

GT 4.2.1 WS MDS Trigger Service: System Administrator's Guide

GT 4.2.1 WS MDS Trigger Service: System Administrator's Guide

Introduction

This guide contains advanced configuration information for system administrators working with the WS MDS Trigger Service. It provides references to information on procedures typically performed by system administrators, including installation, configuring, deploying, and testing the installation.

This information is in addition to the basic Globus Toolkit prerequisite, overview, installation, security configuration instructions in [Installing GT 4.2.1](#) and [WS MDS System Administrator's Guide](#). Read through these guides before continuing!

Table of Contents

Trigger Service How-tos	5
1. Building and Installing	1
2. Configuring the Aggregator Framework	2
1. Configuration overview	2
2. Syntax of the interface	2
3. Additional configuration for the Trigger Service	4
1. Additional configuration for the Trigger Service	4
4. Deploying	5
1. Manually registering the Trigger Service	5
5. Testing	6
6. Security Considerations	7
1. WS MDS Aggregator Services (Index Service and Trigger Service) Security Considerations	7
7. Debugging	8
1. Logging in Java WS Core	8
8. Troubleshooting	10
1. No triggers displayed in mds-trigger-view	10
2. Trigger never fires	10
3. Error Messages	10
4. General troubleshooting information	10
Glossary	12
Index	13

List of Tables

8.1. WS MDS Trigger Service Error Messages	10
--	----

Trigger Service How-tos

C

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

D

- debugging
 - logging,
- deploying,

E

- errors,

L

- logging
 - CEDPS-compliant,
 - debugging,

S

- security considerations,

T

- testing,
- troubleshooting,

Chapter 1. Building and Installing

The Trigger Service is installed as part of the standard Globus Toolkit installation.

Chapter 2. 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](#).

- [Chapter 6, Configuring Execution Aggregator Source](#)
- [Chapter 4, Configuration file: parameters for the query aggregator source](#)
- [Chapter 5, 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 3. Additional configuration for the Trigger Service

1. Additional configuration for the Trigger Service

The set of possible actions (programs) that can be executed by the Trigger Service is specified in the file `$GLOBUS_LOCATION/etc/globus_wsrft_mds_trigger/jndi-config.xml`. The `executableMappings` parameter contains a comma-separated list of name=value pairs. The left hand side of each name/value pair is the name assigned to the trigger action; this name can be used in trigger definitions. The right hand side of each name/value pair is the path name (relative to `$GLOBUS_LOCATION/libexec/trigger` of the file to execute).

The sources of information used by the Trigger Service are configured using the `mds-servicegroup-add` command; see the [Aggregator Framework](#) documentation or the [Trigger Service Easy HOWTO](#) for more details and examples.

Triggers themselves are created using the [command line clients](#).

Chapter 4. Deploying

This component is deployed as part of the standard toolkit installation. By default, there are no trigger actions set to fire on container startup, as these must be configured and registered manually.

1. Manually registering the Trigger Service

To manually register the example described in [Chapter 2. Configuring the Aggregator Framework](#) (above) do the following:

1. At this point, we're ready to make a registration with the `TriggerRegistrationService` by running a command similar to the following:

```
$GLOBUS_LOCATION/bin/mds-servicegroup-add \  
    $GLOBUS_LOCATION/etc/globus_wsrf_mds_trigger/trigger-registration-example.xml
```

You should see output similar to the following if your environment has been configured properly:

```
Processing configuration file...  
INFO: Processed 1 registration entries  
INFO: Successfully registered https://127.0.0.1:8443/wsrf/services/DefaultIndexService \  
to servicegroup at https://127.0.0.1:8443/wsrf/services/TriggerRegistrationService
```

Chapter 5. Testing

To determine if the registration was made properly, you can query the `TriggerRegistrationService` using a tool like `$GLOBUS_LOCATION/bin/wsrf-query` and visually inspect the output.

For example, running:

```
$GLOBUS_LOCATION/bin/wsrf-query -s \  
    https://127.0.0.1:8443/wsrf/services/TriggerRegistrationService "/*"
```

should yield output similar to the following for the example above:

...

```
<ns1:Content xsi:type="ns11:AggregatorContent" xmlns:ns11="http://mds.globus.org/aggrega  
<ns11:AggregatorConfig>  
  <agg:GetResourcePropertyPollType xmlns:agg="http://mds.globus.org/aggregator/types" xm  
  <agg:PollIntervalMillis>30000</agg:PollIntervalMillis>  
  <agg:ResourcePropertyName>wssg:Entry</agg:ResourcePropertyName>  
</agg:GetResourcePropertyPollType>  
</ns11:AggregatorConfig>  
<ns11:AggregatorData/>  
</ns1:Content>
```

...

Chapter 6. Security Considerations

The security considerations for the [Aggregator Framework](#) also apply to the Trigger Service:

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 7. Debugging

Because WS MDS is built on Java WS Core, it uses the same sys admin logging, described below:

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 8. Troubleshooting

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

1. No triggers displayed in mds-trigger-view

I created a trigger using the `mds-trigger-create`, but I don't see any triggers when I type `mds-trigger-view`! Where are the triggers? Why is nothing happening?

Did you set up a trigger registration? (See the Trigger Service [Admin Guide](#)) The trigger has been created (unless there was an error), but you will not see it and you cannot access it if the trigger registration has not been set up.

2. Trigger never fires

I'm sure the registration was made properly, but the trigger script never fires. OR I followed all of the above steps, but where are the triggers? Why is nothing happening?

Once you've completed the trigger registration, you can now create individual triggers. Trigger creation is performed using a client. See the [User's Guide](#) for more information on clients.

3. Error Messages

Table 8.1. WS MDS Trigger Service Error Messages

Error Code	Definition	Possible Solutions
Error ; nested exception is: org.apache.commons.httpclient.NoHttpResponseException: The server xxx.x.x.x failed to respond	Happens when trying to create a trigger for the Trigger Service. The above error is accompanied by the following error in container: [JWSCORE-192] Error processing request java.io.IOException: Token length 1347375956 > 33554432. FIXME - what causes this?	Be sure that you have properly edited the <code>client-config-settings</code> file under <code>globus_wsrf_mds_trigger</code> . The <code>DefaultServiceAddress</code> parameter should properly reflect the service prefix from your container, e.g.: <code>https://127.0.0.1:8444/wsrf/services/</code> . The services you wish to monitor should also be consistent.

WS MDS is built on Java WS Core, please see [Java WS Core Error Codes](#) for more error code documentation.

4. 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 lists¹ 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 Bugzilla⁴ 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/>

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, 2
 - aggregator sinks, 2
 - aggregator sources, 2
 - disable publishing, 3
 - overview, 2
 - Trigger Service-specific, 4
- configuring, 2
 - aggregator sinks, 2
 - aggregator sources, 2
 - disabling publishing, 3
 - overview, 2
 - Trigger Service-specific, 4

D

- debugging
 - logging, 8
- deploying, 5

E

- errors, 10

L

- logging
 - CEDPS-compliant, 8
 - debugging, 8

S

- security considerations, 7

T

- testing, 6
- troubleshooting, 10

GT 4.2.1 WS MDS Trigger Service: User's Guide

GT 4.2.1 WS MDS Trigger Service: User's Guide

Introduction

The WS MDS Trigger Service collects information about Grid resources and can be configured to execute a program if the collected data meets certain conditions.

End-users will typically interact with the Trigger Service indirectly, using some mechanism specific to the triggered executable program (for example, an executable program may send mail to an end-user or write a structured log file that will later be read by some other program).

Table of Contents

Trigger Service How-tos	5
1. Trigger Service - Easy HowTo	1
1. Purpose	1
2. Prerequisites	1
3. Introduction	1
4. Trigger Tutorial	2
2. MDS Trigger Commands	9
1. Create a new trigger - mds-trigger-create	9
2. View information about existing trggers - mds-trigger-view	9
3. Modify a trgger - mds-trigger-edit	10
3. Graphical User Interface	12
4. Troubleshooting	13
1. No triggers displayed in mds-trigger-view	13
2. Trigger never fires	13
3. Error Messages	13
4. General troubleshooting information	13
Index	15

List of Tables

2.1. mds-trigger-create options	9
2.2. mds-trigger-view options	10
2.3. mds-trigger-edit options	10
4.1. WS MDS Trigger Service Error Messages	13

Trigger Service How-tos

C

command line clients
admin,

E

errors,

G

GUI
WebMDS,

T

troubleshooting,
tutorial
basic,

Chapter 1. Trigger Service - Easy HowTo

1. Purpose

The purpose of this Easy HowTo is to introduce the GT4/WS MDS component known as the Trigger, as well as to provide an example of setting one up successfully. The current GT 4.2.1 documentation provides a basic reference and will be updated as features are added, but for those of you who would like to get a simple trigger working without going through all of the documentation, this document is for you.

