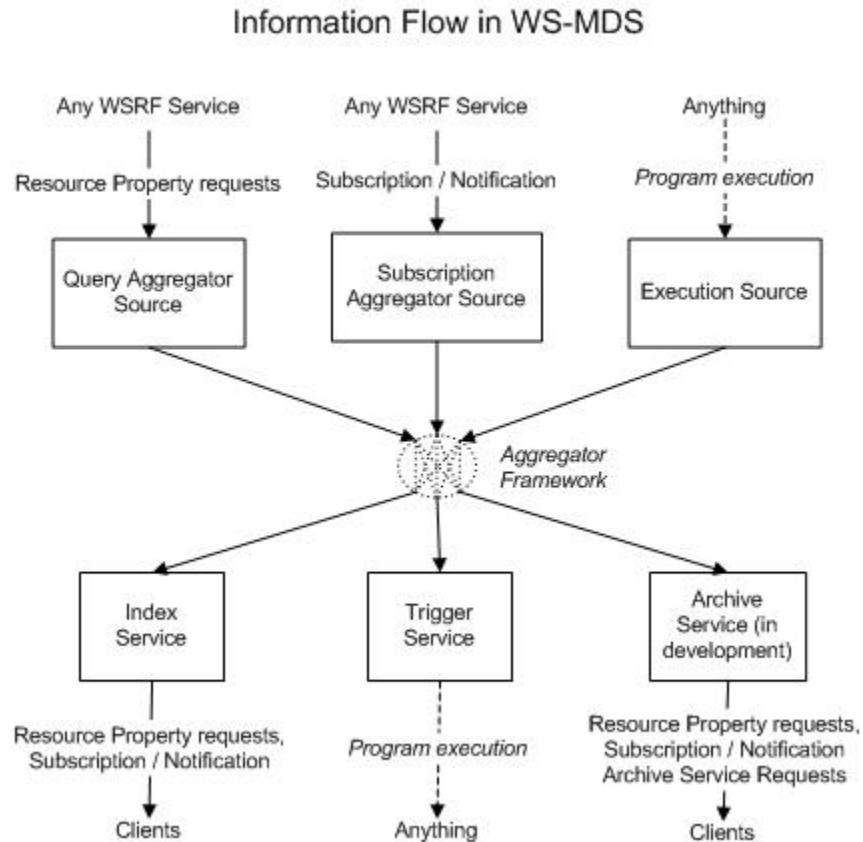
R

- registering
 - ping test example,
- registering aggregator sources,
- registering with the default Index Service,

Chapter 1. Introduction

Aggregator sources [fixme link to glossary] collect information from or about WS-Resources and feed that information to aggregator sinks (such as the Index Service and Trigger Service). The following graphic describes the basic information flow including the three standard aggregator sources: *Query Aggregator Source*, *Subscription Aggregator Source* and Execution Source.

Figure 1.1. Graphic of Information Services Flow



Chapter 2. Types of Aggregator Sources in WS MDS

The aggregator sources supplied with the toolkit collect information using resource property queries (query sources), subscription/notification (subscription sources), and execution of external programs (execution sources).

note? query and subscription agg sources are WSRF, execution source is not.

The aggregator sources supplied with the Globus Toolkit are listed in the following sections.



Note

All aggregator sources listed in this table are in the `org.globus.mds.aggregator.impl` package, so for example the aggregator source listed as `QueryAggregatorSource` is actually `org.globus.mds.aggregator.impl.QueryAggregatorSource`

1. QueryAggregatorSource

The query source collects information from a registered resource by using WS-Resource Properties polling mechanisms:

- `GetResourcePropertyPollType`; requests a single Resource Property from the remote resource.
- `GetMultipleResourcePropertiesPollType`; requests multiple Resource Properties from the remote resource.
- `QueryResourcePropertiesPollType`; requests a query be executed against the Resource Property Set of the remote resource.

The `QueryAggregatorSource` will attempt to detect when the data source EPR is local to the current container instance, and if so set the connection properties to use local transport.

Polls are made periodically, with both the period and target Resource Properties specified in the registration message.

