GT 4.2.1 Index Service: Developer's Guide

GT 4.2.1 Index Service: Developer's Guide

Introduction

The WS MDS Index Service collects information about grid resources and publishes that information as a service group. Client programs use resource property queries or subscription/notification to retrieve information from the index. Information can be added to the index via a number of different mechanisms: since the Index Service is implemented using the *Aggregator Framework*, any *aggregator source* can be used to provide information for the index.

This document describes the programmatic interfaces to the Index 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

List of Figures

List of Tables

Index Service How-tos

A

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

C

configuration interface,
    overview,
configuring,
    defining aggregator sources,
    execution aggregator source,
    overview,

D

debugging,
    logging,
debugging (developer),
dependencies,

E

errors,
execution aggregator information provider,
execution aggregator source,

F

features,

G

globus-index-add,

I

information providers
    writing execution aggregator information provider,

L

logging
    debugging,

M

mds-servicegroup-add,

O

org.globus.mds.aggregator.impl,

P

platforms,

R

refresh existing entries in an index service,
register entries to an index service,
registering resources to MDS aggregator services,
related documents,
resource
    globus-wsrf-query,
    querying resource properties,
resource properties
    getting a single resource property from a resource,
    getting multiple resource properties from a resource,
    getting multiple values,
    getting value,
    globus-wsrf-get-properties,
    globus-wsrf-get-property,
    querying,
    querying the resource property document of a resource,
    wsrf-get-properties,
    wsrf-get-property,
    wsrf-query,
resource properties, AggregatorServiceGroup,

S

security considerations,
services,

T

troubleshooting
    for developers,
tutorial
    build a grid service,

U

usage scenarios,
usage scenarios (programming)
    adding information to an Index Service,
    retrieving information from an Index Service,

W

WSDL,

Chapter 1. Before you begin

1. Feature summary

Features new in release 4.2.1

- The Index Service now supports, in addition to queries made using the default XPath dialect, the new TargetedXPath dialect, which enables users to specify their own namespace mappings in queries.

2. Tested platforms

Tested Platforms for WS-MDS Index Service:

- Linux on i386

- Windows XP

Tested containers for WS-MDS Index Service:

- Java WS Core container

- Tomcat 5.0.28

<xi:include></xi:include>

3. Technology dependencies

The Index Service depends on the following GT components:

- Java WS Core

- Aggregator Framework

4. 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. Retrieving information from an index service

Information is retrieved from the index using the standard Java WS Core API calls for getting resource property information:

- `GetResourceProperty` to request a single resource property by name,

- `GetResourceProperties` to request several resource properties by name,

- `QueryResourceProperty` to perform an XPath query on a resource property document, and

- the notification/subscription mechanism.

See the Chapter 6, APIs for API details.

2. Adding information to an index

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

Chapter 3. GT 4.2.1 Index Service: How to Write a Simple Execution Aggregator Information Provider for WS MDS

1. Introduction

This document is intended to be a starting guide to writing non web-service based *information providers* for the WS MDS. It covers the concepts and walks through a simple example of how to get arbitrary information into the WS MDS using the Chapter 6, Configuring Execution Aggregator Source. This *Aggregator Source* is used for gathering arbitrary XML information about a registered resource by executing an external script. This is mostly useful for scenarios where you would like to publish information into the WS MDS from a *non web-service* based information source. For web-service based information sources that export known Resource Properties, it is much easier to use Chapter 4, Configuration file: parameters for the query aggregator source. However, that source is outside the scope of this document.

This document covers writing a simple information provider that publishes fortune information at a regular interval into the WS MDS's *Index Service*. This example was chosen because it is dynamic and simple, yet it illustrates all the fundamentals of this type of information provider.

2. Choosing (or conforming to) a Schema

The first step to getting information into the WS MDS is to decide which information you would like to have published. Since the data is in XML format, you should choose (or pick) the schema that you'd like the data to conform to. This generally means coming up with element names and types and creating some mapping of the data you're about to retrieve from your non web-service based application before putting it in to the WS MDS. For this example, I'm going to choose this very simple format for the data:

```
<fortuneInformation>
   <fortuneData>
      ... here is the fortune ...
   </fortuneData>
   <fortuneDateAndTime>
      ... date and time of retrieval ...
   </fortuneDateAndTime>
   <fortuneSourceURL>
      ... the URL of where the fortune was retrieved ...
   </fortuneSourceURL>
</fortuneInformation>
```

As you can see, that format is very simple. An example output will look like this:

```
<fortuneInformation>
   <fortuneData>
     186,282 miles per second: It isn't just a good idea, it's the law!
   </fortuneData>
   <fortuneDateAndTime>
```

```
    Thu Jul 14 18:16:01 CDT 2005
  </fortuneDateAndTime>
  <fortuneSourceURL>
    http://anduin.eldar.org/cgi-bin/fortune.pl?text_format=yes
  </fortuneSourceURL>
</fortuneInformation>
```

Once you've chosen how to represent your data in XML format, you can start thinking about how you're going to retrieve and prepare that data for publication.

3. The Code

The second step to getting information into the WS MDS is to write a script (or program) that gathers and formats the appropriate data. This can be C code, shell script, perl code, etc, and it doesn't matter what kind of methods it uses behind the scenes, so long as it produces well formatted XML data.

For example, if we wanted to publish a fortune into the Index Service (using the free and charitable online service located at http://anduin.eldar.org/cgi-bin/fortune.pl), we could write a simple shell script to retrieve it and format it into our chosen XML schema.

You can sample the source code for this example implementation here. It is written as a bash shell script due to its simplicity. Tested platforms include GNU/Linux only. For this script to properly publish information, you must have one (or more) of the following programs installed on the system: *wget*, *lynx*, or *fortune*. All of these programs come standard with most GNU/Linux distributions, and it's important to note that only one of them is required (i.e. not ALL are required). [*NOTE: Windows users must have something like the* cygwin[1] *operating environment for this to work*]

Download the code: fortune_script.sh.

