

**GT 4.2.0 WS Monitoring and Discovery
Services (WS MDS) : System
Administrator's Guide**

DRAFT

GT 4.2.0 WS Monitoring and Discovery Services (WS MDS) : System Administrator's Guide

Introduction

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

Important

This information is in addition to the basic Globus Toolkit prerequisite, overview, installation, security configuration instructions in the [Installing GT 4.2.0](#). Read through this guide before continuing!

DRAFT

Table of Contents

MDS Admin Howtos	5
1. Building and Installing	1
2. Configuring	2
1. Configuration overview	2
2. Default configuration	2
3. Getting started	4
3. Deploying	7
1. Deploying MDS into a Virtual Organization (VO)	7
4. Advanced Configuration	8
1. Registering a WSRF service to an Index Service	8
2. Registering any Grid resource via Information Providers	13
5. Testing	17
6. Security Considerations	18
7. Debugging	19
1. Logging in Java WS Core	19
8. Troubleshooting	21
1. Java WS Core Errors	22
Index	25

List of Tables

4.1. Aggregator configuration parameters	10
8.1. Java WS Core Errors	23

DRAFT

MDS Admin Howtos

A

- aggregator sources,
 - execution,
 - query,
 - subscription,

B

- building,
- building a hierarchy of Index Services,

C

- configuration file, registering
 - Content block options,
 - AggregatorSubscriptionType,
 - ExecutionPollType,
 - GetMultipleResourcePropertiesPollType,
 - GetResourcePropertyPollType,
 - QueryResourcePropertiesPollType,
- example-aggregator-registration.xml,
- parameters,
- configuring
 - default,
 - overview,

D

- debugging
 - logging,
- deploying,
 - in a Virtual Organization (VO),
 - typical,
- deployment
 - typical,

E

- EPR,
- errors,

H

- hierarchy of Index Services,
- hierarchy.xml,

I

- Index Service,
- information providers,
 - clusters,
 - Condor pools,
 - configuring, Aggregator,
 - configuring, UsefulRP,

- default,
- GLUE 1.1 schema,
- installing,

J

- jndi-config.xml,

L

- logging
 - CEDPS-compliant,
 - debugging,

M

- mds-servicegroup-add,

R

- registering
 - automatic, of GT WS components,
- registering to an Index Service
 - any Grid resource,
 - WSRF service,
 - query resource properties via XPath,
 - request multiple resource properties,
 - request single resource properties via XPath,
 - request single resource property,
 - subscribe to a Topic,
- registering with the default Index Service,
- registration
 - automatic, of GT WS components,

T

- testing,
- Trigger Service,

U

- upstream,
- upstream.xml,
- usage
 - simple,
- use cases
 - admin,
- using
 - basic,

W

- WebMDS,
- wsrp-query,

Chapter 1. Building and Installing

MDS is built and installed as part of a default GT 4.2.0 installation. See [Installing GT 4.2.0](#) for more information.

DRAFT

Chapter 2. Configuring

1. Configuration overview

MDS is a collection of monitoring services and related interfaces that collect and act on recent state information from information sources in a Grid. It is based on WSRF and WS-Notification specifications and built on the Java WS Core. You should be familiar with [configuring information for Java WS Core](#) and have a basic understanding of WSRF and web services, especially resource properties.

MDS includes an Index Service, which collects information and then advertises it as resource properties (which can then be queried or subscribed to using standard WSRF and WS-N interfaces) and a Trigger Service, which collects information and can be configured to perform actions when certain user-specified conditions are met. The Index and Trigger Services are built upon a common framework, the Aggregator Framework, which handles the collection of data. The interface for configuring how these services collect their information is the same.

1.1. Admin use cases

Some common administrative uses of WS MDS:

- Use the MDS Index Service to monitor certain GT WSRF services (this is the default).
- Chain MDS Index Services to create a hierarchy throughout a VO, with information aggregated in a central Index Service.
- Use MDS Trigger Service to send an email when a condition on a grid resource is met (such as being notified when a service goes down).