We will be creating a simple trigger from scratch, and setting it up completely. To get the basic idea of how this is done, we will only use elements available in the default GT4 installation to show you how to use triggers.

2. Prerequisites

To get the most out of this tutorial, you will need:

- A Globus Toolkit installation
- Some basic familiarity with [XML Path Language \(XPath\)](#).¹
- A valid X.509 user certificate

3. Introduction

The Trigger Service collects information and then performs actions based on that information. The Trigger Service works like this:

1. Administrators use a configuration file to specify the names and locations of *trigger actions*, programs that can be executed by the Trigger Service as a result of trigger conditions being met.
2. Administrators use a service interface to specify what information will be collected by the Trigger Service. This interface is called the Aggregator Framework and is the same configuration interface used by the Index Service
3. Users use a service interface to define *triggers*. A trigger includes (among other things) an XPath query string and the name of one of the trigger actions defined in step 1.
4. The Trigger Service periodically collects data (based on the configuration specified in step 2) and, for each trigger specified in step 3, evaluates the XPath query associated with the trigger and then executes the trigger's action if the query returns true.

In this example, we will configure the Trigger Service to monitor the Default Index Service running in the same Globus container, and then set up a trigger that will add an entry to a log file any time the number of Index Service is less than 1. This is not necessarily a practical example of how you would use a trigger, but it's simple enough to give you a basic idea of how to set one up. So let's get started!

¹ <http://www.w3.org/TR/xpath>

4. Trigger Tutorial

4.1. Preliminaries: Set Up Your Environment

First things first -- in order to run most Globus commands, you must have your environment set up correctly and have a valid proxy certificate. To set up your environment, first set the `GLOBUS_LOCATION` environment variable to the directory in which Globus is installed. To finish setting up your environment, run:

```
. $GLOBUS_LOCATION/etc/globus-user-env.sh
```

if you're a Bourne shell user, or

```
source $GLOBUS_LOCATION/etc/globus-user-env.csh
```

if you're a C shell user. Finally, generate a proxy certificate with:

```
$GLOBUS_LOCATION/bin/grid-proxy-init -verify -debug
```

4.2. Configure Trigger Action Programs

Next, we will specify what commands can be used in Trigger Service triggers. The Trigger Service comes with some simple action scripts in the `$GLOBUS_LOCATION/libexec/trigger` directory; we will edit the `$GLOBUS_LOCATION/etc/globus_wsrf_mds_trigger/jndi-config.xml` file to enable them:

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
  <global>
    <resource name="configuration"
              type="org.globus.mds.aggregator.impl.AggregatorConfiguration">
      ...
    </resource>

    <resource name="triggerConfiguration"
              type="org.globus.mds.trigger.impl.TriggerConfiguration">
      <resourceParams>
        <parameter>
          ...
        </parameter>
        <parameter>
          <name>executableMappings</name>
          <value>trigger-action-default=trigger-action-default.sh, trigger-action-input-def
        </parameter>
      </resourceParams>
    </resource>

  </global>

  <service name="TriggerRegistrationService">
    ...
```

```
</service>
...
</jndiConfig>
```

This `jndi-config.xml` file defines an `executableMappings` parameter. The format of the `executableMappings` parameter is a sequence of `name=value` strings, separated by commas. The left hand side of each `name/value` pair is the name that users will specify in trigger definitions; the right hand side is the path name (relative to the `$GLOBUS_LOCATION/libexec/trigger` directory) of the program to execute. In this example, we define two trigger actions: `trigger-action-default` maps to `$GLOBUS_LOCATION/libexec/trigger/trigger-action-default.sh`, and `trigger-action-input-default` maps to `$GLOBUS_LOCATION/libexec/trigger-action-input-default.sh`. These action scripts are distributed as part of the Globus distribution. The version of `$GLOBUS_LOCATION/etc/globus_wsrp/mds_trigger/jndi-config.xml` distributed with Globus has these mappings defined in a commented-out section; in order to continue with this example, you must uncomment them.

Before you continue, you'll need to restart your Globus container to make the changes to `jndi-config.xml` take effect. If you normally use `/etc/init.d/gt4container`, then you can type `/etc/init.d/gt4container restart`, or you can kill the running Globus container (if there is one) and run `$GLOBUS_LOCATION/etc/globus-start-container-detached` by hand. If you have a production container running and want to test the trigger service with a different instance on another port, you can run `globus-start-container-detached -p NNNN` to cause the new container to listen on port NNNN.

4.3. Configure the Trigger Service to Collect Information

The next thing we will do is configure the trigger service to collect some information (we will later configure the trigger service to periodically run a query on that information and, based on the results of the query, take some action). In this example, we will configure the trigger service to collect information by querying the Default Index Service running in the same Globus container for the entire contents of its index.

The Trigger Service uses the [Aggregator Framework](#) to configure its sources of information. Aggregator sources are configured through a service interface; we will create a file specifying configuration parameters and then run the `mds-servicegroup-add` command to read that configuration file and register the configuration information with the Trigger Service. We will start with the example trigger registration file included with Globus distributions in `$GLOBUS_LOCATION/etc/globus_wsrp/mds_trigger/trigger-registration-example.xml`

```
<DefaultServiceGroupEPR>
  <wsa:Address>https://localhost:8443/wsrp/services/TriggerRegistrationService</wsa:Address>
</DefaultServiceGroupEPR>
<ServiceGroupRegistrationParameters
  xmlns="http://mds.globus.org/servicegroup/client"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:agg="http://mds.globus.org/aggregator/types">
  <RegistrantEPR
    xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Address>https://localhost:8443/wsrp/services/DefaultIndexService</wsa:Address>
  </RegistrantEPR>
  <RefreshIntervalSecs>600</RefreshIntervalSecs>
  <Content xsi:type="agg:AggregatorContent"
    xmlns:agg="http://mds.globus.org/aggregator/types">
    <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
      <agg:GetResourcePropertyPollType
```

```
        xmlns:wssg="http://docs.oasis-open.org/wsrif/sg-2">
        <agg:PollIntervalMillis>30000</agg:PollIntervalMillis>
        <agg:ResourcePropertyName>wssg:Entry</agg:ResourcePropertyName>
        </agg:GetResourcePropertyPollType>
    </agg:AggregatorConfig>
    <agg:AggregatorData />
</Content>
</ServiceGroupRegistrationParameters>
</ServiceGroupRegistrations>
```

For this example, the only items in the registration file that you might need to edit are the `DefaultServiceGroupEPR` (the address of your trigger registration service) and the `RegistrantEPR` (the address of the resource you want to monitor; in this case, your Default Index Service). If these look correct (i.e., the host names and port numbers correspond to the Globus container you just started), then you do not need to edit this file at all.

Finally, run `mds-servicegroup-add` to register these configuration parameters to the Trigger Service:

```
mds-servicegroup-add $GLOBUS_LOCATION/etc/globus_wsrif_mds_trigger/trigger-registration-exa
```

Of course, if you copied the `trigger-registration-example.xml` file before editing it, you would use the name of the edited file instead. The output from `mds-servicegroup-add` should look something like this:

```
Processing configuration file...
INFO: Processed 1 registration entries
INFO: Successfully registered https://localhost:8443/wsrif/services/DefaultIndexService to
```

Warning

In general, it's a bad idea to use loopback addresses like "localhost" or "127.0.0.1" in MDS configuration files (because if you configure a remote host to poll "localhost" for information, the remote host will poll itself, not the host that the `mds-servicegroup-add` command was run from). We can get away with it here because all the addresses we're using are local, but in real life, it's better to use non-local IP addresses or fully-qualified domain names

4.4. Define Triggers

Now, we're going to define a trigger that checks how many Entries are being registered by the Index Service and then takes actions based on the results. But first, let's check how many Entries are being registered by the Index Service. Type the following command in one line (substituting your hostname and port if appropriate):