This file should be saved in your *$GLOBUS_LOCATION/libexec/aggrexec* directory, although the reason will be explained in the next section.

4. Enabling The Provider

Now that we have the information provider written, the next step is to enable it so that we can test it. To do this you will need to do three things. First, come up with a short name (i.e. a mapping) that can be used to reference your provider, second, copy your provider to the location where it is expected to be found, and finally, register it to the Index Service with the parameters you'd like.

4.1. Establish mapping of your information provider

To establish the mapping of your provider, you need to edit the `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index/jndi-config.xml` file.

You should see an *executableMappings* section that looks something like this:

```
<parameter>
   <name>executableMappings</name>
      <value>
         aggr-test=aggregator-exec-test.sh,
```

---
[1] http://www.cygwin.com/

```
            pingexec=example-ping-exec
        </value>
</parameter>
```

To add our *fortune_script.sh* file, let's decide that we're call it the *fortuneProvider* as the mapped name. Our entry would then look like this:

```
fortuneProvider=fortune_script.sh
```

With that line added, the entire entry should look like this (note that an extra comma had to be added before our new entry):

```
<parameter>
    <name>executableMappings</name>
        <value>
            aggr-test=aggregator-exec-test.sh,
            pingexec=example-ping-exec,
            fortuneProvider=fortune_script.sh
        </value>
</parameter>
```

☞ **Note**

> The reason we are required to establish this mapping in the first place is for security reasons. The execution aggregator source references this mapping when it's registered, rather than a full path name to a script to avoid allowing arbitrary registrations to be made that can execute arbitrary code. Requiring this mapping be configured before starting the globus container guarantees that the system administrator of the deployment has approved of the use of the new provider.

4.2. Copy information provider to correct location

To make sure your provider is in the expected place, it *MUST* be copied to the *$GLOBUS_LOCATION/libexec/aggrexec* directory. Notice how the full path of the script was not specified in the above example when making the mapping. That's because the path of *$GLOBUS_LOCATION/libexec/aggrexec* is simply assumed and it will be pre-pended at run-time for you. Make sure your file resides in this directory with proper executable permissions.

Check the listing to make sure:

```
neillm@glob ~ $ ls -al $GLOBUS_LOCATION/libexec/aggrexec/
total 12
drwxr-xr-x  2 neillm wheel 4096 Jul 16 14:01 .
drwxr-xr-x  6 neillm wheel 4096 Jul  8 14:52 ..
-rwxr-xr-x  1 neillm wheel  345 Jul  8 14:52 aggregator-exec-test.sh
-rwxr-xr-x  1 neillm wheel 1947 Jul 16 13:52 fortune_script.sh
```

4.3. Configure the registration file

So now that we've completed the first two steps of enabling the provider, we only have left to decide on the final details of how to make the registration to the Index Service.

To do this, you'll need a registration file. There are many types of registrations that can possibly occur, due to the flexibility of the Aggregator Framework. You can view several examples in the $GLOBUS_LOCATION/etc/glo-bus_wsrf_mds_aggregator/example-aggregator-registration.xml file.

For this example, we'll simply use the custom fortune registration file provided[2] , which is specific to the fortune provider we've made that uses the Execution Aggregator source. It's relatively simple, and the fields worth mentioning are shown here:

```
<defaultServiceGroupEPR>
    <wsa:Address>https://127.0.0.1:8443/wsrf/services/DefaultIndexService</wsa:Address>
</defaultServiceGroupEPR>

<defaultRegistrantEPR>
    <wsa:Address>https://127.0.0.1:8443/wsrf/services/fortuneProvider</wsa:Address>
</defaultRegistrantEPR>
```

These fields need to be updated to match how you'll be running your container. You'll need to properly address it, that is. For example, if you're running without security enabled on port 8080 and have an IP address of www.xxx.yyy.zzz, you should substitute the "https://127.0.0.1:8443" base part of the address with "http://www.xxx.yyy.zzz:8080".

Next, view or modify this section of the fortune-provider-registration.xml file:

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

  <!-- Renew this registration every 600 seconds (10 minutes) -->
  <RefreshIntervalSecs>600</RefreshIntervalSecs>
  <Content xsi:type="agg:AggregatorContent"
          xmlns:agg="http://mds.globus.org/aggregator/types">
    <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
      <agg:ExecutionPollType>

        <!-- Run our script every 300,000 milliseconds (5 minutes) -->
        <agg:PollIntervalMillis>300000</agg:PollIntervalMillis>

        <!-- Specify our mapped ProbeName registered in the
             $GLOBUS_LOCATION/etc/globus_wsrf_mds_index/jndi-config.xml
             file -->
        <agg:ProbeName>fortuneProvider</agg:ProbeName>

      </agg:ExecutionPollType>
    </agg:AggregatorConfig>
    <agg:AggregatorData/>
  </Content>
</ServiceGroupRegistrationParameters>
```

The relevant fields here that you can configure are the following:

RefreshIntervalSeconds - the amount of that time that should pass before the registration is renewed for you. 600 seconds (i.e. 10 minutes) is generally sufficient, and certainly is for this example. (Note: the mds-servicegroup-add utility will perform these registrations for you automatically at these time intervals). This parameter's unit is in seconds.

---

[2] fortune-provider-registration.xml

PollIntervalMillis - this is the time interval that we execute the specified provider. It's important to not set this value too low, as there's little value in having it execute extremely frequently given the overhead. For our example, we'll set it to 5 minutes (i.e. 300000 milliseconds). This means, the fortune information published in the Index Service will be updated once every 5 minutes. This parameter's unit is in milliseconds.

ProbeName - here is where the executable mapping is put to use. It must exactly match the (left-hand side) name you specified in the *$GLOBUS_LOCATION/etc/globus_wsrf_mds_index/jndi-config.xml*. For this example, we chose this name to be *fortuneProvider*, and you can see that's what we've specified.

Download the example registration file, fortune-provider-registration.xml.

4.4. Register with Index Service: run mds-servicegroup-add

Finally, to make the registration of our provider to the Index Service, you should run the *mds-servicegroup-add* program in a similar mannner:

```
neillm@glob ~ $ $GLOBUS_LOCATION/bin/mds-servicegroup-add -s \
https://127.0.0.1:8443/wsrf/services/DefaultIndexService \
fortune-provider-registration.xml

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

Note that you will have to specify the proper URI location of your Index Service on the command line and not the one specified above (unless it's the same, of course).