2. Default configuration

After a default installation of GT version, the container includes a default Index Service, which should be visible when starting the container, like the following:

```
...  
https://localhost:8443/wsrf/services/DefaultIndexService  
...
```

These WSRF-based services automatically register themselves with the default Index Service running in the same Globus container:

- [GRAM4](#)
- [Community Authorization Service \(CAS\)](#)
- [Reliable File Transfer \(RFT\)](#)

Check the links to see what information is captured for each service and how to configure that information.

This means that without any further configuration, the default Index Service is already monitoring these services.

2.1. Simple usage

A typical example of using the default Index Service is with the [wsrf-query](#) Java WS Core command. For example:

```
$GLOBUS_LOCATION/bin/wsrf-query -s https://localhost:8443/wsrf/services/DefaultIndexService
```

displays all the resource properties collected by the default Index Service on your local host.

You can also use an XPath query to drill down your search as well as other Java WS Core commands such as [wsrf-get-property](#) and [wsrf-get-properties](#). For more information, review the [User's Guide](#).

2.2. Toggle automatic registration of GT WS components

To turn off (or on, depending on your situation) automatic registration of these components, edit the `jndi-config.xml` file of the component.

For example, the jndi config file for GRAM4 is at `$GLOBUS_LOCATION/etc/gram-service/jndi-config.xml`.

Find the following JNDI resource:

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

To configure the automatic registration of GRAM4 to the default MDS Index Service, change the value of the parameter `<reg>` as follows:

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

2.3. Overview of a typical deployment

A typical deployment of MDS will enable a project to:

- discover needed data from services in order to make job submission or replica selection decisions;
- evaluate the status of Grid services for a project testbed;
- be notified when disks are full or other error conditions happen;
- visualize the state of services.

MDS should be deployed in a distributed fashion. Some components should be deployed central to a VO, while others should be deployed on individual resources.

In order to deploy a project or VO-wide WS MDS setup, we recommend the following steps (which we will expand on later):



Note

The services deployed do not need to be run on the same host or be at the same location.

1. Set up a hierarchy of Indexes with one as the central (VO) index to which all other Index Services are registered (see [Section 3.1, “Building a Hierarchy of DefaultIndexServices”](#) below):
 - If you want to set up a site-wide Index Service with all services and resources for the project at that site registered to it, including those provided by Ganglia or Hawkeye, please see [Section 1, “Registering a WSRF service to an Index Service”](#) and [Section 2.3, “Configuring custom information providers”](#) under 'Advanced Configuration'.
 - If you want to deploy an Index Service to act as the centralized data source for the VO to collect information about all of the resources in the VO, please see [Section 1, “Deploying MDS into a Virtual Organization \(VO\)”](#) (in addition to [Section 3.1, “Building a Hierarchy of DefaultIndexServices”](#)).
2. Set up WebMDS to view the contents of a central Index Service in a web browser (please see [Section 3.2, “Visualizing Index Service with WebMDS”](#) below).
3. Deploy a Trigger Service notification script to alert interested parties about changes in the status of the VO (please see [Section 3.3, “Setting up email notifications via the Trigger Service”](#)).

3. Getting started

This section will show you how to set up a basic hierarchy that monitors other GT web services (in [Chapter 4, Advanced Configuration](#), you will learn how to monitor almost any Grid resource). You will learn how to create a hierarchy of index services, use WebMDS to monitor from a web browser and set up a Trigger Service to be notified if any of the services go down.

3.1. Building a Hierarchy of DefaultIndexServices

This example shows how to configure the DefaultIndexService so that it will register itself to an upstream DefaultIndexService. By chaining such upstream registrations between hosts, it is possible to build a hierarchical virtual organization with a complete aggregate data view at the root level.