```
$GLOBUS_LOCATION/bin/wsrif-query -s https://127.0.0.1:8443/wsrif/services/DefaultIndexService
```

On our setup we get: 3.

4.5. Create A Trigger

At this point, the Trigger Service is collecting information, but we haven't told it to do anything with that information. We will now follow a few simple steps to set up a trigger.

1. First, we'll create the trigger:

```
$GLOBUS_LOCATION/bin/mds-trigger-create -b https://127.0.0.1:8443/wsrif/services https://
```

The first argument (`-b https://127.0.0.1:8443/wsrfl/services`) specifies the Trigger Service that we want to use. The second argument specifies which monitored resource we would like to act upon. (Note: in this tutorial, we're only monitoring one resource. But we could be monitoring several, by specifying several sets of `ServiceGroupRegistrationParameters` in the configuration file we used with `mds-servicegroup-add`, or by running `mds-servicegroup-add` multiple times with different configuration files).

The client should produce output similar to the following:

```
MDS4 Trigger Creation Client
-----

**      Service URL: https://127.0.0.1:8443/wsrfl/services/DefaultIndexService

Checking current monitored services (Trigger Registrations)...
OK
Address: https://128.9.64.191:8443/wsrfl/services/TriggerService
Reference property[0]:
<ns1:TriggerResourceKey xmlns:ns1="http://mds.globus.org/2007/03/TriggerResourceKey">54
...
--> Trigger has been created.
```

The `TriggerResourceKey` is an identifier created by the Trigger Service to identify this newly-created trigger. It is sometimes also referred to as a Trigger ID.

2. Now we have a trigger, but not a very interesting one. We can see some information about it by typing:

```
$GLOBUS_LOCATION/bin/mds-trigger-view -b https://127.0.0.1:8443/wsrfl/services
```

As above, the `-b https://127.0.0.1:8443/wsrfl/services` specifies the Trigger Service to use. If you specify a Trigger ID, then `mds-trigger-view` will display detailed information about that trigger. In this case, we didn't, so `mds-trigger-view` will display summary information about all triggers. The output should look something like the following:

```
MDS4 Trigger View Client
-----

Monitored Services (Trigger Registrations)

1) /wsrfl/services/DefaultIndexService

TriggerID:      546aae00-418b-11dd-a5ea-ebfac2dfbbee
TRIGGER NAME:   Default Trigger Name
MATCHING RULE:  count(//*[local-name()='Entry'])<1
ACTION SCRIPT:  trigger-action-default
TRIGGER STATUS: disabled
```

This gives us some important information:

- a shorthand reference to the service being monitored (the same one we specified in the `mds-trigger-create` command)
- the newly-created Trigger ID (which we will use in future requests to the Trigger Service)

- the matching rule (an XPath query that returns true if the number of Index Service entries is less than one)
 - the trigger action (if the condition specified in the matching rule is met, then the trigger will run the command mapped to the name `trigger-action-default` in the trigger service's `jndi-config.xml` file). This is a script that will append a line to the file `$GLOBUS_LOCATION/trigger_service_base_action_log`.
 - Most importantly, the trigger status is disabled, which means that the matching rule will not be evaluated, nor will the trigger action be run.
3. The `mds-trigger-edit` command is used to change the trigger's properties (its matching rule, action, enabled/disabled status, etc.). The syntax is:

```
mds-trigger-edit -b baseURL [TriggerID] [Parameter="Value"]
```

Let's enable this trigger. By enabling/activating the trigger, we turn it "on", meaning that it will take the Matching Rule and evaluate this against incoming aggregator data from our monitored service (the Default Index Service).

```
mds-trigger-edit -b https://127.0.0.1:8443/wsrf/services \
    546aae00-418b-11dd-a5ea-ebfac2dfbbee TriggerStatus=enabled
```

Now that this trigger is "enabled", we have an active trigger that is evaluating data. You may notice this in the service container logs if it is running in "debug" mode (You can allow "debug" information by uncommenting: `log4j.category.org.globus.mds.trigger=DEBUG` in your `$GLOBUS_LOCATION/container-log4j.properties` file). However, in a default Globus setup, the Matching Rule for this trigger always evaluates to false, so the trigger will not fire. (The action associated with this trigger appends a line to the log file `$GLOBUS_LOCATION/trigger_service_base_action_log`. You can verify that the trigger is not firing by checking that the file doesn't exist (or that if it does exist, that it hasn't been appended to recently).

Let's change our Matching Rule so that the trigger will evaluate to "true" and cause the trigger to fire.

```
mds-trigger-edit -b https://127.0.0.1:8443/wsrf/services \
    546aae00-418b-11dd-a5ea-ebfac2dfbbee \
    MatchingRule="count(//*[local-name()='Entry'])>0"
```

Typing `mds-trigger-view -b https://127.0.0.1:8443/wsrf/services` will summarize what we've done:

```
MDS4 Trigger View Client
```

```
-----  
Monitored Services (Trigger Registrations)
```

```
1) /wsrf/services/DefaultIndexService
```

```
TriggerID:      546aae00-418b-11dd-a5ea-ebfac2dfbbee  
TRIGGER NAME:   Default Trigger Name  
MATCHING RULE:  count(//*[local-name()='Entry'])>0  
ACTION SCRIPT:  trigger-action-default  
TRIGGER STATUS: enabled
```

To view more details about this particular trigger, type:

```
mds-trigger-view -b https://127.0.0.1:8443/wsrf/services \
    546aae00-418b-11dd-a5ea-ebfac2dfbbee
```

MDS4 Trigger View Client

-----Detailed Trigger Information-----

MONITORED SERVICE: https://127.0.0.1:8443/wsrf/services/DefaultIndexService
TriggerID: 546aae00-418b-11dd-a5ea-ebfac2dfbbee
TRIGGER NAME: Default Trigger Name

MATCHING RULE: count(//*[local-name()='Entry'])>0
ACTION SCRIPT: trigger-action-default
TRIGGER STATUS: enabled

ENABLE BOOLEAN: true
ACTION SCRIPT INPUT FULL ORIGINAL: true
ACTION SCRIPT INPUT XPATH QUERY RESULT: true

MINIMUM FIRING INTERVAL: 20
MINIMUM MATCH TIME: 30

START TIME: N/A
END TIME: N/A

INVALIDITY START TIME: N/A
INVALIDITY END TIME: N/A

-----Non-editable stats-----

RULE LAST CHECKED AT: 2008-06-23 19:09:18 PDT-0700
CONDITION TRUE SINCE: 2008-06-23 19:03:48 PDT-0700
ACTION FIRED AT: 2008-06-23 19:09:18 PDT-0700

Now after a minute or so, you will notice that the trigger has fired successfully. You can verify this by checking the contents of the log file we created in our action script from above:

```
more $GLOBUS_LOCATION/trigger_service_base_action_log
```

This should look similar to the following

```
Trigger Service Entry: Sun Jun 17 14:45:26 CDT 2007  
Trigger Service Entry: Sun Jun 17 14:45:56 CDT 2007  
Trigger Service Entry: Sun Jun 17 14:46:26 CDT 2007  
Trigger Service Entry: Sun Jun 17 14:46:56 CDT 2007
```

There is a 30 second interval that we specified in our aggregator configuration file above. This should probably be lengthened eventually so that you don't have the triggers going off so often.

4.6. Congratulations!

You have now successfully setup, configured, registered, created, edited and tested a trigger from scratch!

Next Steps: Check out the documentation and create more triggers to perform actions more relevant to your own objectives. Experiment with the XPath queries to expand the possibilities of what you can use them for. If you have questions, feel free to [\[fixme contact us\]](#)!

4.7. Troubleshooting

For MDS Trigger troubleshooting information, see [Troubleshooting MDS Trigger](#).

Chapter 2. MDS Trigger Commands

The `mds-servicegroup-add(1)` command in the Aggregator Framework is used to configure the Trigger Registration Service to collect information from various sources. In addition, the Trigger Service has three command-line clients

1. Create a new trigger - `mds-trigger-create`

Synopsis

```
mds-trigger-create [options] -b baseURL monitoredURL
```

Description

This command creates a new trigger.

Table 2.1. `mds-trigger-create` options

<i>-b baseURL</i>	Specify the trigger service's base URL (everything in the Trigger Service URL up to the service name). This option is used instead of the customary <code>-s</code> or <code>-e</code> options because this client communicates with more than one trigger-related service.
<i>monitoredURL</i>	Specify the URL of the service to be monitored; this should be the same as the address of a service registered to the Trigger Registry Service's service group.

Example

The first command creates a new trigger on the server *triggerhost.org* to monitor the information in the default Index Server running in the same Globus container. The second command creates a new trigger on the server *triggerhost.org* to monitor the information in an Index Server running on the server *otherhost.org*