2. SubscriptionAggregatorSource

The `SubscriptionAggregatorSource` gathers resource property values from the registered resource using WS-Notification subscriptions. Data is delivered when property values change, rather than periodically.

The `SubscriptionAggregatorSource` will attempt to detect when the data source EPR is local to the current container instance, and if so set the connection properties to use local transport.

3. ExecutionAggregatorSource

The execution aggregation source provides a way to aggregate data (arbitrary XML information) about a registered resource using an arbitrary local executable (such as an external script). The executable will be passed registration information as parameters and is expected to output the gathered data.

A basic example of the use of this API is described in the [fixme link to ping test example] for the aggregator execution source.

The execution aggregation source will periodically execute an identified executable. The identity of the executable and the frequency with which it is to run are specified in the registration message.

Details of the interface between the execution source and local executables are in [Chapter 6, Configuring Execution Aggregator Source](#)

Chapter 3. Registering Aggregator Sources

The following is general configuration information necessary for all aggregator sources (including any custom ones).

1. Registering resources (general)

To register resources with the Index Service:

1. Create a configuration file in XML that specifies registrations. See `$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml` for several specific examples. The configuration file is described in more detail below.
2. Run `mds-servicegroup-add(1)` to perform the registrations specified in that configuration file. For example, to register to the `DefaultIndexService` with a modified `example-aggregator-registration.xml` file, you could run a command similar to the following:

```
$GLOBUS_LOCATION/bin/mds-servicegroup-add -s \  
https://127.0.0.1:8443/wsrp/services/DefaultIndexService \  
$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml
```

- Each registration has a limited lifetime; **mds-servicegroup-add** should be left running in the background so that it can continue to refresh registrations.
- Depending on administration preference, it may be run on the same host as the aggregator service, on the same host as a member resource, or on any other host(s).

The configuration file consists of an optional `defaultServiceGroupEPR`, an optional `defaultRegistrantEPR`, and then one or more `ServiceGroupRegistrationParameters` blocks, each of which represents one registration.

You can use the example configuration at `$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml`¹, replacing the EPRs in that file with the EPRs for your resources. It includes many examples of configurations for GRAM, RFT and other situations.

The general syntax of the configuration file is:

```
<?xml version="1.0" encoding="UTF-8" ?>  
<ServiceGroupRegistrations  
  xmlns="http://mds.globus.org/servicegroup/client">  
  
  // An optional default service group EPR.  
  <defaultServiceGroupEPR>  
    // Default service group EPR  
  </defaultServiceGroupEPR>  
  
  // An optional default registrant EPR.  
  <defaultRegistrantEPR>
```

¹ http://viewcvs.globus.org/viewcvs.cgi/*checkout*/ws-mds/aggregator/source/etc/example-aggregator-registration.xml?revision=1.13

```
// Default registrant EPR
</defaultRegistrantEPR>

// An optional default security descriptor file.
<defaultSecurityDescriptorFile>
  // Path name of default security descriptor file
</defaultSecurityDescriptorFile>

// One or more service group registration blocks:

<ServiceGroupRegistrationParameters>
  <ServiceGroupEPR>
    // EPR of the service group to register to
  </ServiceGroupEPR>
  <RegistrantEPR>
    // EPR of the entity to be monitored.
  </RegistrantEPR>
  <InitialTerminationTime>
    // Initial termination time
  </InitialTerminationTime>
  <RefreshIntervalSecs>
    // Refresh interval, in seconds
  </RefreshIntervalSecs>
  <Content type="agg:AggregatorContent">
    // Aggregator-source-specific configuration parameters
  </Content>
</ServiceGroupRegistrationParameters>

</ServiceGroupRegistrations>
```

The following table describes the different blocks of the file and any parameters:

Table 3.1. Aggregator configuration parameters

defaultService-GroupEPR block	Provides a convenient way to register a number of resources to a single service group -- for example, if you wish to register several resources to your default VO index, you can specify that index as the default service group and omit the ServiceGroupEPR blocks from each ServiceGroupRegistrationParameters block.
defaultRegistrantEPR	Provides a convenient way to register a single resource to several service groups -- for example, if you wish to register your local GRAM server to several index servers, you can specify your GRAM server as the default registrant and omit the RegistrantEPR blocks from each ServiceGroupRegistrationParameters block.
defaultSecurityDescriptorFile	Simply the path to the security descriptor file .
ServiceGroupRegistrationParameters	Each ServiceGroupRegistrationParameters block specifies the parameters used to register a resource to a service group. The parameters specified in this block are:
ServiceGroupEPR	The EPR of the service group to register to. This parameter may be omitted if a defaultServiceGroupEPR block is specified; in this case, the value of defaultServiceGroupEPR will be used instead.
RegistrantEPR	The EPR of the resource to register. This parameter may be omitted if a defaultRegistrantEPR block is specified; in this case, the value of defaultRegistrantEPR will be used instead.
InitialTerminationTime	The initial termination time of this registration (this may be omitted). If the initial termination time is omitted, then the <code>mds-servicegroup-add</code> sets the initial termination time to the current wall time plus 2 times that of the specified RefreshIntervalSecs parameter.
RefreshIntervalSecs	The refresh interval of the registration, in seconds. The <code>mds-servicegroup-add(1)</code> will attempt to refresh the registration according to this interval, by default incrementing the termination time of the registration by 2 times this interval for every successful refresh. If at any point the termination time for the registration expires the registration will be subject to removal within a maximum of 5 minutes.
Content	Aggregator-source-specific registration parameters. The content blocks for the various aggregator sources are described in detail in the following sections.

2. Further configuration

Further configuration information is detailed per source: [QueryAggregatorSource](#), [SubscriptionAggregatorSource](#), and [ExecutionAggregatorSource](#).



Note

Using the ExecutionAggregatorSource also requires configuring the executable itself, which is also covered in the [ExecutionAggregatorSource](#) section.

Chapter 4. Configuration file: parameters for the query aggregator source

The QueryAggregatorSource can use one of the following three configuration blocks in the Content block: GetResourcePropertyPollType, GetMultipleResourcePropertiesPollType and QueryResourcePropertiesPollType.

1. GetResourcePropertyPollType

If a GetResourcePropertyPollType block is used, QueryAggregatorSource will request a single resource property. The block has this form:

```
<Content xsi:type="agg:AggregatorContent"
  xmlns:agg="http://mds.globus.org/aggregator/types">
  <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
  <agg:GetResourcePropertyPollType>
  <agg:PollIntervalMillis>interval_in_ms</agg:PollIntervalMillis>
  <agg:ResourcePropertyName>rp_namespace:rp_localname</agg:ResourceProp
  </agg:GetResourcePropertyPollType>
  </agg:AggregatorConfig> <agg:AggregatorData/>
</Content>
```

where:

PollIntervalMillis This parameter is the poll refresh period in milliseconds; the ResourcePropertyNames parameter is the QName of the resource property to poll for.

2. GetMultipleResourcePropertiesPollType

If a GetMultipleResourcePropertiesPollType block is used, QueryAggregatorSource will request one or more resource properties. The block has this form:

```
<Content
  xmlns:agg="http://mds.globus.org/aggregator/types"
  xsi:type="agg:AggregatorContent"> <agg:AggregatorConfig
  xsi:type="agg:AggregatorConfig">
  <agg:GetMultipleResourcePropertiesPollType>
  <agg:PollIntervalMillis>interval_in_ms</agg:PollIntervalMillis>
  <agg:ResourcePropertyNames>rp1_namespace:rp1_localname</agg:ResourcePr
  <agg:ResourcePropertyNames>rp2_namespace:rp3_localname</agg:ResourcePr
  <agg:ResourcePropertyNames>rp3_namespace:rp3_localname</agg:ResourcePr
  </agg:GetMultipleResourcePropertiesPollType>
  </agg:AggregatorConfig> <agg:AggregatorData/>
</Content>
```

where:

PollIntervalMillis This parameter is the poll refresh period in milliseconds; the ResourcePropertyNames parameters are the QNames of the resource properties to poll for. There is no limit on the number of ResourcePropertyNames that may be specified.