- 1. Change to the `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index` directory.
- 2. Open the file `hierarchy.xml` and add one or more upstream entries using the URL to DefaultIndexService(s) you wish to register to. Example syntax for an upstream entry is as follows:

```
<upstream>https://vo-host:8443/wsrf/services/DefaultIndexService</upstream>
```

Please note that if a secure container is targeted, you must set the correct service URL (https protocol, right port number) in the upstream entry. Note: The defaultIndexService(s) you enter here will be higher in the hierarchy than the current one

- 2.a (This step is optional) Change the default upstream registration settings (if desired) in the `upstream.xml` file:

```
<ServiceGroupRegistrationParameters  
  xmlns="http://mds.globus.org/servicegroup/client" >
```

```

<!-- Specifies that the registration will be renewed every 600
seconds (= 10 minutes) -->
<RefreshIntervalSecs>600</RefreshIntervalSecs>

<Content xsi:type="agg:AggregatorContent"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:agg="http://mds.globus.org/aggregator/types">

  <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
    <agg:GetResourcePropertyPollType
      xmlns:wssg="http://docs.oasis-open.org/wsrf/sg-2">
      <!-- Specifies that the upstream index should refresh information
every 10 minutes -->
      <agg:PollIntervalMillis>600000</agg:PollIntervalMillis>

      <!-- specified that the upstream index should collect the
Entry resource properties from this index -->
      <agg:ResourcePropertyName>wssg:Entry</agg:ResourcePropertyName>

    </agg:GetResourcePropertyPollType>
  </agg:AggregatorConfig>
  <agg:AggregatorData/>
</Content>

</ServiceGroupRegistrationParameters>

```

The two settings of note here are the *RefreshIntervalSecs* and the *PollIntervalMillis* parameters.

RefreshIntervalSecs This parameter specifies that the local *DefaultIndexService* will attempt to refresh the registration made to the upstream *DefaultIndexService* every (n) seconds. If the upstream *DefaultIndexService* is a high level or root-level member of a VO hierarchy, or is intended to support a large number of registrants, it may be desirable to set the refresh interval to be a larger value so that the overall amount of registration traffic is reduced. Note that setting the *RefreshIntervalSecs* to a larger value will not affect the relative "freshness" of the aggregated data from each registrant; rather, this is controlled by the *PollIntervalMillis* parameter of the *GetResourcePropertyPollType* element.

PollIntervalMillis The *GetResourcePropertyPollType* element contained within the *AggregatorConfig* element specifies that the upstream *DefaultIndexService* should poll the local *DefaultIndexService* "Entry" resource property every (n) milliseconds, based on the value of this parameter. Alter this parameter in order to control the interval upon which data from the downstream (local) *DefaultIndexService* is refreshed in the upstream *DefaultIndexService*.

- 3. Start the container by running:

```
$ ./bin/globus-start-container
```

When the *DefaultIndexService* is first activated, it will read the contents of the *hierarchy.xml* file and attempt to make an upstream registration for each upstream entry found within that file. The service will make each registration

according to the parameters found in the `upstream.xml` file. Any errors or warnings during the registration process will be written to the container log. If successful, the "Entry" resource property of the upstream Index will contain a child element for the downstream (local) Index, and a copy of the downstream Index "Entry" resource property will be cached within that element.

3.2. Visualizing Index Service with WebMDS

WebMDS is built and installed as part of a default GT installation and you only need to deploy the servlet to view the contents of the `DefaultIndexService` (in the local container) via a web browser.

To visualize the Index Service:

- Deploy the servlet into a servlet container such as Tomcat. For more information see the [Deploying section in the WebMDS Admin Guide](#).
- Point your web browser at `http://your-tomcat-host:your-tomcat-port/webmds`
- Click on the link labelled "A list of resources registered to the local default index service".

For more detailed information about changing the look of WebMDS and more advanced configuration, see the [WebMDS Admin Guide](#).