```
mds-trigger-create -b https://triggerhost.org:8443/wsrf/services \
https://triggerhost.org:8443/wsrf/services/DefaultIndexService
```

```
mds-trigger-create -b https://triggerhost.org:8443/wsrf/services \
https://otherhost.org:8443/wsrf/services/DefaultIndexService
```

2. View information about existing triggers - `mds-trigger-view`

Synopsis

```
mds-trigger-view [options] -b baseURL [TriggerID]
```

Description

This displays information about triggers.

Table 2.2. mds-trigger-view options

<i>-b baseURL</i>	Specify the trigger service's base URL (everything in the Trigger Service URL up to the service name). This option is used instead of the customary <i>-s</i> or <i>-e</i> options because this client communicates with more than one trigger-related service.
<i>TriggerID</i>	If a Trigger ID is specified, detailed information about the specified trigger will be displayed; if not, summary information about all triggers will be displayed.

Example

The first command displays summary information about all triggers known to the Trigger Service; the second displays detailed information about one trigger

```
mds-trigger-view -b https://triggerhost.org:8443/wsrf/services
```

```
mds-trigger-view -b https://triggerhost.org:8443/wsrf/services \
546aae00-418b-11dd-a5ea-ebfac2dfbbee
```

3. Modify a trigger - mds-trigger-edit

Synopsis

```
mds-trigger-edit [options] -b baseURL TriggerID Parameter=Value
```

Description

This command is used to modify trigger parameters, in order to change the trigger conditions, actions, status (enabled or disabled), etc.

Table 2.3. mds-trigger-edit options

<i>-b baseURL</i>	Specify the trigger service's base URL (everything in the Trigger Service URL up to the service name). This option is used instead of the customary <i>-s</i> or <i>-e</i> options because this client communicates with more than one trigger-related service.
<i>TriggerID</i>	The identifier of the trigger to be modified
<i>Param=value</i>	Set the named parameter to the specified value. The parameter can be any writable Trigger Service resource property

Examples

The first command enables a trigger; the second command disables it.

```
mds-trigger-edit -b https://triggerhost.org:8443/wsrf/services \
546aae00-418b-11dd-a5ea-ebfac2dfbbee \
TriggerStatus=enabled
```

```
mds-trigger-edit -b https://triggerhost.org:8443/wsrf/services \
546aae00-418b-11dd-a5ea-ebfac2dfbbee \
TriggerStatus=disabled
```

Change the trigger condition (matching rule) so that the trigger fires if there are no Index Service entries.

```
mds-trigger-edit -b https://triggerhost.org:8443/wsrp/services \  
546aae00-418b-11dd-a5ea-ebfac2dfbbee \  
MatchingRule="count(//*[local-name()='Entry']=0"
```

Change the trigger action.

```
mds-trigger-edit -b https://triggerhost.org:8443/wsrp/services \  
546aae00-418b-11dd-a5ea-ebfac2dfbbee \  
ActionScript=trigger-action-input-default
```

Chapter 3. Graphical User Interface

The release contains WebMDS which can be used to display the status of resources registered to a Trigger Service in a normal web browser.

Chapter 4. Troubleshooting

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

1. No triggers displayed in mds-trigger-view

I created a trigger using the `mds-trigger-create`, but I don't see any triggers when I type `mds-trigger-view`! Where are the triggers? Why is nothing happening?

Did you set up a trigger registration? (See the Trigger Service [Admin Guide](#)) The trigger has been created (unless there was an error), but you will not see it and you cannot access it if the trigger registration has not been set up.

2. Trigger never fires

I'm sure the registration was made properly, but the trigger script never fires. OR I followed all of the above steps, but where are the triggers? Why is nothing happening?

Once you've completed the trigger registration, you can now create individual triggers. Trigger creation is performed using a client. See the [User's Guide](#) for more information on clients.

3. Error Messages

Table 4.1. WS MDS Trigger Service Error Messages

Error Code	Definition	Possible Solutions
Error ; nested exception is: org.apache.commons.httpclient.NoHttpResponseException: The server xxx.x.x.x failed to respond	Happens when trying to create a trigger for the Trigger Service. The above error is accompanied by the following error in container: [JWSCORE-192] Error processing request java.io.IOException: Token length 1347375956 > 33554432. FIXME - what causes this?	Be sure that you have properly edited the <code>client-config-settings</code> file under <code>globus_wsrf_mds_trigger</code> . The <code>DefaultServiceAddress</code> parameter should properly reflect the service prefix from your container, e.g.: <code>https://127.0.0.1:8444/wsrf/services/</code> . The services you wish to monitor should also be consistent.

WS MDS is built on Java WS Core, please see [Java WS Core Error Codes](#) for more error code documentation.

4. 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 lists¹ 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 Bugzilla⁴ 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/>

Index

C

command line clients
admin, 9

E

errors, 13

G

GUI
WebMDS, 12

T

troubleshooting, 13
tutorial
basic, 1

GT 4.2.1 WS MDS Trigger Service: Developer's Guide

GT 4.2.1 WS MDS Trigger Service: Developer's Guide

Introduction

The WS MDS Trigger Service collects information about Grid resources and can be configured to execute a program if the collected data meets certain conditions. This document describes the programmatic interfaces to the Trigger Service.

This document describes the programmatic interfaces to the Trigger Service. See also general Globus Toolkit [coding guidelines](#)¹ and [GT 4.2.1 best practices](#).

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

Table of Contents

Trigger Service How-tos	5
1. Before you Begin	1
1. Feature summary	1
2. Tested platforms	1
3. Backward compatibility summary	1
4. Technology dependencies	1
5. Security considerations	2
2. Usage scenarios	3
1. Controlling information collected by the Trigger Service	3
2. Controlling the Conditions Under Which the Executable is Executed	3
3. Programming the Executable	3
3. Trigger Service - Easy HowTo	4
1. Purpose	4
2. Prerequisites	4
3. Introduction	4
4. Trigger Tutorial	5
4. Tutorials	12
5. Architecture and design overview	13
6. APIs	14
1. Programming Model Overview	14
7. WS and WSDL	15
1. Protocol overview	15
2. Operations	15
3. WS MDS Aggregator Framework Resource Properties	16
4. Faults	16
5. WSDL and Schema Definition	16
8. Additional WSDL information for the Trigger Service	17
1. Trigger Service Resource Properties	17
9. MDS Trigger Commands	19
1. Create a new trigger - mds-trigger-create	19
2. View information about existing trggers - mds-trigger-view	19
3. Modify a trgger - mds-trigger-edit	20
10. Additional configuration for the Trigger Service	22
1. Additional configuration for the Trigger Service	22
11. Graphical User Interface	23
12. Trigger Action Script	24
1. Format of action script stdout	24
13. Aggregator sources	25
14. Debugging	26
1. Development Logging in Java WS Core	26
2. Enable Debug Logging in the Trigger Service	26
15. Troubleshooting	28
1. Java WS Core Errors	29
2. General troubleshooting information	31
16. Related Documentation	32
Glossary	33
Index	34

List of Tables

9.1. mds-trigger-create options	19
9.2. mds-trigger-view options	20
9.3. mds-trigger-edit options	20
15.1. Java WS Core Errors	30

Trigger Service

How-tos

A

- aggregator sources,
- AggregatorServiceGroup resource properties,
- apis,
- architecture,

C

- command line clients
 - admin,
- compatibility,
- configuration interface
 - Trigger Service-specific,
- configuring
 - Trigger Service-specific,

D

- debugging,
 - logging,
- dependencies,

E

- errors,

F

- features,

G

- GUI
 - WebMDS,

L

- logging
 - debugging,

P

- platforms,

R

- related documents,
- resource properties,
 - ActionFiredAt,
 - ActionScript,
 - ActionScriptInputFullOriginal,
 - ActionScriptInputXPathQueryResult,
 - ConditionTrueSince,

- EnableBoolean,
- EndTime,
- InvalidityEndTime,
- InvalidityStartTime,
- MatchingRule,
- MemberEPR,
- MinimumFiringInterval,
- MinimumMatchTime,
- NamespaceMappings,
- RuleLastCheckedAt,
- StartTime,
- TriggerID,
- TriggerName,
- TriggerStatus,
- resource properties, AggregatorServiceGroup,

S

- scripting the trigger,
- security considerations,
- services,

T

- trigger action script,
- troubleshooting
 - for developers,
- tutorial
 - basic,

U

- usage scenarios,
 - controlling information collected,
 - controlling when script/executable is fired,
 - programming script/executable,

W

- WSDL,

Chapter 1. Before you Begin

Before you begin:

1. Feature summary

The Trigger Service has been rewritten and restructured for the most recent 4.2.1 version to allow for a number of improvements including the following new features:

- Individual triggers may be created without restarting the container.
- Once created, triggers can be enabled/activated or disabled/deactivated without restarting the container.
- The parameters which define the trigger can be edited individually, in real-time, without restarting the container.

Other Supported Features

- Uses the *Aggregator Framework* to monitor XML data for matching trigger conditions
- When a trigger condition matches, fires a customizable action: for example, sends email to an administrator.
- Monitored services are managed through service group-based registration API, allowing use of many of the same clients that can be used in the *Index Service*.

Deprecated Features

- Not applicable

2. Tested platforms

Tested Platforms for WS MDS Trigger Service

- Linux on i386

Tested containers for WS MDS Trigger Service:

- Java WS Core container

3. Backward compatibility summary

The Trigger Service interfaces and underlying protocols are fully backward compatible with the GT 4.0.x version.

The Trigger Service interfaces and underlying protocols have changed since the GT 4.0 version. The Trigger Service will not interoperate with GT 4.0 servers or clients; however, trigger action programs written for previous versions of the Trigger Service should continue to work with this one.

4. Technology dependencies

The Trigger Service depends on the following GT components:

- Java WS Core

- Aggregator Framework

The Trigger Service depends on the following 3rd party software:

- None

5. Security considerations

The security considerations for the Aggregator Framework also apply to the Trigger Service:

5.1. 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 2. Usage scenarios

1. Controlling information collected by the Trigger Service

Information is collected by the Trigger Service by way of an *aggregator source*. The Globus Toolkit distribution includes several standard aggregator sources (see [Aggregator Sources Reference](#) for more details). To create your own custom information source, see the [Developer's Guide](#).

2. Controlling the Conditions Under Which the Executable is Executed

This is handled through configuration options (see [Additional configuration for the Trigger Service](#) for details).

3. Programming the Executable

The executable program triggered by the Trigger Service can be written in any programming or scripting language. The specifications for this program are documented in [Chapter 6, Configuring Execution Aggregator Source](#).

Chapter 3. Trigger Service - Easy HowTo

1. Purpose

The purpose of this Easy HowTo is to introduce the GT4/WS MDS component known as the Trigger, as well as to provide an example of setting one up successfully. The current GT 4.2.1 documentation provides a basic reference and will be updated as features are added, but for those of you who would like to get a simple trigger working without going through all of the documentation, this document is for you.

We will be creating a simple trigger from scratch, and setting it up completely. To get the basic idea of how this is done, we will only use elements available in the default GT4 installation to show you how to use triggers.

2. Prerequisites

To get the most out of this tutorial, you will need:

- A Globus Toolkit installation
- Some basic familiarity with [XML Path Language \(XPath\)](#).¹
- A valid X.509 user certificate

3. Introduction

The Trigger Service collects information and then performs actions based on that information. The Trigger Service works like this:

1. Administrators use a configuration file to specify the names and locations of *trigger actions*, programs that can be executed by the Trigger Service as a result of trigger conditions being met.
2. Administrators use a service interface to specify what information will be collected by the Trigger Service. This interface is called the Aggregator Framework and is the same configuration interface used by the Index Service
3. Users use a service interface to define *triggers*. A trigger includes (among other things) an XPath query string and the name of one of the trigger actions defined in step 1.
4. The Trigger Service periodically collects data (based on the configuration specified in step 2) and, for each trigger specified in step 3, evaluates the XPath query associated with the trigger and then executes the trigger's action if the query returns true.

In this example, we will configure the Trigger Service to monitor the Default Index Service running in the same Globus container, and then set up a trigger that will add an entry to a log file any time the number of Index Service is less than 1. This is not necessarily a practical example of how you would use a trigger, but it's simple enough to give you a basic idea of how to set one up. So let's get started!

¹ <http://www.w3.org/TR/xpath>

4. Trigger Tutorial

4.1. Preliminaries: Set Up Your Environment

First things first -- in order to run most Globus commands, you must have your environment set up correctly and have a valid proxy certificate. To set up your environment, first set the `GLOBUS_LOCATION` environment variable to the directory in which Globus is installed. To finish setting up your environment, run:

```
. $GLOBUS_LOCATION/etc/globus-user-env.sh
```

if you're a Bourne shell user, or

```
source $GLOBUS_LOCATION/etc/globus-user-env.csh
```

if you're a C shell user. Finally, generate a proxy certificate with:

```
$GLOBUS_LOCATION/bin/grid-proxy-init -verify -debug
```

4.2. Configure Trigger Action Programs

Next, we will specify what commands can be used in Trigger Service triggers. The Trigger Service comes with some simple action scripts in the `$GLOBUS_LOCATION/libexec/trigger` directory; we will edit the `$GLOBUS_LOCATION/etc/globus_wsrf_mds_trigger/jndi-config.xml` file to enable them:

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
  <global>
    <resource name="configuration"
              type="org.globus.mds.aggregator.impl.AggregatorConfiguration">
      ...
    </resource>

    <resource name="triggerConfiguration"
              type="org.globus.mds.trigger.impl.TriggerConfiguration">
      <resourceParams>
        <parameter>
          ...
        </parameter>
        <parameter>
          <name>executableMappings</name>
          <value>trigger-action-default=trigger-action-default.sh, trigger-action-input-def
        </parameter>
      </resourceParams>
    </resource>

  </global>

  <service name="TriggerRegistrationService">
    ...
```

```
</service>
...
</jndiConfig>
```

This `jndi-config.xml` file defines an `executableMappings` parameter. The format of the `executableMappings` parameter is a sequence of `name=value` strings, separated by commas. The left hand side of each `name/value` pair is the name that users will specify in trigger definitions; the right hand side is the path name (relative to the `$GLOBUS_LOCATION/libexec/trigger` directory) of the program to execute. In this example, we define two trigger actions: `trigger-action-default` maps to `$GLOBUS_LOCATION/libexec/trigger/trigger-action-default.sh`, and `trigger-action-input-default` maps to `$GLOBUS_LOCATION/libexec/trigger-action-input-default.sh`. These action scripts are distributed as part of the Globus distribution. The version of `$GLOBUS_LOCATION/etc/globus_wsrp/mds_trigger/jndi-config.xml` distributed with Globus has these mappings defined in a commented-out section; in order to continue with this example, you must uncomment them.

Before you continue, you'll need to restart your Globus container to make the changes to `jndi-config.xml` take effect. If you normally use `/etc/init.d/gt4container`, then you can type `/etc/init.d/gt4container restart`, or you can kill the running Globus container (if there is one) and run `$GLOBUS_LOCATION/etc/globus-start-container-detached` by hand. If you have a production container running and want to test the trigger service with a different instance on another port, you can run `globus-start-container-detached -p NNNN` to cause the new container to listen on port NNNN.

4.3. Configure the Trigger Service to Collect Information

The next thing we will do is configure the trigger service to collect some information (we will later configure the trigger service to periodically run a query on that information and, based on the results of the query, take some action). In this example, we will configure the trigger service to collect information by querying the Default Index Service running in the same Globus container for the entire contents of its index.