5. An Example Query

```
neillm@glob bin $ ./wsrf-query -s \
https://127.0.0.1:8443/wsrf/services/DefaultIndexService \
"//*[local-name()='fortuneInformation']"

<fortuneInformation xmlns="">
<fortuneData>
They told me you had proven it When they discovered our results About
a month before. Their hair began to curl The proof was valid, more or
less Instead of understanding it But rather less than more. We'd run
the thing through PRL. He sent them word that we would try Don't tell
a soul about all this To pass where they had failed For it must ever
be And after we were done, to them A secret, kept from all the rest
The new proof would be mailed. Between yourself and me. My notion was
to start again Ignoring all they'd done We quickly turned it into code
To see if it would run.
</fortuneData>
<fortuneDateAndTime>
Wed Jul 20 12:36:36 BST 2005
```

```
</fortuneDateAndTime>
<fortuneSourceURL>
http://anduin.eldar.org/cgi-bin/fortune.pl?text_format=yes
</fortuneSourceURL>
</fortuneInformation>
```

This segment of the query output represents the fortune data we've just written and configured for use. As you can see the *fortuneInformation* block was properly published into the Index Service since it's now been properly configured and registered!

6. Contact the author

Contact the author at <u>neillm@mcs.anl.gov</u>[3].

---

[3] mailto:neillm@mcs.anl.gov

Chapter 4. Tutorials

Use of the index service is covered in the <u>Build a Grid Service Tutorial (GlobusWORLD 2005)</u>[1].

---

[1] http://www.globus.org/toolkit/tutorials/BAS/

Chapter 5. Architecture and design overview for the WS MDS Index Service

There are essentially two interfaces to the Index Service -- one for getting information into the index, and one for retrieving information from the index.

Information is retrieved from the Index Service as service group entries using the standard WS MDS Core APIs for resource property queries or subscription/notification.

Because the Index is implemented as a WS MDS Aggregator Framework, the programmatic interface for getting information into the index is to create an aggregator source. The Aggregator Framework's architecture is described in the next section.

Chapter 6. Architecture and design overview for the WS MDS Aggregator Framework

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

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

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

Figure 6.1. Graphic of Information Services Flow

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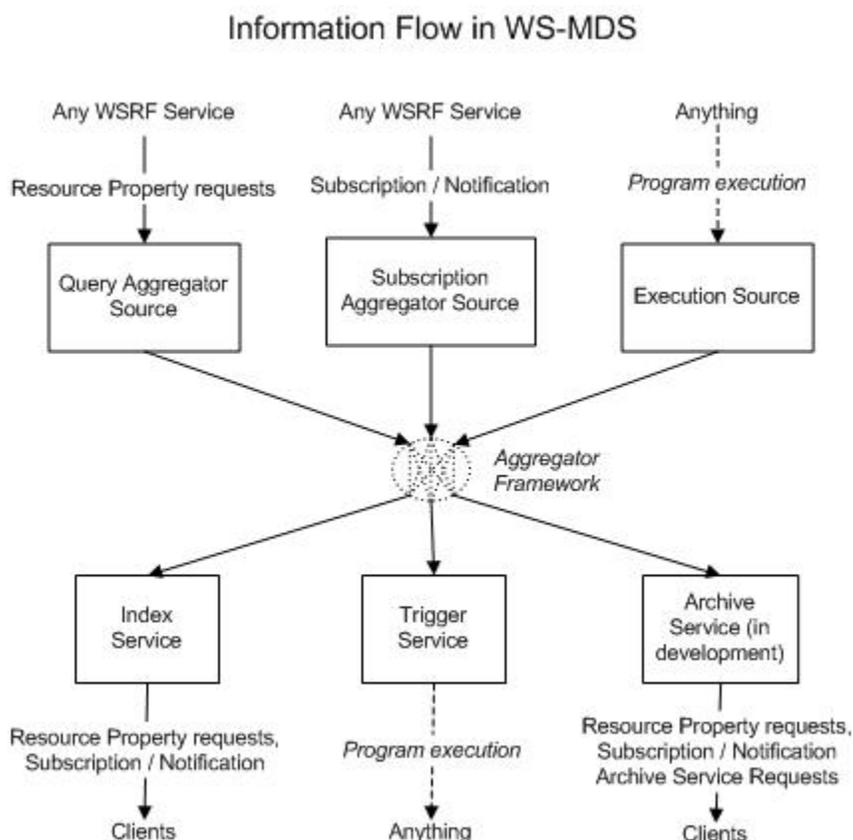
Aggregators work on a type of service group called an **AggregatorServiceGroupRP**. Resources may be *registered* to an AggregatorServiceGroupRP using the service group `add` operation, which will cause an entry to be added to the service group. The entry will include configuration parameters for the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

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

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

1. Standard aggregator sinks

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

Table 6.1. Standard aggregator sinks

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

2. Standard aggregator sources

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

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

☞ **Note**

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

Table 6.2. Standard aggregator sources

Aggregator Source	Description
QueryAggregatorSource	The query source collects information from a registered resource by using WS-Resource Properties polling mechanisms: • `GetResourcePropertyPollType`; requests a single Resource Property from the remote resource. • `GetMultipleResourcePropertiesPollType`; requests multiple Resource Properties from the remote resource. • `QueryResourcePropertiesPollType`; requests a query be executed against the Resource Property Set of the remote resource. Polls are made periodically, with both the period and target Resource Properties specified in the registration message.
SubscriptionAggregator-Source	The subscription source collects information from a registered resource using WS-Notification mechanisms. Data is delivered when property values change, rather than periodically.
ExecutionAggregator-Source	The execution source collects information about (not necessarily from) a registered resource by execution of a local executable, which is passed as input the identity of the registered resource. Details of the interface between the execution source and local executables are in Chapter 6, Configuring Execution Aggregator Source.

