

GT 4.2.1 WS MDS UsefulRP : Developer's Guide

GT 4.2.1 WS MDS UsefulRP : Developer's Guide

Introduction

This guide contains information of interest to developers working with the WS MDS UsefulRP. It provides reference information for application developers, including APIs, architecture, procedures for using the APIs and code samples.

Table of Contents

1. Before you begin	1
1. Feature summary	1
2. Tested platforms	1
3. Backward compatibility summary	1
4. Technology dependencies	1
5. UsefulRP Security Considerations	1
2. Usage scenarios	2
3. Writing an External Element Information Provider	3
1. Introduction	3
2. Choosing (or conforming to) a Schema	3
3. The Code	4
4. Enabling The Provider	4
5. An Example Query	6
6. Contact the author	7
4. Writing a File Element Information Provider	8
1. Introduction	8
2. Choosing (or conforming to) a Schema	8
3. The File	9
4. Enabling The Provider	9
5. An Example Query	11
6. Contact the author	11
5. Architecture and design overview	12
1. Locating the code	12
6. Producers	13
1. The Current RPProviders	13
2. External Process Element Producer	13
3. File Element Producer	14
7. Transforms	16
8. APIs	17
1. Programming Model Overview	17
2. Component API	17
I. UsefulRP Admin Command	18
mds-gluerp-configure	19
9. Domain-specific interface for UsefulRP/RPProvider	20
10. Configuring UsefulRP	21
1. Configuration overview	21
11. Environment variable interface	23
12. Debugging	24
1. Logging in Java WS Core	24
2. Enabling verbose logging	25
13. Related Documentation	26
Glossary	27

Chapter 1. Before you begin

1. Feature summary

Features in GT 4.2.1 WS MDS UsefulRP

- Provides API and implementation for working with customized resource properties and information providers
- Provides an implementation of a GLUE 1.1 schema compliant resource property
- Provides sample resource property implementations for service developers to incorporate into their own services

2. Tested platforms

Linux, Windows (except GLUERP)

3. Backward compatibility summary

Protocol changes since GT 4.0.x

- N/A, as usefulrp is an API, not a service

API changes since GT 4.0.x

- None

Exception changes since GT 4.0.x

- None

Schema changes since GT 4.0.x

- None

4. Technology dependencies

WS MDS UsefulRP depends on the following GT components:

- Java WSRF Core

WS MDS UsefulRP depends on the following 3rd party software:

- none

5. UsefulRP Security Considerations

Security recommendations for this component include:

- FIXME list

Chapter 2. Usage scenarios

[FIXME this is the section where you can describe common scenarios for developers while working with this component
- can also link to standalone howto docs]

Chapter 3. GT 4.2.1 Index Service: How to Write an External Element producer using the RPPProvider Framework in 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 using the RPPProvider Framework. It covers the concepts and walks through a simple example of how to get arbitrary information into the WS MDS using the External Element producer. This External Element producer is part of the RPPProvider Framework and is used for gathering arbitrary XML information about a 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 WSRF compliant 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.

It should be noted that this document is very similar to the document describing how to write an execution aggregator provider of the same fortune example. However, there are many differences using the newer RPPProvider framework, and this document covers them.

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](http://www.cygwin.com/)¹ operating environment for this to work*]

Download the code: [fortune_script.sh](#).

This file should be saved in your `$GLOBUS_LOCATION/libexec` directory, although if you feel more comfortable placing it somewhere else on your file system, it's ok to do so.

Download the RPPProvider configuration file for customization: [rp-provider-config.xml](#).

This file should be saved in your `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index` directory.

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 a few things. First, enable the RPPProvider framework (if it's not already). Second, enable the Fortune Provider that we've just written, and finally restart your container to view the new information.

¹ <http://www.cygwin.com/>

4.1. Enable the RPPProvider framework

It's possible that your Globus Toolkit installation already has the RPPProvider framework enabled. This section shows you how to verify if it is, and what to do if it is not. To do this, you will need to edit the `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index/server-config.wsdd` file.