The Trigger Service uses the [Aggregator Framework](#) to configure its sources of information. Aggregator sources are configured through a service interface; we will create a file specifying configuration parameters and then run the `mds-servicegroup-add` command to read that configuration file and register the configuration information with the Trigger Service. We will start with the example trigger registration file included with Globus distributions in `$GLOBUS_LOCATION/etc/globus_wsrp/mds_trigger/trigger-registration-example.xml`

```
<DefaultServiceGroupEPR>
  <wsa:Address>https://localhost:8443/wsrp/services/TriggerRegistrationService</wsa:Address>
</DefaultServiceGroupEPR>
<ServiceGroupRegistrationParameters
  xmlns="http://mds.globus.org/servicegroup/client"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:agg="http://mds.globus.org/aggregator/types">
  <RegistrantEPR
    xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Address>https://localhost:8443/wsrp/services/DefaultIndexService</wsa:Address>
  </RegistrantEPR>
  <RefreshIntervalSecs>600</RefreshIntervalSecs>
  <Content xsi:type="agg:AggregatorContent"
    xmlns:agg="http://mds.globus.org/aggregator/types">
    <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
      <agg:GetResourcePropertyPollType
```

```
        xmlns:wssg="http://docs.oasis-open.org/wsr/sg-2">
        <agg:PollIntervalMillis>30000</agg:PollIntervalMillis>
        <agg:ResourcePropertyName>wssg:Entry</agg:ResourcePropertyName>
        </agg:GetResourcePropertyPollType>
    </agg:AggregatorConfig>
    <agg:AggregatorData />
</Content>
</ServiceGroupRegistrationParameters>
</ServiceGroupRegistrations>
```

For this example, the only items in the registration file that you might need to edit are the `DefaultServiceGroupEPR` (the address of your trigger registration service) and the `RegistrantEPR` (the address of the resource you want to monitor; in this case, your Default Index Service). If these look correct (i.e., the host names and port numbers correspond to the Globus container you just started), then you do not need to edit this file at all.

Finally, run `mds-servicegroup-add` to register these configuration parameters to the Trigger Service:

```
mds-servicegroup-add $GLOBUS_LOCATION/etc/globus_wsr/trigger-registration-exa
```

Of course, if you copied the `trigger-registration-example.xml` file before editing it, you would use the name of the edited file instead. The output from `mds-servicegroup-add` should look something like this:

```
Processing configuration file...
INFO: Processed 1 registration entries
INFO: Successfully registered https://localhost:8443/wsr/services/DefaultIndexService to
```

Warning

In general, it's a bad idea to use loopback addresses like "localhost" or "127.0.0.1" in MDS configuration files (because if you configure a remote host to poll "localhost" for information, the remote host will poll itself, not the host that the `mds-servicegroup-add` command was run from). We can get away with it here because all the addresses we're using are local, but in real life, it's better to use non-local IP addresses or fully-qualified domain names

4.4. Define Triggers

Now, we're going to define a trigger that checks how many Entries are being registered by the Index Service and then takes actions based on the results. But first, let's check how many Entries are being registered by the Index Service. Type the following command in one line (substituting your hostname and port if appropriate):

```
$GLOBUS_LOCATION/bin/wsr-query -s https://127.0.0.1:8443/wsr/services/DefaultIndexService
```

On our setup we get: 3.

4.5. Create A Trigger

At this point, the Trigger Service is collecting information, but we haven't told it to do anything with that information. We will now follow a few simple steps to set up a trigger.

1. First, we'll create the trigger:

```
$GLOBUS_LOCATION/bin/mds-trigger-create -b https://127.0.0.1:8443/wsr/services https://
```

The first argument (`-b https://127.0.0.1:8443/wsrfl/services`) specifies the Trigger Service that we want to use. The second argument specifies which monitored resource we would like to act upon. (Note: in this tutorial, we're only monitoring one resource. But we could be monitoring several, by specifying several sets of `ServiceGroupRegistrationParameters` in the configuration file we used with `mds-servicegroup-add`, or by running `mds-servicegroup-add` multiple times with different configuration files).

The client should produce output similar to the following:

```
MDS4 Trigger Creation Client
-----

**      Service URL: https://127.0.0.1:8443/wsrfl/services/DefaultIndexService

Checking current monitored services (Trigger Registrations)...
OK
Address: https://128.9.64.191:8443/wsrfl/services/TriggerService
Reference property[0]:
<ns1:TriggerResourceKey xmlns:ns1="http://mds.globus.org/2007/03/TriggerResourceKey">54
...
--> Trigger has been created.
```

The `TriggerResourceKey` is an identifier created by the Trigger Service to identify this newly-created trigger. It is sometimes also referred to as a Trigger ID.

2. Now we have a trigger, but not a very interesting one. We can see some information about it by typing:

```
$GLOBUS_LOCATION/bin/mds-trigger-view -b https://127.0.0.1:8443/wsrfl/services
```

As above, the `-b https://127.0.0.1:8443/wsrfl/services` specifies the Trigger Service to use. If you specify a Trigger ID, then `mds-trigger-view` will display detailed information about that trigger. In this case, we didn't, so `mds-trigger-view` will display summary information about all triggers. The output should look something like the following:

```
MDS4 Trigger View Client
-----

Monitored Services (Trigger Registrations)

1) /wsrfl/services/DefaultIndexService

TriggerID:      546aae00-418b-11dd-a5ea-ebfac2dfbbee
TRIGGER NAME:   Default Trigger Name
MATCHING RULE:  count(//*[local-name()='Entry'])<1
ACTION SCRIPT:  trigger-action-default
TRIGGER STATUS: disabled
```

This gives us some important information:

- a shorthand reference to the service being monitored (the same one we specified in the `mds-trigger-create` command)
- the newly-created Trigger ID (which we will use in future requests to the Trigger Service)

- the matching rule (an XPath query that returns true if the number of Index Service entries is less than one)
 - the trigger action (if the condition specified in the matching rule is met, then the trigger will run the command mapped to the name `trigger-action-default` in the trigger service's `jndi-config.xml` file). This is a script that will append a line to the file `$GLOBUS_LOCATION/trigger_service_base_action_log`.
 - Most importantly, the trigger status is disabled, which means that the matching rule will not be evaluated, nor will the trigger action be run.
3. The `mds-trigger-edit` command is used to change the trigger's properties (its matching rule, action, enabled/disabled status, etc.). The syntax is:

```
mds-trigger-edit -b baseURL [TriggerID] [Parameter="Value"]
```

Let's enable this trigger. By enabling/activating the trigger, we turn it "on", meaning that it will take the Matching Rule and evaluate this against incoming aggregator data from our monitored service (the Default Index Service).

```
mds-trigger-edit -b https://127.0.0.1:8443/wsrf/services \
    546aae00-418b-11dd-a5ea-ebfac2dfbbee TriggerStatus=enabled
```

Now that this trigger is "enabled", we have an active trigger that is evaluating data. You may notice this in the service container logs if it is running in "debug" mode (You can allow "debug" information by uncommenting: `log4j.category.org.globus.mds.trigger=DEBUG` in your `$GLOBUS_LOCATION/container-log4j.properties` file). However, in a default Globus setup, the Matching Rule for this trigger always evaluates to false, so the trigger will not fire. (The action associated with this trigger appends a line to the log file `$GLOBUS_LOCATION/trigger_service_base_action_log`. You can verify that the trigger is not firing by checking that the file doesn't exist (or that if it does exist, that it hasn't been appended to recently).

Let's change our Matching Rule so that the trigger will evaluate to "true" and cause the trigger to fire.

```
mds-trigger-edit -b https://127.0.0.1:8443/wsrf/services \
    546aae00-418b-11dd-a5ea-ebfac2dfbbee \
    MatchingRule="count(//*[local-name()='Entry'])>0"
```

Typing `mds-trigger-view -b https://127.0.0.1:8443/wsrf/services` will summarize what we've done:

```
MDS4 Trigger View Client
```

```
-----  
Monitored Services (Trigger Registrations)
```

```
1) /wsrf/services/DefaultIndexService
```

```
TriggerID:      546aae00-418b-11dd-a5ea-ebfac2dfbbee  
TRIGGER NAME:   Default Trigger Name  
MATCHING RULE:  count(//*[local-name()='Entry'])>0  
ACTION SCRIPT:  trigger-action-default  
TRIGGER STATUS: enabled
```

To view more details about this particular trigger, type:

```
mds-trigger-view -b https://127.0.0.1:8443/wsrf/services \
    546aae00-418b-11dd-a5ea-ebfac2dfbbee
```

MDS4 Trigger View Client

-----Detailed Trigger Information-----

MONITORED SERVICE: https://127.0.0.1:8443/wsrf/services/DefaultIndexService
TriggerID: 546aae00-418b-11dd-a5ea-ebfac2dfbbee
TRIGGER NAME: Default Trigger Name

MATCHING RULE: count(//*[local-name()='Entry'])>0
ACTION SCRIPT: trigger-action-default
TRIGGER STATUS: enabled

ENABLE BOOLEAN: true
ACTION SCRIPT INPUT FULL ORIGINAL: true
ACTION SCRIPT INPUT XPATH QUERY RESULT: true

MINIMUM FIRING INTERVAL: 20
MINIMUM MATCH TIME: 30

START TIME: N/A
END TIME: N/A

INVALIDITY START TIME: N/A
INVALIDITY END TIME: N/A

-----Non-editable stats-----

RULE LAST CHECKED AT: 2008-06-23 19:09:18 PDT-0700
CONDITION TRUE SINCE: 2008-06-23 19:03:48 PDT-0700
ACTION FIRED AT: 2008-06-23 19:09:18 PDT-0700

Now after a minute or so, you will notice that the trigger has fired successfully. You can verify this by checking the contents of the log file we created in our action script from above:

```
more $GLOBUS_LOCATION/trigger_service_base_action_log
```

This should look similar to the following

```
Trigger Service Entry: Sun Jun 17 14:45:26 CDT 2007  
Trigger Service Entry: Sun Jun 17 14:45:56 CDT 2007  
Trigger Service Entry: Sun Jun 17 14:46:26 CDT 2007  
Trigger Service Entry: Sun Jun 17 14:46:56 CDT 2007
```

There is a 30 second interval that we specified in our aggregator configuration file above. This should probably be lengthened eventually so that you don't have the triggers going off so often.