Chapter 7. APIs

1. Programming Model Overview

Index Service queries are performed using resource property requests; consult <u>Java WS Core</u> for details.

The contents of an index are maintained using the aggregator framework programming model, and can receive data from any *aggregator source*. 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 the <u>Aggregator Sources Reference</u>; information about how to create new aggregator sources can be found in <u>Developer's Guide</u>.

Chapter 8. WS and WSDL

1. Protocol overview

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

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

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

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

1.   The appropriate aggregation source and sinks will be informed,

2.   the aggregator source will begin collecting data and inserting it into the corresponding service group entry,

3.   and the aggregator sink will begin processing the information in the service group entries.

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

2. Operations

2.1. AggregatorServiceGroup

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

2.2. AggregatorServiceGroupEntry

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

---

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

---

3. WS MDS Aggregator Framework Resource Properties

3.1. AggregatorServiceGroup Resource Properties

- `Entry:` This resource property publishes details of each registered resource, including both an EPR to the resource, the Aggregator Framework configuration information, and data from the sink.

- `RegistrationCount:` This resource property publishes registration load information (the total number of registrations since service startup and decaying averages)

4. Faults

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

5. WSDL and Schema Definition

- AggregatorServiceGroup[3]

- AggregatorServiceGroupEntry[4]

- common types used by AggregatorServiceGroup and AggregatorServiceGroupEntry[5]

Other relevant source files are the:

- WSRF service group schema[6]

- WSRF resource lifetime schema[7]

- MDS Usefulrp schema.

---

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

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

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

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

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

WS MDS Index User Commands

The index service exposes information via service groups and is accessed using the same command-line tools used to query other WSRF services for information. These tools are part of <u>Java WS Core</u> .

* <u>wsrf-query</u>

* <u>wsrf-get-property</u>

* <u>wsrf-get-properties</u>

A set of functionally equivalent tools exist written using WS C core. They tend to be faster alternatives to the above java programs. These tools are part of <u>C WS Core</u> .

* <u>globus-wsrf-query(1)</u>

* <u>globus-wsrf-get-property(1)</u>

* <u>globus-wsrf-get-properties(1)</u>

The following commands are originally documented under their respective component guides, but are reproduced here for convenience.

Name

wsrf-query -- Performs query on a resource property document

```
wsrf-query
```

Tool description

Queries the resource property document of a resource. By default, a simple XPath query is assumed that returns the entire resource property document.

Command syntax

```
wsrf-query [options] [query expression] [dialect]
```

Table 3. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.

-k, --key <name value>	Specifies the resource key. The **name** is the QName of the resource key in the string form: **{namespaceURI}localPart**, while the **value** is the simple value of the key. For complex keys, use the **--eprFile** option. Example:
	
	`-k "{http://www.globus.org}MyKey"`
	`    123`

-f, --descriptor <file>	Specifies a client security descriptor. Overrides all other security settings.
-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-g, --delegation <mode>	Enables delegation. **mode** can be either **'limited'** or **'full'**. Only supported with the GSI Secure Conversation authentication mechanism.
-l, --contextLifetime <value>	Sets the lifetime of the client security context. **value** is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.
-m, --securityMech <type>	Specifies the authentication mechanism. **type** can be **'msg'** for GSI Secure Message, or **'conv'** for GSI Secure Conversation.
-c, --serverCertificate <file>	Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-p, --protection <type>	Specifies the protection level. **type** can be **'sig'** for signature or **'enc'** for encryption.
-x, --proxyFilename <value>	Sets the proxy file to use as client credential.
-z, --authorization <type>	Specifies authorization type. **type** can be **'self'**, **'host'**, **'none'**, or a string specifying the expected identity of the remote party.
-t, --timeout <timeout>	Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.

Examples:

```
 $ wsrf-query -s https://127.0.0.1:8443/wsrf/services/DefaultIndexService \
     "count(//*[local-name()='Entry'])"

 $ wsrf-query -s https://127.0.0.1:8443/wsrf/services/DefaultIndexService \
     "number(//*[local-name()='GLUECE']/glue:ComputingElement/glue:State/@glue:FreeCPUs)=0"

 $ wsrf-query -s http://localhost:8080/wsrf/services/ContainerRegistryService \
     "/*/*/*/*[local-name()='Address']"
```

Name

wsrf-get-property -- Gets values of a single resource property

```
wsrf-get-property
```

Tool description

Gets a single resource property from a resource.

Command syntax

```
wsrf-get-property [options] <property>
```

The **<property>** is a QName of the resource property in the string form: **{namespaceURI}localPart**.

Table 4. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an _XML_ file that contains the _WS-Addressing_ endpoint reference.
-s, --service <url>	Specifies the service URL.