3.3. Setting up email notifications via the Trigger Service

Deploy the Trigger Service to notify interested parties about certain configured changes in the status of the resources that an Index Service is monitoring. This can be useful for receiving emails about a service going down, etc. For a basic tutorial to get you started with the Trigger Service, see [Trigger Service - Easy HowTo](#).

Chapter 3. Deploying

Because MDS is built on top of Java WS Core, make sure you are familiar with the deployment information [here](#).

1. Deploying MDS into a Virtual Organization (VO)

MDS should be deployed in a distributed fashion. Some components should be deployed central to a VO, while others should be deployed on individual resources. This document is a suggested deployment, and is not the only way in which MDS components can be deployed.

1.1. What to deploy where

The following diagrams outlines a VO-wide deployment:

[FIXME better diagram]

1.2. Central to the VO

1. The following steps are VO-wide, although the services deployed do not need to be run on the same host or be at the same location.
 - Deploy an index service central to the VO to collect information about all of the resources in the VO.
 - If your VO is part of a larger VO, register the central index service of your VO to the central index of the larger VO.
 - Install the WebMDS servlet to show the contents of the central index service in a web browser.
 - Deploy a Trigger Service to notify interested parties about changes in the status of the VO.
2. On each GRAM4 installation
 - Configure cluster monitoring at each GRAM installation to publish information about each resource. [FIXME more details about how schedulers do not need config?]

Chapter 4. Advanced Configuration

Now that you have a basic hierarchy of default Index Services monitoring the other GT services, you will most likely want to monitor other resources from your Grid. The following sections show how to register WSRF services as well as other non-WSRF resources.

1. Registering a WSRF service to an Index Service

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

- Decide how you want to get information into the Index Service. The following correspond to common Java WS Core user commands (in fact, you can use those commands ahead of time to 'test' what information will end up in the Index Service):
 - Request single resource property: **wsrf-get-property**
 - Request one or more resource properties: **wsrf-get-properties**
 - Query resource properties with XPath: **wsrf-query** or **globus-xpath-query**
 - Subscribe to a WSRF Topic: **wsn-subscribe**
- Know the EPR of the Index Service you want to register to. This can be seen in the container output at startup of the container on the index host, and will look something like this: `https://myhost:8443/wsrf/services/DefaultIndexService`, (from a default installation of GT version)
- Know the EPR to the resource and the names of the resource properties you want to advertise (basically the same information you would use for the corresponding user command).



Note

If the WSRF service does not advertise the data you want as RPs, you need to use the instructions for [Section 2.3.2, “Configuring a custom information provider based on UsefulRP/RPProvider”](#).

For more information, see the documentation for the [the `mds-servicegroup-add\(1\)` tool](#).

1.1. Registering resources (general)

To register resources with the Index Service:

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

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

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

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

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

The general syntax of the configuration file is:

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

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

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

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

  // One or more service group registration blocks:

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

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

```

    // Aggregator-source-specific configuration parameters
  </Content>
</ServiceGroupRegistrationParameters>

</ServiceGroupRegistrations>

```

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

Table 4.1. Aggregator configuration parameters

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

1.2. Options for the Content block

1.2.1. Request single resource property (GetResourcePropertyPollType)

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

where:

`PollIntervalMillis` is the poll refresh period in milliseconds.

`ResourcePropertyName` is the QName of the resource property you want to poll.

1.2.2. Request one or more resource properties (GetMultipleResourcePropertiesPollType)

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

where:

`PollIntervalMillis` is the poll refresh period in milliseconds.

`ResourcePropertyNames` are the QNames of the resource properties to poll for. There is no limit on the number of `ResourcePropertyNames` that may be specified.

1.2.3. Query resource properties with XPath (QueryResourcePropertiesPollType)

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

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

where:

PollIntervalMillis is the poll refresh period in milliseconds.

QueryExpression is an xsd:any element - the Dialect attribute specifies the dialect of the query expression.