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

```
<service name="DefaultIndexService" provider="Handler"
  use="literal" style="document">
  <parameter name="providers"
    value="org.globus.wsrf.impl.servicegroup.ServiceGroupRegistrationProvider
          org.globus.mds.usefulrp.rpprovider.ResourcePropertyProviderCollection
          GetRPPProvider
          GetMRPPProvider
          QueryRPPProvider
          DestroyProvider
          SetTerminationTimeProvider
          SubscribeProvider
          GetCurrentMessageProvider" />

  <parameter name="handlerClass"
    value="org.globus.axis.providers.RPCProvider" />
  <parameter name="scope" value="Application" />
  <parameter name="allowedMethods" value="*" />
  <parameter name="rpProviderConfigFile"
    value="/etc/globus_wsrf_mds_index/rp-provider-config.xml" />
  <parameter name="className"
    value="org.globus.mds.index.impl.DefaultIndexService" />
  <wsdlFile>share/schema/mds/index/index_service.wsdl</wsdlFile>
</service>
```

If the *DefaultIndexService Handler* section doesn't look like this, cut and paste the above and replace the existing *Handler* section for the *DefaultIndexService*. The two key distinctions are in the *providers* value section that has the line `org.globus.mds.usefulrp.rpprovider.ResourcePropertyProviderCollection` and the *rpProviderConfigFile* value. This latter value can be anything you want (relative to the system's `$GLOBUS_LOCATION`), but this document assumes the value that's specified above.

4.2. Enable the Fortune Provider

This step involves editing the `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index/rp-provider-config.xml` file that was specified in the above step. It can be a different file, but you will have to adjust the value in the above step to change it here. For now, we're assuming that it's left as the default. If this file doesn't already exist, feel free to copy the sample file located at `$GLOBUS_LOCATION/etc/globus_wsrf_mds_usefulrp/gluece-rpprovider-sample-config.xml` to `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index/rp-provider-config.xml`, or download the sample tailored for this HOWTO from [here](#)² and copy it to `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index/rp-provider-config.xml`, and start editing from there.

² `rp-provider-config.xml`

To enable the fortune provider, you will need to add a config block that looks like this (or see the `rp-provider-config.xml` file for downloading above):

```
<ns1:resourcePropertyProviderConfiguration xsi:type="ns1:resourcePropertyProviderConfig">
  <ns1:resourcePropertyName xsi:type="xsd:QName"
    xmlns:mds="http://my.ns.domain/fortuneProvider"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">mds:FortuneRP</ns1:resourcePropertyName>
    <ns1:resourcePropertyImpl xsi:type="xsd:string"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">org.globus.mds.usefulrp.rprovider.SingleV
  <ns1:resourcePropertyElementProducers
    xsi:type="ns1:resourcePropertyElementProducerConfig">
    <ns1:className xsi:type="xsd:string"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">org.globus.mds.usefulrp.rprovider.producers
    <ns1:arguments xsi:type="xsd:string"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">/MY/GLOBUS/LOCATION/libexec/fortune_script.s
    <ns1:arguments xsi:type="xsd:string"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">dummyArgument</ns1:arguments>
    <ns1:period xsi:type="xsd:int"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">300</ns1:period>
  </ns1:resourcePropertyElementProducers>
</ns1:resourcePropertyProviderConfiguration>
```

As you can see, there are a number of configurable parameters there such as the location of the fortune provider script, as well as the period at which it will be executed (in seconds). Feel free to edit those as necessary. NOTE: The argument line containing the `dummyArgument` is optional and is only provided as an example of how to pass command line arguments to your script. The fortune script will ignore that argument. If you need to add multiple arguments for your own custom scripts, just add another line similar to that for each of them and they will be received by the script in the same order that they appear in this file.

At this point, the fortune provider has been enabled for use with the RPPProvider framework! As you can see, this framework simplifies much of the work of details of getting information into the `IndexService`. Simply restart the container and make sure there are no errors. If errors are reported, please check over all of the above steps and make sure that they were completed successfully. It's very easy to make unexpected typos!

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>
You will remember, Watson, how the dreadful business of the Abernetty
family was first brought to my notice by the depth which the parsley
had sunk into the butter upon a hot day. -- Sherlock Holmes
</fortuneData>
<fortuneDateAndTime>
Tue Oct 27 12:07:16 PST 2006
</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!

6. Contact the author

Contact the author at neillm@mcs.anl.gov³.

³ <mailto:neillm@mcs.anl.gov>

Chapter 4. GT 4.2.1 Index Service: How to Write a File Element producer using the RPPProvider Framework in WS MDS

1. Introduction

This document is intended to be a starting guide to writing one of the simplest kinds of *information providers* for the WS MDS using the RPPProvider Framework. It covers the concepts and walks through a simple example of how to get arbitrary information into the WS MDS using the File Element producer. This File Element producer is part of the RPPProvider Framework and is used for storing arbitrary XML information by pulling in the static contents from a file. This is mostly useful for scenarios where you would like to publish static information into the WS MDS.