-k, --key <name value>	Specifies the resource key. The **name** is the QName of the resource key in the string form: **{namespaceURI}localPart**, while the **value** is the simple value of the key. For complex keys, use the **--eprFile** option. Example:
	
	`-k "{http://www.globus.org}MyKey"`
	`        123`

-f, --descriptor <file>	Specifies a client security descriptor. Overrides all other security settings.
-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-g, --delegation <mode>	Enables delegation. **mode** can be either **'limited'** or **'full'**. Only supported with the GSI Secure Conversation authentication mechanism.
-l, --contextLifetime <value>	Sets the lifetime of the client security context. **value** is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.
-m, --securityMech <type>	Specifies the authentication mechanism. **type** can be **'msg'** for GSI Secure Message, or **'conv'** for GSI Secure Conversation.
-c, --serverCertificate <file>	Specifies the server's _certificate_ file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-p, --protection <type>	Specifies the protection level. **type** can be **'sig'** for signature or **'enc'** for encryption.
-x, --proxyFilename <value>	Sets the proxy file to use as client credential.
-z, --authorization <type>	Specifies authorization type. **type** can be **'self'**, **'host'**, **'none'**, or a string specifying the expected identity of the remote party.
-t, --timeout <timeout>	Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.

Example:

```
$ wsrf-get-property -s http://localhost:8080/wsrf/services/CounterService \ -k
   "{http://counter.com}CounterKey" 123 \
   "{http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd}Cu
```

Name

wsrf-get-properties -- Gets values of multiple resource properties

```
wsrf-get-properties
```

Tool description

Gets multiple resource properties from a resource.

Command syntax

```
wsrf-get-properties [options] <property1> [<property2>...
    <propertyN>]
```

Each **<propertyN>** is a QName of the resource property in the string form: **{namespaceURI}localPart**.

Table 5. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an _XML_ file that contains the _WS-Addressing_ endpoint reference.
-s, --service <url>	Specifies the service URL.

-k, --key <name value>	Specifies the resource key. The **name** is the QName of the resource key in the string form: **{namespaceURI}localPart**, while the **value** is the simple value of the key. For complex keys, use the **--eprFile** option. Example:
	
	`-k "{http://www.globus.org}MyKey"`
	`        123`

-f, --descriptor <file>	Specifies a client security descriptor. Overrides all other security settings.
-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-g, --delegation <mode>	Enables delegation. **mode** can be either **'limited'** or **'full'**. Only supported with the GSI Secure Conversation authentication mechanism.
-l, --contextLifetime <value>	Sets the lifetime of the client security context. **value** is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.
-m, --securityMech <type>	Specifies the authentication mechanism. **type** can be **'msg'** for GSI Secure Message, or **'conv'** for GSI Secure Conversation.
-c, --serverCertificate <file>	Specifies the server's _certificate_ file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-p, --protection <type>	Specifies the protection level. **type** can be **'sig'** for signature or **'enc'** for encryption.
-x, --proxyFilename <value>	Sets the proxy file to use as client credential.
-z, --authorization <type>	Specifies authorization type. **type** can be **'self'**, **'host'**, **'none'**, or a string specifying the expected identity of the remote party.
-t, --timeout <timeout>	Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.

Example:

```
$ wsrf-get-properties -s http://localhost:8080/wsrf/services/CounterService \ -k
   "{http://counter.com}CounterKey" 123 \
   "{http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd}Cu
   \
   "{http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd}Te
```

Name

globus-wsrf-query -- Query a WSRF resource's Resource Property document

```
globus-wsrf-query [OPTIONS]... SERVICE-SPECIFIER QUERY-EXPRESSION
```

Tool description

Perform an XPATH query on a resource property document.

Command syntax

```
globus-wsrf-query [OPTIONS]... SERVICE-SPECIFIER QUERY-EXPRESSION
```

Table 6. Application-specific options

-n | ----nsMapFile FILENAME.	Use the namespace map entries in *FILENAME* in the XPATH context.
-N | --namespace PRE-FIX=NAMESPACE-URI	Create a namespace mapping in the XPATH context for the *PREFIX* string to resolve to the *NAMESPACE-URI* namespace.
-D | --dialect DIALECT-URI	Set query dialect to *DIALECT-URI*. The value **targeted** will be interpreted as **http://wsrf.globus.org/core/query/targetedXPath** (default: **http://www.w3.org/TR/1999/REC-xpath-19991116**).

Table 7. Common options

**-a	--anonymous**	Use anonymous authentication. Requires either -m 'conv' or transport (https) security.
-d, --debug	Enables debug mode. In debug mode, all SOAP messages will be displayed to stderr and full WSRF Fault messages will be displayed.
**-e	--eprFile FILENAME**	Load service EPR from FILENAME. This EPR is used to contact the WSRF service.
**-h	--help**	Displays help information about the command.
**-k	--key KEYNAME VALUE**	Set resource key in the service EPR to be named KEYNAME with VALUE as its value. This can be combined with **-s** to construct an EPR without having an xml file on hand. The **KEYNAME** is a QName string in the format **{namespaceURI}localPart**. while the **VALUE** is a literal string to place in the element. For example, the option **-k '{http://www.globus.org}MyKey' 128** would be rendered as **<MyKey xmlns="http://www.globus.org">128</MyKey>**
-m, --securityMech TYPE	Set authentication mechanism. TYPE is one of **msg** for WS-SecureMessage or **conv** for WS-SecureConversation.
-p, --protection LEVEL	Set message protection level. LEVEL is one of **sig** for digital signature or **enc** for encryption. The default is 'sig'.
**-s	--service ENDPOINT**	Set ENDPOINT the service URL to use. Will be composed with the **-k** parameter if present to add ReferenceProperties to the ENDPOINT
**-t	--timeout SECONDS**	Set client timeout to SECONDS.
**-u	--usage**	Print short usage message.
**-V	--version**	Show version information and exit.
**-v	--certKeyFiles CERTIFICATE-FILENAME KEY-FILENAME**	Use credentials located in **CERTIFICATE-FILENAME** and **KEY-FILENAME**. The key file must be unencrypted.
**-x	--proxyFilename FILENAME**	Use proxy credentials located in **FILENAME**.
**-z	--authorization TYPE**	Set authorization mode. **TYPE** can be **self**, **host**, **none**, or a string specifying the identity of the remote party. The default is **self**.
--versions	Show version information for all loaded modules and exit.