3. QueryResourcePropertiesPollType

If a `QueryResourcePropertiesPollType` block is used, `QueryAggregatorSource` will request that a query be executed against the Resource Property Set of the remote resource. In the GT4 implementation of WSRF Core, the only query language that is supported is XPath. The block has this form:

```
<Content xmlns:agg="http://mds.globus.org/aggregator/types"
xsi:type="agg:AggregatorContent"> <agg:AggregatorConfig
xsi:type="agg:AggregatorConfig">
<agg:QueryResourcePropertiesPollType>
<agg:PollIntervalMillis>interval_in_ms</agg:PollIntervalMillis>
<agg:QueryExpressionDialect="dialect">
Query Expression </agg:QueryExpression>
</agg:QueryResourcePropertiesPollType>
</agg:AggregatorConfig> <agg:AggregatorData/>
</Content>
```

where:

<code>PollIntervalMillis</code>	This parameter is the poll refresh period in milliseconds.
<code>QueryExpression</code>	An <code>xsd:any</code> element; the <code>Dialect</code> attribute specifies the dialect of the query expression.

Chapter 5. Configuration file: parameters for the subscription aggregator source

The configuration block for `SubscriptionAggregatorSource` looks like this:

```
<Content
  xmlns:agg="http://mds.globus.org/aggregator/types"
  xsi:type="agg:AggregatorContent">
  <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
    <agg:AggregatorSubscriptionType>
      <TopicExpression Dialect="dialect">
        Topic Expression
      </TopicExpression>
      <Precondition Dialect="dialect">
        Precondition
      </Precondition>
      <Selector Dialect="dialect">
        Selector
      </Selector>
      <SubscriptionPolicy>
        Subscription Policy
      </SubscriptionPolicy>
      <InitialTerminationTime>time</InitialTerminationTime>
    </agg:AggregatorSubscriptionType>
  </agg:AggregatorConfig>
  <agg:AggregatorData/>
</Content>
```

where:

`TopicExpression` This is the only required parameter; it specifies the topic expression to use in the subscription request.

For more information, see [this section](#) of the Java WS Core Developer's Guide.

Chapter 6. Configuring Execution Aggregator Source

The ExecutionAggregatorSource requires more configuration than Query and Subscription. In addition to the config file parameters, you must also configure the executable itself. Troubleshooting info and an example are also provided.

1. Configuration file: parameters for the execution aggregator source

The configuration block for ExecutionAggregatorSource (inside the Content block) looks like this:

```
<Content xsi:type="agg:AggregatorContent"
xmlns:agg="http://mds.globus.org/aggregator/types">
<agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
<agg:ExecutionPollType>
<agg:PollIntervalMillis>interval_in_ms</agg:PollIntervalMillis>
<agg:ProbeName>dummy_namespace:probe_name</agg:ProbeName>
</agg:ExecutionPollType>
</agg:AggregatorConfig>
<agg:AggregatorData/>
</Content>
```

where:

PollIntervalMillis This parameter is the poll refresh period in milliseconds.

ProbeName This parameter specifies name of the probe to run. This probe is defined in the `jndi-config.xml` file for the service being configured (for example, the file for the MDS Index Service is `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index_jndi-config.xml`). An `executableMappings` parameter should be defined within this file to map probe names to executable names. For example, this maps the probe names `aggr-test` and `pingexec` to the executables called `aggregator-exec-test.sh` and `example-ping-exec`, respectively. All executables are presumed to be in the directory `$GLOBUS_LOCATION/libexec/aggrexec`.

```
<resource name="configuration"
type="org.globus.mds.aggregator.impl.Aggregator"
<resourceParams>
// ...
<parameter>
<name>executableMappings</name>
<value>aggr-test=aggregator-exec-test.sh, pingexec=example-ping-exec</value>
</parameter>
</resourceParams>
</resource>
```

2. Troubleshooting

If you've properly configured and registered your script for execution but are getting errors from the container because it cannot find the specified script, there are two likely causes.