4.6. Congratulations!

You have now successfully setup, configured, registered, created, edited and tested a trigger from scratch!

Next Steps: Check out the documentation and create more triggers to perform actions more relevant to your own objectives. Experiment with the XPath queries to expand the possibilities of what you can use them for. If you have questions, feel free to [\[fixme contact us\]](#)!

4.7. Troubleshooting

For MDS Trigger troubleshooting information, see [Troubleshooting MDS Trigger](#).

Chapter 4. Tutorials

There are no tutorials available at this time.

Chapter 5. Architecture and design overview

The Trigger Service collects information and acts on it, by executing an administrator-supplied executable program when certain conditions (expressed as XPath matches on the collected information) are met. The Trigger Registration Service first "registers" itself with a monitored service. Then individual triggers are created to act on data aggregated from that monitored service.

There are command-line clients designed to allow one to easily create, edit, and view the triggers.

Chapter 6. APIs

1. Programming Model Overview

Information about how to configure existing aggregator sources (such as the aggregator sources distributed with the Globus Toolkit, which include one that polls for resource property information, one that collects resource property information through subscription/notification, and one that collects information by executing an executable program) is found in [Aggregator Sources Reference](#); information about how to create new aggregator sources can be found in [Developer's Guide](#).

The administrator of a Globus installation configures the set of available executable programs that are available to be used as action scripts (for example, an executable program may send mail to an end-user or write a structured log file that will later be read by some other program).

Chapter 7. 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

Chapter 8. Additional WSDL information for the Trigger Service

1. Trigger Service Resource Properties

In addition to the resource properties for the Aggregator Framework, the Trigger Service exposes the following:

TriggerName	This resource property allows one to arbitrarily name the trigger. This is used to assist one in managing many triggers.
TriggerStatus	This resource property is used to indicate the current status of the trigger. There are two states allowed: <code>enabled</code> and <code>disabled</code> .
MemberEPR	This resource property reveals the monitored service that the trigger is associated with
TriggerID	This resource property is a unique ID assigned to the trigger. It is essentially the EPR's Resource Key.
MatchingRule	This resource property is the XPath expression that will be used in evaluating incoming aggregator data. The trigger will fire (if enabled) if the expression is "true" (in a boolean sense). But if "EnableBoolean" is set to "false", then if the MatchingRule returns any data, the trigger will fire. This is consistent with pre-4.2 trigger functionality.
NamespaceMappings	This resource property allows one to use namespaces in the MatchingRule.
ActionScript	This resource property is the name of the action script that should be fired when the trigger evaluation is "true". The action script is located in the <code>\$GLOBUS_LOCATION/libexec/trigger/</code> directory.
EnableBoolean	This resource property is by default <code>true</code> , meaning that it is set up to evaluate XPath queries as "true" or "false", firing only when "true". If this property is set to "false", then the trigger will fire only if the MatchingRule evaluation returns any data
MinimumFiringInterval	The action script will not be executed more than once in this number of seconds. If unspecified, there will be no minimum interval.
MinimumMatchTime	The MatchingRule must be true for this number of seconds before the ActionScript will be executed. If unspecified, there is no minimum time period that the rule must match and the rule will be eligible to fire immediately after the MatchingRule becomes true.
StartTime	The trigger will not fire, nor will the TriggerService perform any evaluations before the StartTime, if indicated. If a start time is not indicated, the TriggerService will begin immediately performing evaluations, if the trigger is active (i.e. TriggerStatus is set to "enabled")
EndTime	The TriggerService will not perform any evaluations after the EndTime, if indicated. If an end time is not indicated, the TriggerService will continue performing evaluations (of "active" triggers) until an EndTime is specified, otherwise until

the container is shutdown. After an `EndTime` has passed, the `TriggerService` will basically be doing nothing, but you may whenever you wish change the `EndTime`, and the trigger evaluations will then begin again until the `EndTime` again is reached.

InvalidityStartTime The trigger will not fire, nor will the `TriggerService` perform any evaluations after the `InvalidityStartTime`, if indicated. If an `InvalidityStartTime` is indicated, an `InvalidityEndTime` must also be specified. During this time period, the `TriggerService` will not perform any evaluations, if the trigger is active (i.e. `TriggerStatus` is set to "enabled")

InvalidityEndTime This parameter requires an `InvalidityStartTime`, and during the time period between the `InvalidityStartTime` and `InvalidityEndTime`, the `TriggerService` will not perform any evaluations. If there is an `EndTime` specified, then trigger evaluations will begin after the `InvalidityEndTime` until the `EndTime`.

ActionScriptInputFullOriginal This parameter, if set to "true" will pass the original trigger message input (to which the matching rule was applied) to the action script. The default behavior is to always pass the entire input message to the action scripts. For action scripts which do not need to consume the unmodified input, this variable may be set to "false" in order to increase performance. For users familiar with previous versions of the `Trigger Service`, if you set `ActionScriptInputFullOriginal` to "true", this is equivalent to setting `disableUnmodifiedActionScriptInput` to "false", in other words it will pass the original trigger message input (to which the matching rule was applied) to the action script.

ActionScriptInputXPathQueryResult If this value is present and set to true, the service will pass the filtered output result of the `XPath MatchingRule` as additional input to the `stdin` of the action script, in addition to the original input to which the matching rule was applied. The default behavior if unspecified is true, meaning the `Xpath` query result will be passed as input to the action script. For users familiar with previous versions of the `Trigger Service`, if you set `ActionScriptInputXPathQueryResult` to "true", this is equivalent to setting `enableFilteredActionScriptInput` to "true".

The following resource properties are not editable; they are trigger run-time statistics.

RuleLastCheckedAt This resource property reveals when the `MatchingRule` was last checked/evaluated.

ConditionTrueSince This resource property reports since when the `MatchingRule` evaluated against the incoming aggregator data results in `true`

ActionFiredAt This resource property reveals exactly when the trigger was last fired.

Chapter 9. MDS Trigger Commands

The `mds-servicegroup-add(1)` command in the Aggregator Framework is used to configure the Trigger Registration Service to collect information from various sources. In addition, the Trigger Service has three command-line clients

1. Create a new trigger - `mds-trigger-create`

Synopsis

```
mds-trigger-create [options] -b baseURL monitoredURL
```

Description

This command creates a new trigger.

Table 9.1. `mds-trigger-create` options

<i>-b baseURL</i>	Specify the trigger service's base URL (everything in the Trigger Service URL up to the service name). This option is used instead of the customary <code>-s</code> or <code>-e</code> options because this client communicates with more than one trigger-related service.
<i>monitoredURL</i>	Specify the URL of the service to be monitored; this should be the same as the address of a service registered to the Trigger Registry Service's service group.

Example

The first command creates a new trigger on the server *triggerhost.org* to monitor the information in the default Index Server running in the same Globus container. The second command creates a new trigger on the server *triggerhost.org* to monitor the information in an Index Server running on the server *otherhost.org*

```
mds-trigger-create -b https://triggerhost.org:8443/wsrf/services \
https://triggerhost.org:8443/wsrf/services/DefaultIndexService
```

```
mds-trigger-create -b https://triggerhost.org:8443/wsrf/services \
https://otherhost.org:8443/wsrf/services/DefaultIndexService
```

2. View information about existing triggers - `mds-trigger-view`

Synopsis

```
mds-trigger-view [options] -b baseURL [TriggerID]
```

Description

This displays information about triggers.