SERVICE-SPECIFIER: [-s URI [-k KEY VALUE] | -e FILENAME]

QUERY-EXPRESSION: XPath-Expression-String

Examples:

```
% globus-wsrf-query -e widget.epr "//*[local-name() = 'CurrentTime']"
<ns02:CurrentTime
    xmlns:ns00="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ns01="http://www.w3.org/2001/XMLSchema"
    xmlns:ns02="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft
    ns00:type="ns01:dateTime">2006-05-30T13:53:15Z</ns02:CurrentTime>
```

```
% globus-wsrf-query -e widget.epr "//*[local-name() = 'CurrentTime']/text()"
2006-05-30T13:53:35Z


% globus-wsrf-query -e widget.epr \
        -N wsrl=http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-
        "//wsrl:CurrentTime/text()"
2006-05-30T13:54:36Z
```

Contents of *widget.epr*:

```
<ns01:EndpointReference xmlns:ns01="http://schemas.xmlsoap.org/ws/2004/03/addressing">
  <ns01:Address>http://globus.my.org:8080/wsrf/services/WidgetService</ns01:Address>
  <ns01:ReferenceProperties>
    <ResourceID xmlns:ns02="http://www.w3.org/2001/XMLSchema-instance" xmlns:ns03="http://
  </ns01:ReferenceProperties>
</ns01:EndpointReference>
```

Limitations

- The namespace mapping option and use of namespace prefixes in the *XPath-Expression-String* does not work when communicating with the Java container unless the *http://wsrf.globus.org/core/query/targetedXPath* dialect is used.

Output and Exit Code

If the query is successful, the program displays the output of the query to *stdout* and terminates with exit code 0. In the case of an error, the type of error will be displayed to *stderr* and the program will terminate with a non-0 exit code.

Name

globus-wsrf-get-property -- Get a resource property's value

```
globus-wsrf-get-property [OPTIONS]... SERVICE-SPECIFIER PROPERTY-NAME
```

Tool description

Get the value of a resource property from a WSRF resource.

Command syntax

```
globus-wsrf-get-property [OPTIONS]... SERVICE-SPECIFIER PROPERTY-NAME
```

Table 8. Common options

-a | --anonymous	Use anonymous authentication. Requires either -m 'conv' or transport (https) security.
-d, --debug	Enables debug mode. In debug mode, all SOAP messages will be displayed to stderr and full WSRF Fault messages will be displayed.
-e | --eprFile FILENAME	Load service EPR from FILENAME. This EPR is used to contact the WSRF service.
-h | --help	Displays help information about the command.
-k | --key KEYNAME VALUE	Set resource key in the service EPR to be named KEYNAME with VALUE as its value. This can be combined with **-s** to construct an EPR without having an xml file on hand. The **KEYNAME** is a QName string in the format **{namespaceURI}localPart**. while the **VALUE** is a literal string to place in the element. For example, the option **-k '{http://www.globus.org}MyKey' 128** would be rendered as **<MyKey xmlns="http://www.globus.org">128</MyKey>**
-m, --securityMech TYPE	Set authentication mechanism. TYPE is one of **msg** for WS-SecureMessage or **conv** for WS-SecureConversation.
-p, --protection LEVEL	Set message protection level. LEVEL is one of **sig** for digital signature or **enc** for encryption. The default is 'sig'.
-s | --service ENDPOINT	Set ENDPOINT the service URL to use. Will be composed with the **-k** parameter if present to add ReferenceProperties to the ENDPOINT
-t | --timeout SECONDS	Set client timeout to SECONDS.
-u | --usage	Print short usage message.
-V | --version	Show version information and exit.
-v | --certKeyFiles CERTIFIC-ATE-FILENAME KEY-FILE-NAME	Use credentials located in **CERTIFICATE-FILENAME** and **KEY-FILENAME**. The key file must be unencrypted.
-x | --proxyFilename FILENAME	Use proxy credentials located in **FILENAME**.
-z | --authorization TYPE	Set authorization mode. **TYPE** can be **self**, **host**, **none**, or a string specifying the identity of the remote party. The default is **self**.
--versions	Show version information for all loaded modules and exit.

SERVICE-SPECIFIER: [-s URI [-k KEY VALUE] | -e FILENAME]

PROPERTY-NAME: [{Namespace-URI}]Property-Name

Example:

```
% globus-wsrf-get-property -e widget.epr \
       '{http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd
```

```
<ns02:CurrentTime
    xmlns:ns00="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ns01="http://www.w3.org/2001/XMLSchema"
    xmlns:ns02="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft
    ns00:type="ns01:dateTime">2006-05-30T14:26:35Z</ns02:CurrentTime>
```

Output and Exit Code

If the property exists, its values (if any) are displayed to *stdout* and the program terminates with exit code 0. In the case of an error, the type of error will be displayed to *stderr* and the program will terminate with a non-0 exit code.

Name

globus-wsrf-get-properties -- Get multiple resource property value

```
globus-wsrf-get-properties [OPTIONS]... SERVICE-SPECIFIER PROPERTY-NAME...
```

Tool description

Get the value of multiple resource properties from a WSRF resource.