First, make sure that your script/program is executable and is located in the `$GLOBUS_LOCATION/libexec/aggrexec` directory. When it's specified in the configuration mentioned above, only specify the name of the script/program, without any qualification or path. For example, using the `ProbeName` as `test-script` will be specifying the file `$GLOBUS_LOCATION/libexec/aggrexec/test-script` script.

Next, make sure that you have correctly created an `executableMappings` definition in the appropriate `jndi-config.xml` file.

3. Configuring the executable

3.1. Name of executable

The executable to run will be `$GLOBUS_LOCATION/libexec/aggrexec/<scriptname>` with `scriptname` supplied by the `ProbeName` parameter in the configuration file.

3.2. Input to executable

Information about the registration will be supplied as command line parameters and on `stdin`.

A single command line parameter will be supplied to the executable. This will be the URL from the EPR of the registered service.

Two XML documents will be sent to `stdin`, in sequence:

1. The first document will be the full EPR to the registered service.
2. The second document will be the `AggregatorConfig` block from the registration message (configuration file).

3.3. Output from executable

The executable must output a well-formed XML document to `stdout`. This output document will be delivered into the Aggregator Framework.

4. Ping test example

4.1. Introduction

This file describes an example of using the Execution Aggregator Source API.

The example provides basic ping information about a registrant. It is intended for illustrative purposes, rather than real deployment use.

The aggregator framework is used by other services to collect information. In this example, it will be shown how to configure the index service to use the execution aggregator source.

4.2. Deploying the example

4.2.1. Install the example script in the correct location.

The example script is installed as: `$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-ping-exec`. It is necessary to copy this to the directory where the execution source will look for executables, and ensure that it's executable:

```
$ cp $GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-ping-  
$GLOBUS_LOCATION/libexec/aggrexec/. $ chmod a+x  
$GLOBUS_LOCATION/libexec/aggrexec/example-ping-exec
```

4.2.2. Configure the index to use the execution source.

By default, the index is configured to use a WS-Resource Properties polling mechanism. It is necessary for this example to change the index configuration to use the execution source instead.

4.2.3. Register some resources.

This can be achieved using the [mds-servicegroup-add tool](#).

Rather than configuring the client to register with parameters for the Resource Property polling source, parameters for the execution source should be supplied in each registration.

Register both resources that you know exist, and also some non-existent resources.

4.2.4. Observe the results.

Start the container hosting the index, start the `mds-servicegroup-add` tool, then query the contents of the index with:

```
$ wsrf-query -s http://localhost:8080/wsrf/services/DefaultIndexService '/*'
```

Each registration should be represented as an `Entry` resource property value in the output of `wsrf-query`; embedded in each entry should be an `$examplePingResult` element with the results of ping testing.

Glossary

Q

query aggregator source An aggregator source (included in WS MDS) that polls a WSRF service for resource property information.

S

subscription aggregator source An aggregator source (included in WS MDS) that collects data from a WSRF service via WSRF subscription/notification.

Index

A

- aggregator sources, 1
 - configuring
 - executable for the execution aggregator source, 11
 - execution , 10
 - query , 7
 - subscription , 9
 - execution, 2
 - query, 2
 - registering, 4
 - subscription, 2
 - troubleshooting, 11

C

- configuration file, registering
 - example-aggregator-registration.xml, 4
 - parameters, 5

E

- EPR, 6

I

- information flow (diagram), 1

M

- mds-servicegroup-add, 4

R

- registering
 - ping test example, 11
- registering aggregator sources, 4
- registering with the default Index Service, 4

GT 4.2.1 WS MDS Migration Guide

GT 4.2.1 WS MDS Migration Guide

Introduction

The following provides available information about migrating from previous versions of the Globus Toolkit.

Table of Contents

1. Migrating MDS from GT4	1
2. Migrating MDS from GT3	2
3. Migrating MDS from GT2	3
Glossary	4

List of Tables

1.1. Comparison of MDS in GT3 and GT4	1
2.1. Comparison of MDS in GT3 and GT4	2
3.1. Comparison of MDS in GT2 and GT4	3

Chapter 1. Migrating MDS from GT4

Although the basic functionality remains the same for MDS in GT4, the architecture has changed from OGSi in GT3 to WSRF in GT4. In OGSi, services advertise *service data*; in WSRF, services advertise *resource properties*. Resource Properties and service data are very similar -- both provide a mechanism for expressing arbitrary data about grid resources in XML format, as well as query and notification/subscription interfaces to that data.