1.2.4. Subscribe to a WSRF Topic (AggregatorSubscriptionType)

The configuration block for SubscriptionAggregatorSource looks like this:

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

where:

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

2. Registering any Grid resource via Information Providers

There are other types of resources in a Grid you may want to register: a non-WSRF resource (anything other than a service that publishes resource properties; for example, a web server or a file system) or a WSRF service that has the data you want but does not publish it as resource properties.

Important

Any time you want to register a resource but cannot directly access resource properties, you need an [information provider](#) to interface between that resource and WS MDS.

As discussed in the [MDS Key Concepts guide](#), information providers are basically the link between what you want monitored and WS MDS. WS MDS comes with standard information providers to monitor clusters, queueing systems, some non-WSRF services, and other useful things. You can also write your own custom information providers for just about any kind of Grid resource you need to monitor.

2.1. Aggregator Framework vs UsefulRP

There are two interfaces available for deploying information providers: [UsefulRP](#) and the [Execution Aggregator Source](#).

The UsefulRP framework can be used by any service that runs in a Globus container to create and advertise a resource property. The service code makes a call to a UsefulRP Java method, which in turn reads a configuration file to determine how to generate the resource property (by executing a command, reading a file, calling a Java method, or some combination of those methods). For a real-life example, the GRAM service uses UsefulRP to create and publish a resource property containing information about the queues that it manages, then registers that resource property to the default Index server running in the same Globus container. End-users can find that information by querying GRAM directly or by querying MDS. Because the Index Service is a WSRF service that runs in the Globus container, it can also publish resource properties using UsefulRP; however, these will be published as separate resource properties rather than added to the same service group as the rest of the index information.

The Execution Aggregator Source is a mechanism for adding information to the resource property / service group used by the Index (or Trigger) service. An Index or Trigger service configured to use the Execution Aggregator Source will execute a program periodically and update its service group with the output of that program.

However, all a sysadmin typically needs to know about aggregator vs. usefulrp is that they require different methods of configuration (outlined in [a later section]). The following section lists available information providers that are based on both aggregator and usefulrp.

2.2. Default Information Providers

The following list includes information providers available with a default installation of GT 4.2.0:

- UsefulRP-based Information Providers: Information providers are available for [Ganglia], [Hawkeye] and [Nagios]. They are based on UsefulRP and the GLUE 1.1 schema. For more information, see [Section 2.4, “Registering Clusters and Condor Pools”](#).

- **Aggregator-based Information Providers:**
 - [Web Check Information Provider] - This provider connects to a web server. If the connection can successful be made the server is registered as alive, otherwise it is down.
 - [Cert Check Information Provider] - This provider will monitor the life of a remote certificate. It connects to an SSL TCP listener and obtains the certificate from it. If then checks the cert for subject, start date, end data, and email address. All of this information is put in the index if it is successfully obtained.
 - [GridFTP Information Provider] - This provider connects to a GridFTP server, reads its banner, and puts the banner in the Index Service. If the connection cannot be made in 30 seconds, or some other error occurs the server is marked as down.
 - [GKrellm Information Provider] - Monitors local resources and provides basic system information for Unix systems.

2.3. Configuring custom information providers

2.3.1. Configuring a custom information provider based on Aggregator

The Aggregator Framework is the underlying foundation of the MDS services (Index, Trigger) and is what registers WSRF services to a DefaultIndexService. In this [method], an *execution aggregator source* information provider gathers arbitrary XML information by executing an external script or program (similar to how we set up the [Trigger Service](#)).

But first, a little bit about aggregator sources, in particular the execution aggregator source. In [Key Concepts](#), we talked about three aggregator sources: query, subscription and execution. We have already covered the query and subscription sources - they are based on WSRF and are used when [registering WSRF services](#):

- The query aggregator source is used when registering via single resource property, multiple resource properties and querying.
- The subscription aggregator source, as you can guess, is used when registering via subscriptions.

In general, you do not need to know much more about those two (unless you want to develop applications using those mechanisms, in which case you can learn more [here](#)).