Command syntax

```
globus-wsrf-get-properties [OPTIONS]... SERVICE-SPECIFIER PROPERTY-NAME...
```

Table 9. Common options

-a | --anonymous	Use anonymous authentication. Requires either -m 'conv' or transport (https) security.
-d, --debug	Enables debug mode. In debug mode, all SOAP messages will be displayed to stderr and full WSRF Fault messages will be displayed.
-e | --eprFile FILENAME	Load service EPR from FILENAME. This EPR is used to contact the WSRF service.
-h | --help	Displays help information about the command.
-k | --key KEYNAME VALUE	Set resource key in the service EPR to be named KEYNAME with VALUE as its value. This can be combined with **-s** to construct an EPR without having an xml file on hand. The **KEYNAME** is a QName string in the format **{namespaceURI}localPart**. while the **VALUE** is a literal string to place in the element. For example, the option **-k '{http://www.globus.org}MyKey' 128** would be rendered as **<MyKey xmlns="http://www.globus.org">128</MyKey>**
-m, --securityMech TYPE	Set authentication mechanism. TYPE is one of **msg** for WS-SecureMessage or **conv** for WS-SecureConversation.
-p, --protection LEVEL	Set message protection level. LEVEL is one of **sig** for digital signature or **enc** for encryption. The default is 'sig'.
-s | --service ENDPOINT	Set ENDPOINT the service URL to use. Will be composed with the **-k** parameter if present to add ReferenceProperties to the ENDPOINT
-t | --timeout SECONDS	Set client timeout to SECONDS.
-u | --usage	Print short usage message.
-V | --version	Show version information and exit.
-v | --certKeyFiles CERTIFICATE-FILENAME KEY-FILE-NAME	Use credentials located in **CERTIFICATE-FILENAME** and **KEY-FILENAME**. The key file must be unencrypted.
-x | --proxyFilename FILENAME	Use proxy credentials located in **FILENAME**.
-z | --authorization TYPE	Set authorization mode. **TYPE** can be **self**, **host**, **none**, or a string specifying the identity of the remote party. The default is **self**.
--versions	Show version information for all loaded modules and exit.

SERVICE-SPECIFIER: [-s URI [-k KEY VALUE] | -e FILENAME]

PROPERTY-NAME: [{Namespace-URI}]Property-Name

Example:

```
% globus-wsrf-get-properties \
        -s http://grid.example.org:8080/wsrf/services/WidgetService \
        -k "{http://www.globus.org/namespaces/2004/06/core}WidgetKey" 123 \
        "{http://widgets.com}foo" \
        "{http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xs
<ns02:foo
    xmlns:ns00="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ns01="http://www.w3.org/2001/XMLSchema"
    xmlns:ns02="http://widgets.com"
    ns00:type="ns01:string">
Foo Value String
</ns02:foo><ns03:CurrentTime
    xmlns:ns00="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ns01="http://www.w3.org/2001/XMLSchema"
    xmlns:ns03="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft
    ns00:type="ns01:dateTime">2006-05-30T16:04:15Z</ns03:CurrentTime>
```

Output and Exit Code

If the properties exist, their values (if any) are displayed to *stdout* and the program terminates with exit code 0. In the case of an error, the type of error will be displayed to *stderr* and the program will terminate with a non-0 exit code.

WS MDS Index Admin Commands

The mds-servicegroup-add(1) command in the Aggregator Framework is used to configure the Index Service to query information from various sources.

The globus-index-add(1) command line tool is written using WS C and offers similar functionality to mds-servicegroup-add(1) with a few new options.

Name

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

```
mds-servicegroup-add
```

Tool description

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

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

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

Registrations are defined in an XML configuration file, which is documented here: Chapter 3, Registering Aggregator Sources.

For an example using an Index Service, see Simple usage for the Index Service.

And remember to note the section on Limitations.

Command syntax

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

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

where:

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

	self Fail if the server's identity is different from the user's identity.
	host Fail if the server does not have a valid server certificate.
	none Continue regardless of the server's identity.
config.xml	Path to the registration configuration file (see Chapter 3, Registering Aggregator Sources).
	The Globus Toolkit distribution includes an example configuration file: `$GLOBUS_LOC-ATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml`.

The common java client options are also supported.

Registering a resource manually

Prerequisites

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

☞ **Note**

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

- Have a working Index Service, as documented in the Index Service System Administrator's Guide.

- Know the EPR to the resource.

- Know the EPR to the Index Service. This can be seen in the container output at startup of the container on the index host, and will look something like this: `https://myhost:8443/wsrf/services/DefaultIndexService`

Simple usage for the Index Service

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

1. Edit the example configuration file (`$GLOBUS_LOCATION/etc/globus_wsrf_mds_aggregator/example-aggregator-registration.xml`), replacing the EPRs in that file with the EPRs for your resources

2. Run **mds-servicegroup-add** to perform the registrations specified in that file.

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

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

Limitations

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

Name

globus-index-add -- Registering grid resources to MDS index services

```
globus-index-add
```

Tool description

globus-index-add Allows a user to register entries to an Index service and to refresh existing entries. The tool can be run in daemon mode where it is much like mds-servicegroup-add(1). In daemon mode it runs until canceled, periodically refreshing the entry. Unlike mds-servicegroup-add(1) the add and refresh step can be seperated. When adding the user can save the EPR of the entry they just added. Then at a later time they can use that EPR to update the entry. This feature makes it possible to script around updates and adds. Further it allows for entry information to be permisted in the event that the add client dies.

Registrations are defined in an XML configuration file, which is documented here: Section 1, "Registering resources (general)".

For an example using an Index Service, see Simple usage for the Index Service.

Command syntax

The basic syntax for **globus-index-add** is:

```
globus-index-add [options]
```

where:

-h	Print a usage message.
-q	Write no output messages
-d	Run in daemon mode, refreshing the entry every updates cycle.
-u	Update the entry given in the epr.
-vb	Verbose output
-e <endpoint string>	The endpoint string. Ex: http(s)://host:port/service-group-address. This is used with -a
-a <file>	Add to the index service the entry describe in this ServiceGroupRegistrations file.
-E <file>	<file> contains the EPR of the entry to update with -u
-t <minutes>	Set the number of minutes for the entry to live. Used for both -a and -u.
-me <integer>	Set the maximum amount of retries to update before failing. The program will continue to retry to attempt the refresh <integer> number of times
-z [self] | [host] | [id <subject>]	Set the authz method.
-ms [sig] | [conv] | [trans]	Security mechanism. 'msg' for secure message, 'conv' for secure conversation, 'trans' for transport. The default is trans.