The GT4 *Index Service* provides the same functionality as the GT3 Index Service; however, the GT4 Index Service supports WSRF service group registration and resource property query and subscription/notification mechanisms, while the GT3 Index Service supported OGSi service group registration and service data query and subscription/notification mechanisms.

The following table shows a mapping of some GT3 concepts/tools to GT4.

Table 1.1. Comparison of MDS in GT3 and GT4

Description	GT2 Version	GT4 Version
Query Operations	FindServiceData (to retrieve a single service data element by name or to perform an XPath query against a service's service data elements)	GetResourceProperty (to retrieve a single resource property by name), GetMultipleResourceProperties (to retrieve multiple resource properties by name), and QueryResourceProperties (to perform an XPath query against a service's resource properties).
APIs used for queries	OGSI (GT3) Core APIs	WS Core APIs
Command-line clients used for queries	<code>ogsi-find-service-data</code>	<code>wsrf-get-property</code> , <code>wsrf-get-properties</code> , <code>wsrf-query</code>
Available GUIs	globus-sdb (standalone client) and WebSDB (web interface)	WebMDS (web interface)
Operations for subscription/notification	OGSI NotificationSource / NotificationSink	WS-Notification
APIs used for subscription/notification	OGSI (GT3) Core APIs	WS Core APIs
Index registration mechanism	GT3 services can be configured to publish their service data to index services.	Index Servers maintain aggregating service groups that include registration information (timeout values, the mechanism to use to acquire information, and additional mechanism-specific parameters) The registration is accomplished by adding an entry to an aggregating service group via the <code>mds-servicegroup-add</code> command. In addition, services may be configured to register themselves to the default index server running in the same container.

A more detailed mapping of OGSi concepts to WSRF concepts can be found [here](http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrfl.0.pdf)¹.

¹ http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrfl.0.pdf

Chapter 2. Migrating MDS from GT3

Although the basic functionality remains the same for MDS in GT4, the architecture has changed from OGSi in GT3 to WSRF in GT4. In OGSi, services advertise *service data*; in WSRF, services advertise *resource properties*. Resource Properties and service data are very similar -- both provide a mechanism for expressing arbitrary data about grid resources in XML format, as well as query and notification/subscription interfaces to that data.

The GT4 *Index Service* provides the same functionality as the GT3 Index Service; however, the GT4 Index Service supports WSRF service group registration and resource property query and subscription/notification mechanisms, while the GT3 Index Service supported OGSi service group registration and service data query and subscription/notification mechanisms.

The following table shows a mapping of some GT3 concepts/tools to GT4.

Table 2.1. Comparison of MDS in GT3 and GT4

Description	GT2 Version	GT4 Version
Query Operations	FindServiceData (to retrieve a single service data element by name or to perform an XPath query against a service's service data elements)	GetResourceProperty (to retrieve a single resource property by name), GetMultipleResourceProperties (to retrieve multiple resource properties by name), and QueryResourceProperties (to perform an XPath query against a service's resource properties).
APIs used for queries	OGSi (GT3) Core APIs	WS Core APIs
Command-line clients used for queries	<code>ogsi-find-service-data</code>	<code>wsrf-get-property</code> , <code>wsrf-get-properties</code> , <code>wsrf-query</code>
Available GUIs	globus-sdb (standalone client) and WebSDB (web interface)	WebMDS (web interface)
Operations for subscription/notification	OGSi NotificationSource / NotificationSink	WS-Notification
APIs used for subscription/notification	OGSi (GT3) Core APIs	WS Core APIs
Index registration mechanism	GT3 services can be configured to publish their service data to index services.	Index Servers maintain aggregating service groups that include registration information (timeout values, the mechanism to use to acquire information, and additional mechanism-specific parameters) The registration is accomplished by adding an entry to an aggregating service group via the <code>mds-servicegroup-add</code> command. In addition, services may be configured to register themselves to the default index server running in the same container.