Unlike the query and subscription sources, the execution aggregator source is not based on WSRF; instead, it gets its information through any executable with an output of well-formed XML.

To configure this type of information provider, in general, you need to:

1. Save the executable to `$GLOBUS_LOCATION/libexec/aggrexec/`. Note: for Aggregator, it is required that you save the executable to this location.
2. Enable the provider in WS MDS by editing the `jndi-config.xml` for the Index Service to add a mapping in `executableMappings`.
3. Configure the registration file using `ExecutionPollType` in the `Content` block:

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

```
<agg:PollIntervalMillis>interval_in_ms</agg:PollIntervalMillis>
<agg:ProbeName>dummy_namespace:probe_name</agg:ProbeName>
</agg:ExecutionPollType>
</agg:AggregatorConfig>
<agg:AggregatorData/>
</Content>
```

4. Register with **mds-servicegroup-add**.

For a detailed tutorial, see [GT 4.2.0 Index Service: How to Write a Simple Execution Aggregator Information Provider for WS MDS](#)

2.3.2. Configuring a custom information provider based on UsefulRP/RPProvider

This method is similar to the above method in that it requires an executable that outputs well-formed XML. However the procedure is a little simpler. In general, you need to:

1. Choose a schema for your data.
2. Save the executable to `$GLOBUS_LOCATION/libexec/` (although it is not strictly necessary; ie, you can save them anywhere you want as long as the path is reflected in the following config file).
3. Edit an RPProvider configuration file to enable the information provider you just wrote. This file must be saved to `$GLOBUS_LOCATION/etc/globus_wsrp_mds_index/`.
4. Restart container

These steps are described in more detail in two documents:

- [How to Write an External Element producer using the RPProvider Framework](#) - detailed tutorial shows you how to register a non-WSRF, dynamic resource via UsefulRP/RPProvider.
- [How to Write a File Element producer using the RPProvider Framework](#) - as above, but for static resources (pulls in static contents from a file).

For even more details, see the documentation for [UsefulRP](#).

Note: If this is a GLUE-based information provider, you do not need to do any configuration. GRAM4 and WS MDS will automatically take care of the registration. [FIXME - make sure i'm saying this right]

2.4. Registering Clusters and Condor Pools

WS MDS is frequently used with cluster monitoring systems, such as Ganglia and Hawkeye (for Condor pools). GT 4.2.0 includes information providers for Hawkeye, Ganglia and Nagios. These information providers are based on the GLUE 1.1 schema and UsefulRP to collect information from two sources: the scheduler and the cluster monitoring system. Both sources are merged to form a single output resource property in the GLUE schema (which is used by [GRAM4](#) and accessible via the Index Service).

In general, you simply edit a configuration file to add the correct configuration block, then restart the container. To make it simpler, you can use the `mds-gluerp-configure` tool to create a configuration file with the correct values.



Note

The scheduler does not need to be configured for default providers - GRAM4 already knows the scheduler bundle to use through other means.

For configuration details, follow these links:

- [Ganglia](#)
- [Hawkeye](#)
- [Nagios](#)

2.4.1. Writing new cluster providers -> dev guide

There are two kinds of providers used by the GLUE Resource Property:

- Scheduler providers - which provide information about the queues that a batch system makes available. Scheduler providers should be written as executables with a name corresponding to the name of the batch system.
- Cluster information providers - which provide information about the host(s) on which GRAM will run jobs. Cluster providers can be written either as Java classes or as executables.

Java providers should implement the [GLUEElementProducer²](#) interface.

Executable providers should output a single XML document to stdout when executed, and then exit.

² http://www-unix.globus.org/api/javadoc-4.0.0/globus_wsrf_mds_usefulrp/org/globus/mds/usefulrp/glue/GLUEElementProducer.html

Chapter 5. Testing

[FIXME describe]

DRAFT

Chapter 6. Security Considerations

[FIXME - which frag to import?]