Chapter 9. Graphical User Interface

There is no GUI specifically for the Index Service. The release contains <u>WebMDS"</u> which can be used to display monitoring information collected in an Index Service in a normal web browser.

Chapter 10. Configuring an executable to retrieve information

1. Interface introduction

The ExecutionAggregatorSource, which may be used by the Index Service, has a domain-specific interface (specifically, the inputs provided to and outputs expected from the executable program).

2. Syntax of the interface

The syntax of the execution source's domain-specific interface is described in <u>Chapter 6, Configuring Execution Aggregator Source</u>.

Chapter 11. Configuring the WS MDS Index Service

☞ **Note**

> The aggregation source used to collect data can be changed from default, as detailed in the <u>Defining the Aggregator Sources</u> section below.

1. Configuration overview

For a basic installation, the Index Service itself does not need any configuration changes from default; a default Index Service is available and automatically "registers" with the following GT web services based resources to allow monitoring and discovery: [CAS], [RFT], and [GRAM4] (click the links for information about what data is sent and how to change it).

☞ **Note**

> Auto-registration is turned on by default in GT 4.2.1. See the per service links above for information about configuring this capability.

In order for information to appear in the Index Service, the source of that information must be registered to the Index Service. Information sources are registered using tools like <u>mds-servicegroup-add(1)</u>. 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 index, on the same host as a member resource, or on any other host(s).

The Index Service is built on <u>Aggregator Framework</u> and can use any <u>Aggregator Sources Reference</u> to collect information. In the most common case, the index service uses the `QueryAggregatorSource` to gather resource property values from the registered resource using one of the three WS-Resource Properties operations to poll for information; the polling method used depends on the configuration element supplied in the registration content.

Two other aggregator sources are supplied with the distribution: the `SubscriptionAggregatorSource`, which gathers resource property values through subscription/notification, and the `ExecutionAggregatorSource`, which executes an external program to gather information.

2. Defining the Aggregator Sources

The aggregation sources used to collect data can be changed from default by editing the aggregatorSources parameter in the JNDI service configuration. See `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index/jndi-config.xml`:

```
<resource name="configuration"
            type="org.globus.mds.aggregator.impl.AggregatorConfiguration">
  <resourceParams>
    <parameter>
      <name> factory</name>
      <value>org.globus.wsrf.jndi.BeanFactory</value>
    </parameter>
    <parameter>
```

```
      <name>aggregatorSource</name>
      <value>org.globus.mds.aggregator.impl.QueryAggregatorSource
              org.globus.mds.aggregator.impl.SubscriptionAggregatorSource
              org.globus.mds.aggregator.impl.ExecutionAggregatorSource
      </value>
   </parameter>
</resourceParams>
```

This parameter specifies one or more Java classes that may be used to collect data for the Index. By default it is set to use the QueryAggregatorSource, SubscriptionAggregatorSource, and ExecutionAggregatorSource. Details of these standard sources are in the Aggregator Sources Reference.

Chapter 12. 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 for the Index Service

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

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

to the appropriate properties file. Since the Index 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)
[4] http://tomcat.apache.org/tomcat-5.5-doc/logging.html

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

Chapter 13. Troubleshooting

For a list of common errors in GT, see <u>Error Codes</u>.

1. Java WS Core Errors

Table 13.1. Java WS Core Errors

Error Code	Definition
`Failed to acquire notification consumer home instance from registry`	Caused by `javax.naming.NameNotFoundException: Name services 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 *WS-Addressing* 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 H specifies `8443` as a port number and `http` 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 wit currently running with.
`org.globus.wsrf.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 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 `/etc/hosts` isn't set up correctly and/or you do not have DNS fo
`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 regist registered upon deployment without intervention by the user, sometimes they do not.

`No socket factory for 'https' protocol`	When a client fails with the following exception:
	
	`java.io.IOException: No socket factory for 'https' protocol at`
	`        org.apache.axis.transport.http.HTTPSender.getSocket(HTTPSender.j`
	`        org.apache.axis.transport.http.HTTPSender.writeToSocket(HTTPSend`
	`        org.apache.axis.transport.http.HTTPSender.invoke(HTTPSender.java`
	
	FIXME - it may have happened because...

Error Code	Definition
`No client transport named 'https' found`	When a client fails with the following exception: ```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)``` The client is most likely loading an incorrect `client-config.wsdd` configuration file.

`ConcurrentModificationException` in Tomcat 5.0.x	If the following exception is visible in the Tomcat logs at startup, it might cause the `HTTPSValve` to fail:
	
	```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.```
	
	The `HTTPSValve` might fail with the following exception:
	
	```java.lang.NullPointerException at org.apache.coyote.tomcat5.CoyoteRequest.setAttribute(CoyoteReque org.apache.coyote.tomcat5.CoyoteRequestFacade.setAttribute(Coyot org.globus.tomcat.coyote.valves.HTTPSValve.expose(HTTPSValve.jav```
	
	These exceptions will prevent the transport security from working properly in Tomcat.

`java.net.SocketException: Invalid argument or cannot assign requested address`	FIXME - what causes this?
GAR deploy/undeploy fails with `container is running` error	A GAR file can only be deployed or undeployed locally while the container is off. However, GAR deployme fail with this error even if the container is off. This usually happens if the container has crashed or was stopp from cleaning up its state files.

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.org[2] or jwscore-user@globus.org[3] (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 14. Related Documentation

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

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.

C

certificate

A public key plus information about the certificate owner bound together by the digital signature of a CA. In the case of a CA certificate, the certificate is self signed, i.e. it was signed using its own private key.

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.

T

Trigger Service

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

W

Web Services Addressing (WSA)

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

---

[14] http://www.w3.org/2002/ws/addr/

X

XML   Extensible Markup Language (XML) is standard, flexible, and extensible data format used for web services. See the <u>W3C XML site</u>[20] for details.

---

[20] http://www.w3.org/XML/

Index