A more detailed mapping of OGSi concepts to WSRF concepts can be found [here](http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf)¹.

¹ http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf

Chapter 3. Migrating MDS from GT2

Although the basic functionality remains the same for MDS in GT4, the architecture, standards used, and implementation have changed significantly in GT2. The following table shows a mapping of some GT2 concepts to GT4 concepts.

Table 3.1. Comparison of MDS in GT2 and GT4

Description	GT2 Version	GT4 Version
Format of data describing a resource	LDAP data hierarchy	XML data document
Query language	LDAP queries	XPath queries
Wire protocol for queries	LDAP	WS-ResourceProperties
APIs used for queries	LDAP APIs	WS Core APIs
Command-line clients used for queries	<code>grid-info-search</code>	<code>wsrf-get-property</code> , <code>wsrf-get-properties</code> , <code>wsrf-query</code>
Available GUIs	Various LDAP browsers	WebMDS
Wire protocol for subscription/notification	Not supported	WS-Notification
APIs used for subscription/notification	Not supported	WS Core APIs
Security support	SAML-based security using X.509 user, proxy and host certificates	HTTPS-based security using X.509 user, proxy and host certificates
Queryable index of aggregated information	GIIS, which publishes data using the LDAP-related standards listed above	WS MDS Index Server, which publishes data using the WSRF-related standards listed above
Queryable source of non-aggregated information	GRIS, which uses <i>information providers</i> to gather data from services and then publishes that data the LDAP-related standards listed above	Individual web services, which publish data about their own resources using WSRF-related standards listed above.
Index registration mechanism	MDS servers (GRIS's and, in some cases, GIIS's) register themselves with a GIIS. An MDS server is configured to register itself to a remote index by editing the local MDS server's <code>grid-info-resource-register.conf</code> file, providing information about the location of the remote index to register to and timeout values for the registration	WS MDS Index servers maintain aggregating service groups that include registration information (timeout values, the mechanism to use to acquire information, and additional mechanism-specific parameters) The registration is accomplished by adding an entry to an aggregating service group via the <code>mds-servicegroup-add</code> command. In addition, services may be configured to register themselves to the default index server running in the same container.
Mechanism used by an index to collect information	GIIS's send LDAP queries to remote serves.	WS MDS Index servers use a plugin-based architecture to support several mechanisms to collect information. The Globus Toolkit supplies plugins that support collecting information via polling (resource property queries), subscription/notification, and by program execution.

Glossary

I

Index Service

An aggregator service in WS MDS that serves as a registry similar to UDDI, but much more flexible. Indexes collect information and publish that information as WSRF resource properties.

information provider

A "helper" software component that collects or formats resource information, for use in WS MDS by an aggregator source or by a WSRF service when creating resource properties.

GT 4.2.1 WS MDS Aggregator Framework: Quality Profile

Table of Contents

1. Test coverage reports	1
2. Code analysis reports	1
3. Outstanding bugs	1
4. Bug Fixes	1
5. Performance reports	1

<titleabbrev>Quality Profile</titleabbrev>

1. Test coverage reports

- None available at this time.

2. Code analysis reports

- None available at this time.

3. Outstanding bugs

- [2807: execution aggregator source junit test needs some work](#)¹
- [All open aggregator bug reports and enhancement requests](#)²

4. Bug Fixes

- No new bugfixes in this release.

5. Performance reports

- None available at this time.

¹ http://bugzilla.globus.org/globus/show_bug.cgi?id=2807

² http://bugzilla.globus.org/globus/buglist.cgi?short_desc_type=allwordssubstr&short_desc=&product=MDS&component=wsrf_aggregator&long_desc_type=allwordssubstr&long_desc=&bug_file_loc_type=allwordssubstr&bug_file_loc=&bug_status=NEW&bug_status=AS-SIGNED&bug_status=REOPENED&emailtype1=substring&email1=&emailtype2=substring&email2=&bugidtype=include&bug_id=&votes=&changedin=&chfieldfrom=&chfieldto=Now&chfieldvalue=&cmdtype=doit&newqueryname=&order=Reuse+same+sort+as+last+time&field0-0=noop&type0-0=noop&value0-0-0=