This document covers writing a simple information provider that takes the contents of a Message Of The Day (MOTD) file and stores it in the WS MDS's *Index Service*. While the XML data stored in the file is static, the WS MDS will check it periodically and pull in any changes so that it works in some sense, like a logging system as well. 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 the MOTD file before putting it in to the WS MDS. For this example, I'm going to choose this very simple format for the data:

```
<mysns:MOTD xmlns:mysns="http://myorg.ns.for.testing">

  <mysns:MessageEntry mysns:Status="STATUS">
    <mysns:DateEntered>START DATE</mysns:DateEntered>
    <mysns:DateValidUntil>END DATE</mysns:DateValidUntil>
    <mysns:Content> ... content here ... </mysns:Content>
  </mysns:MessageEntry>

  <mysns:MessageEntry mysns:Status="STATUS">
    ... another message can be here ...
  </mysns:MessageEntry>

</mysns:MOTD>
```

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

```
<mysns:MOTD>
  <mysns:MessageEntry mysns:Status="Urgent">
    <mysns:DateEntered>Wed Nov 15 10:38:25 CST 2006</mysns:DateEntered>
    <mysns:DateValidUntil>Wed Nov 15 11:38:25 CST 2006</mysns:DateValidUntil>
    <mysns:Content>
```

```
URGENT:  It has come to our attention that a disk in the BBX
machine is failing and we will be replacing it within the hour.
Sorry for the inconvenience.
-- Your friendly Systems Staff (systems@foo.org)
</myns:Content>
</myns:MessageEntry>
</myns:MOTD>
```

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. For this kind of information provider, the most common scenarios of getting the formatted XML data into a file published to the Index Service is either by editing it by hand, or by having it auto-generated by some kind of script. For this example, we will assume hand edits are done by a System's person with adequate permissions to the MOTD file.

3. The File

The second step to getting information into the WS MDS is to write a file that contains the appropriate data. For this example, we've provided a sample file that you can download and test. After you've verified that it's all working, feel free to modify the file as you see fit.

Download the sample file: [motd.xml](#).

This file could be saved in your `$GLOBUS_LOCATION/etc` directory, although if you feel more comfortable placing it somewhere else on your file system, it's ok to do so. For this example, we will assume it's located at the suggested location.

4. Enabling The Provider

Now that we have the data for publication, the next step is to enable it so that we can test it. To do this you will need to do a few things. First, enable the RPPProvider framework (if it's not already). Second, enable the MOTD Provider that we've just prepared, and finally restart your container to view the new information.

4.1. Enable the RPPProvider framework

It's possible that your Globus Toolkit installation already has the RPPProvider framework enabled. This section shows you how to verify if it is, and what to do if it is not. To do this, you will need to edit the `$GLOBUS_LOCATION/etc/globus_wsrf_mds_index/server-config.wsdd` file.

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

```
<service name="DefaultIndexService" provider="Handler"
  use="literal" style="document">
  <parameter name="providers"
    value="org.globus.wsrf.impl.servicegroup.ServiceGroupRegistrationProvider
          org.globus.mds.usefulrp.rpprovider.ResourcePropertyProviderCollecti
          GetRPPProvider
          GetMRPPProvider
          QueryRPPProvider
          DestroyProvider
          SetTerminationTimeProvider
          SubscribeProvider
```

```
GetCurrentMessageProvider" />

<parameter name="handlerClass"
  value="org.globus.axis.providers.RPCProvider" />
<parameter name="scope" value="Application" />
<parameter name="allowedMethods" value="*" />
<parameter name="rpProviderConfigFile"
  value="/etc/globus_wsrf_mds_usefulrp/rp-provider-config.xml" />
<parameter name="className"
  value="org.globus.mds.index.impl.DefaultIndexService" />
<wsdlFile>share/schema/mds/index/index_service.wsdl</wsdlFile>
</service>
```

If the `DefaultIndexService Handler` section doesn't look like this, cut and paste the above and replace the existing `Handler` section for the `DefaultIndexService`. The two key distinctions are in the `providers` value section that has the line `org.globus.mds.usefulrp.rpprovider.ResourcePropertyProviderCollection` and the `rpProviderConfigFile` value. This latter value can be anything you want (relative to the system's `$GLOBUS_LOCATION`), but this document assumes the value that's specified above.