DRAFT

Chapter 7. Debugging

Because WS MDS is based on the Java WS Core, information on troubleshooting the container is included here:

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/4.2/4.2.0/common/javawscore/sample-container-log.txt>

Chapter 8. Troubleshooting

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

FIXME - should we pull in the error tables from aggregator/index/trigger/webmds?

DRAFT

1. Java WS Core Errors

DRAFT

Table 8.1. 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 <code>WS-Addressing 'To'</code> request header is missing	This warning is logged by the container if the request did not contain the necessary <code>WS-Addressing</code> headers, those headers at all or is somehow misconfigured.
<code>java.io.IOException: Token length X > 33554432</code>	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.
<code>java.lang.NoSuchFieldError: DOCUMENT</code>	This error usually indicates a mismatch between the version of Apache Axis that the code was compiled with and the version currently running with.
<code>org.globus.wsrfl.InvalidResourceKeyException: Argument key is null / Resource key is missing</code>	These errors usually indicate that a resource key was not passed with the request or that an invalid resource key was used (the element <code>QName</code> of the resource key did not match what the service expected).
Unable to connect to <code>localhost:xxx</code>	Cannot resolve <code>localhost</code> . The machine's <code>/etc/hosts</code> isn't set up correctly and/or you do not have DNS for <code>localhost</code> .
<code>org.globus.common.ChainedIOException: Failed to initialize security context</code>	This may indicate that the user's proxy is invalid.
Error: <code>org.xml.sax.SAXException: Unregistered type: class xxx</code>	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 <code>'https'</code> protocol	When a client fails with the following exception: <pre>java.io.IOException: No socket factory for 'https' protocol at org.apache.axis.transport.http.HTTPSender.getSocket(HTTPSender.java:100) org.apache.axis.transport.http.HTTPSender.writeToSocket(HTTPSender.java:110) org.apache.axis.transport.http.HTTPSender.invoke(HTTPSender.java:120)</pre> <p>FIXME - it may have happened because...</p>

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

Index

A

- aggregator sources, 14
 - execution, 14
 - query, 14
 - subscription, 14

B

- building, 1
- building a hierarchy of Index Services, 4

C

- configuration file, registering
 - Content block options, 11
 - AggregatorSubscriptionType, 12
 - ExecutionPollType, 14
 - GetMultipleResourcePropertiesPollType, 11
 - GetResourcePropertyPollType, 11
 - QueryResourcePropertiesPollType, 12
 - example-aggregator-registration.xml, 8
 - parameters, 10
- configuring
 - default, 2
 - overview, 2

D

- debugging
 - logging, 19
- deploying, 7
 - in a Virtual Organization (VO), 7
 - typical, 3
- deployment
 - typical, 3

E

- EPR, 8, 10
- errors, 22

H

- hierarchy of Index Services, 4
- hierarchy.xml, 4

I

- Index Service, 2
- information providers, 13
 - clusters, 15
 - Condor pools, 15
 - configuring, Aggregator, 14
 - configuring, UsefulRP, 15

- default, 13
- GLUE 1.1 schema, 15
- installing, 1

J

- jndi-config.xml, 3, 14

L

- logging
 - CEDPS-compliant, 19
 - debugging, 19

M

- mds-servicegroup-add, 8

R

- registering
 - automatic, of GT WS components, 3
- registering to an Index Service
 - any Grid resource, 13
 - WSRF service, 8
 - query resource properties via XPath, 8
 - request multiple resource properties, 8
 - request single resource properties via XPath, 8
 - request single resource property, 8
 - subscribe to a Topic, 8
- registering with the default Index Service, 8
- registration
 - automatic, of GT WS components, 3

T

- testing, 17
- Trigger Service, 6

U

- upstream, 4
- upstream.xml, 4
- usage
 - simple, 2
- use cases
 - admin, 2
- using
 - basic, 2

W

- WebMDS, 6
- wsrf-query, 2