Table 9.2. mds-trigger-view options

<i>-b baseURL</i>	Specify the trigger service's base URL (everything in the Trigger Service URL up to the service name). This option is used instead of the customary <i>-s</i> or <i>-e</i> options because this client communicates with more than one trigger-related service.
<i>TriggerID</i>	If a Trigger ID is specified, detailed information about the specified trigger will be displayed; if not, summary information about all triggers will be displayed.

Example

The first command displays summary information about all triggers known to the Trigger Service; the second displays detailed information about one trigger

```
mds-trigger-view -b https://triggerhost.org:8443/wsrf/services
```

```
mds-trigger-view -b https://triggerhost.org:8443/wsrf/services \
546aae00-418b-11dd-a5ea-ebfac2dfbbee
```

3. Modify a trigger - mds-trigger-edit

Synopsis

```
mds-trigger-edit [options] -b baseURL TriggerID Parameter=Value
```

Description

This command is used to modify trigger parameters, in order to change the trigger conditions, actions, status (enabled or disabled), etc.

Table 9.3. mds-trigger-edit options

<i>-b baseURL</i>	Specify the trigger service's base URL (everything in the Trigger Service URL up to the service name). This option is used instead of the customary <i>-s</i> or <i>-e</i> options because this client communicates with more than one trigger-related service.
<i>TriggerID</i>	The identifier of the trigger to be modified
<i>Param=value</i>	Set the named parameter to the specified value. The parameter can be any writable Trigger Service <u>resource property</u>

Examples

The first command enables a trigger; the second command disables it.

```
mds-trigger-edit -b https://triggerhost.org:8443/wsrf/services \
546aae00-418b-11dd-a5ea-ebfac2dfbbee \
TriggerStatus=enabled
```

```
mds-trigger-edit -b https://triggerhost.org:8443/wsrf/services \
546aae00-418b-11dd-a5ea-ebfac2dfbbee \
TriggerStatus=disabled
```

Change the trigger condition (matching rule) so that the trigger fires if there are no Index Service entries.

```
mds-trigger-edit -b https://triggerhost.org:8443/wsrp/services \  
546aae00-418b-11dd-a5ea-ebfac2dfbbee \  
MatchingRule="count(//*[local-name()='Entry'])=0"
```

Change the trigger action.

```
mds-trigger-edit -b https://triggerhost.org:8443/wsrp/services \  
546aae00-418b-11dd-a5ea-ebfac2dfbbee \  
ActionScript=trigger-action-input-default
```

Chapter 10. Additional configuration for the Trigger Service

1. Additional configuration for the Trigger Service

The set of possible actions (programs) that can be executed by the Trigger Service is specified in the file `$GLOBUS_LOCATION/etc/globus_wsrft_mds_trigger/jndi-config.xml`. The `executableMappings` parameter contains a comma-separated list of name=value pairs. The left hand side of each name/value pair is the name assigned to the trigger action; this name can be used in trigger definitions. The right hand side of each name/value pair is the path name (relative to `$GLOBUS_LOCATION/libexec/trigger`) of the file to execute.

The sources of information used by the Trigger Service are configured using the `mds-servicegroup-add` command; see the [Aggregator Framework](#) documentation or the [Trigger Service Easy HOWTO](#) for more details and examples.

Triggers themselves are created using the [command line clients](#).

Chapter 11. Graphical User Interface

The release contains WebMDS which can be used to display the status of resources registered to a Trigger Service in a normal web browser.

Chapter 12. Trigger Action Script

1. Format of action script stdout

The action script should output an XML document to stdout. The xml document does not need to match any particular schema. This output will be included in the ServiceGroupEntry for the rule.

Chapter 13. Aggregator sources

The public interfaces for creating and configuring aggregator sources -- sources of information used by the trigger service -- can be found in [Aggregator Sources Reference](#).

Chapter 14. 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 6, Debugging](#).

1. Development Logging in Java WS Core

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

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

1.1. Configuring server side developer logs

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

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

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

1.2. Configuring client side developer logs

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

2. Enable Debug Logging in the Trigger Service

To turn on debug logging for the Trigger Service, add the line:

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

to the appropriate properties file. Since the Trigger Service is implemented using the [Aggregator Framework](#), you may also want to turn on aggregator debugging by adding this line:

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>

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

Chapter 15. Troubleshooting

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

1. Java WS Core Errors

Table 15.1. 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:154) at org.apache.coyote.tomcat5.CoyoteRequestFacade.setAttribute(CoyoteRequestFacade.java:154) at org.globus.tomcat.coyote.valves.HTTPSValve.expose(HTTPSVAlve.java:154)</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 16. Related Documentation

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

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 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.

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 Group¹⁴ for details.

¹⁴ <http://www.w3.org/2002/ws/addr/>

Index

A

- aggregator sources, 25
- AggregatorServiceGroup resource properties, 16
- apis, 14
- architecture, 13

C

- command line clients
 - admin, 19
- compatibility, 1
- configuration interface
 - Trigger Service-specific, 22
- configuring
 - Trigger Service-specific, 22

D

- debugging, 26
 - logging, 26
- dependencies, 1

E

- errors, 29

F

- features, 1

G

- GUI
 - WebMDS, 23

L

- logging
 - debugging, 26

P

- platforms, 1

R

- related documents, 32
- resource properties, 17
 - ActionFiredAt, 18
 - ActionScript, 17
 - ActionScriptInputFullOriginal, 18
 - ActionScriptInputXPathQueryResult, 18
 - ConditionTrueSince, 18
 - EnableBoolean, 17
 - EndTime, 17
 - InvalidityEndTime, 18

- InvalidityStartTime, 18
- MatchingRule, 17
- MemberEPR, 17
- MinimumFiringInterval, 17
- MinimumMatchTime, 17
- NamespaceMappings, 17
- RuleLastCheckedAt, 18
- StartTime, 17
- TriggerID, 17
- TriggerName, 17
- TriggerStatus, 17
- resource properties, AggregatorServiceGroup, 16

S

- scripting the trigger, 24
- security considerations, 2
- services, 15

T

- trigger action script, 24
- troubleshooting
 - for developers, 28
- tutorial
 - basic, 4

U

- usage scenarios, 3
 - controlling information collected, 3
 - controlling when script/executable is fired, 3
 - programming script/executable, 3

W

- WSDL, 15, 17

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_wsrf_1.0.pdf)¹.

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

- None known.

4. Bug Fixes

- Not applicable.

5. Performance reports

- None available at this time.

GT 4.2.1 Release Notes: WS MDS Trigger Service

Table of Contents

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

<titleabbrev>Release Notes</titleabbrev>

1. Component Overview

The Trigger Service collects data from resources on the grid and, if administrator defined rules match, can perform various actions. An example use is to send email when queue length on a compute resource goes over a threshold value.

2. Feature summary

The Trigger Service has been rewritten and restructured for the most recent 4.2.1 version to allow for a number of improvements including the following new features:

- Individual triggers may be created without restarting the container.
- Once created, triggers can be enabled/activated or disabled/deactivated without restarting the container.
- The parameters which define the trigger can be edited individually, in real-time, without restarting the container.

Other Supported Features

- Uses the *Aggregator Framework* to monitor XML data for matching trigger conditions
- When a trigger condition matches, fires a customizable action: for example, sends email to an administrator.
- Monitored services are managed through service group-based registration API, allowing use of many of the same clients that can be used in the *Index Service*.

Deprecated Features

- Not applicable

3. Summary of Changes in WS MDS Trigger Service

No new changes in this release

4. Bug Fixes

- Not applicable.

5. Known Problems

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

5.1. Limitations

- None known

5.2. Outstanding bugs

- None known.

6. Technology dependencies

The Trigger Service depends on the following GT components:

- [Java WS Core](#)
- [Aggregator Framework](#)

The Trigger Service depends on the following 3rd party software:

- None

7. Tested platforms

Tested Platforms for WS MDS Trigger Service

- Linux on i386

Tested containers for WS MDS Trigger Service:

- Java WS Core container

8. Backward compatibility summary

The Trigger Service interfaces and underlying protocols are fully backward compatible with the GT 4.0.x version.

The Trigger Service interfaces and underlying protocols have changed since the GT 4.0 version. The Trigger Service will not interoperate with GT 4.0 servers or clients; however, trigger action programs written for previous versions of the Trigger Service should continue to work with this one.

9. Associated Standards

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

10. For More Information

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

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.