GT 4.2.1 Release Notes: WS MDS Aggregator Framework

Table of Contents

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

<titleabbrev>Release Notes</titleabbrev>

1. Component Overview

The Aggregator Framework is the software framework on which WS MDS services (currently, the [Index](#), [Trigger](#) and [Archive Services](#)) are built. The Aggregator Framework collects data from an *aggregator source* and sends that data to an *aggregator sink* for processing. Aggregator sources distributed with the Globus Toolkit include modules that query service data, acquire data through subscription/notification, and execute programs to generate data. Aggregator sinks include modules that implement the Index, Trigger and Archive Services interfaces.

2. Feature summary

Features new in release GT 4.2.1

- The `mds-servicegroup-add` command no longer requires the `-s` or `-e` arguments
- The `mds-set-multiple-termination-time` command has been created to aid in lifetime management of service group entry resources created via `mds-servicegroup-add`

Other Supported Features

- Collects information from grid resources using pluggable aggregation sources which collect information by polling, subscription, and by execution of local scripts.
- Delivers collected information to pluggable information sinks.
- Management of individual aggregations is now performed over the wire through WS ServiceGroup APIs.
- The `QueryAggregatorSource` and `SubscriptionAggregatorSource` now attempt to detect when the data source EPR is local to the current container instance, and if so set the connection properties to use local transport.

- Added a service level configuration option for suppressing the publication of aggregator configuration elements in aggregator service group registry entries. In other words, the Service Group "Content" Resource Property will contain only the aggregated data.

3. Summary of Changes in WS MDS Aggregator Framework

No new changes in this release.

4. Bug Fixes

- No new bugfixes in this release.

5. Known Problems

The following problems and limitations are known to exist for WS MDS Aggregator Framework at the time of the 4.2.1 release:

5.1. Limitations

- None known at this time.

5.2. Outstanding bugs

- [2807: execution aggregator source junit test needs some work](#)¹
- [All open aggregator bug reports and enhancement requests](#)²

6. Technology dependencies

Aggregator Framework depends on the following GT components:

- Java WS Core

Aggregator Framework depends on the following 3rd party software:

- None

7. Tested platforms

Tested Platforms for WS MDS Aggregator Framework

- Linux on i386

¹ http://bugzilla.globus.org/globus/show_bug.cgi?id=2807

² http://bugzilla.globus.org/globus/buglist.cgi?short_desc_type=allwordssubstr&short_desc=&product=MDS&component=wsrf_aggregator&long_desc_type=allwordssubstr&long_desc=&bug_file_loc_type=allwordssubstr&bug_file_loc=&bug_status=NEW&bug_status=AS-SIGNED&bug_status=REOPENED&emailtype1=substring&email1=&emailtype2=substring&email2=&bugidtype=include&bug_id=&votes=&changedin=&chfieldfrom=&chfieldto=Now&chfieldvalue=&cmdtype=doit&newqueryname=&order=Reuse+same+sort+as+last+time&field0-0=noop&type0-0=noop&value0-0-0=

- Windows XP

8. Backward compatibility summary

The Aggregator framework is completely backward compatible with the version included in GT 4.0.x.

Protocol changes since GT version 4.0

- The Aggregator Framework is affected by the Java WS Core protocol changes (see the [Java WS Core Release Notes](#) for details)

API changes since GT version 4.0

- None. Aggregator sources and execution information providers written for GT version 4.0 should continue to work in this version.

Schema changes since GT version 4.0

- See the [Java WS Core Release Notes](#).

9. Associated Standards

Associated standards for WS MDS Aggregator Framework:

- WS-ResourceProperties (WSRF-RP)
- WS-ResourceLifetime (WSRF-RL)
- WS-ServiceGroup (WSRF-SG)
- WS-BaseNotification
- WS-Topics

10. For More Information

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

Glossary

A

aggregator source

A Java class that implements an interface (defined as part of the Aggregator Framework) to collect XML-formatted data. WS MDS contains three aggregator sources: the query aggregator source, the subscription aggregator source, and the execution aggregator source.