4.2. Enable the MOTD Provider

This step involves editing the `$GLOBUS_LOCATION/etc/globus_wsrf_mds_usefulrp/rp-provider-config.xml` file that was specified in the above step. It can be a different file, but you will have to adjust the value in the above step to change it here. For now, we're assuming that it's left as the default. If this file doesn't already exist, feel free to copy the sample file located at `$GLOBUS_LOCATION/etc/globus_wsrf_mds_usefulrp/gluece-rpprovider-sample-config.xml` to `$GLOBUS_LOCATION/etc/globus_wsrf_mds_usefulrp/rp-provider-config.xml` and start editing from there.

To enable the MOTD provider, you will need to add a config block that looks like this:

```
<ns1:resourcePropertyProviderConfiguration xsi:type="ns1:resourcePropertyProviderConfig">
  <ns1:resourcePropertyName xsi:type="xsd:QName" xmlns:myns="http://myorg.ns.for.testing">myNS:rpProvider
  <ns1:resourcePropertyImpl xsi:type="xsd:string">org.globus.mds.usefulrp.rpprovider.Single
  <ns1:resourcePropertyElementProducers xsi:type="ns1:resourcePropertyElementProducerConfig">
    <ns1:className xsi:type="xsd:string">org.globus.mds.usefulrp.rpprovider.producers.FileE
    <ns1:arguments xsi:type="xsd:string">/PATH/TO/GLOBUS_LOCATION/etc/motd.xml</ns1:argumen
    <ns1:period xsi:type="xsd:int">120</ns1:period>
  </ns1:resourcePropertyElementProducers>
</ns1:resourcePropertyProviderConfiguration>
```

As you can see, there are a number of configurable parameters there such as the name of the RP that the data will be published as, the location of the MOTD file (`motd.xml`), as well as the period at which it will be checked for updates and refreshed. Feel free to edit those as necessary. In this example, any updates to the file will appear in the `Index Service` within 2 minutes (120 seconds). This should generally be sufficient for hand edited updates, but of course it can be configured to suit your needs.

At this point, the MOTD provider has been enabled for use with the RPProvider framework! As you can see, this framework simplifies much of the work of details of getting information into the `IndexService`. Simply restart the container and make sure there are no errors. If errors are reported, please check over all of the above steps and make sure that they were completed successfully. It's very easy to make unexpected typos!

5. An Example Query

```
neillm@macglob /usr/local/gt-current/bin $ ./wsrf-query -s
http://127.0.0.1:20202/wsrf/services/DefaultIndexService "//*[local-name()='MOTD']"

<myns:MOTD xmlns:myns="http://myorg.ns.for.testing">

  <myns:MessageEntry myns:Status="Urgent">
    <myns:DateEntered>Wed Nov 15 10:38:25 CST 2006</myns:DateEntered>
    <myns:DateValidUntil>Wed Nov 15 11:38:25 CST
    2006</myns:DateValidUntil>
    <myns:Content>
      URGENT:  It has come to our attention that a disk in the BBX
      machine is failing and we will be replacing it within the hour.
      Sorry for the inconvenience.
      -- Your friendly Systems Staff (systems@foo.org)
    </myns:Content>
  </myns:MessageEntry>

  <myns:MessageEntry myns:Status="Normal">
    <myns:DateEntered>Tue Nov 14 10:00:25 CST 2006</myns:DateEntered>
    <myns:DateValidUntil>--</myns:DateValidUntil>
    <myns:Content>
      Free donuts in the Coffee Room!  Come one, Come All!!  Get 'em
      while they're hot!
      -- Bob (bob@foo.org)
    </myns:Content>
  </myns:MessageEntry>

</myns:MOTD>
```

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

6. Contact the author

Contact the author at neillm@mcs.anl.gov¹.

¹ <mailto:neillm@mcs.anl.gov>

Chapter 5. Architecture and design overview

The rpprovider framework is comprised of three main sub-components:

- *Providers* - A simple execution environment for collections of programs (generally called *information providers*) which periodically output XML data that compose the values of a Resource Property. This execution environment consists of an GT4 Java operation provider that schedules periodic background tasks using the `org.globus.ws-
rf.impl.timer.TimerManagerImpl` class to execute information providers in a separate thread. When a timer event fires, the information provider code is invoked and the subsequent XML output ingested as Resource Property values.
- *Producers* - An API for creating plug-in (or adapter) modules for the execution environment and a standard set of basic information provider programs that can collect XML output using a variety of mechanisms, for example; reading from a file, reading the output of an HTTP GET operation, or executing a child process and parsing the standard output stream of the child process into valid XML, and others. These plug-in modules are generally called *element producers*, since the API interface which defines them (the `org.globus.mds.usefulrp.rppro-
vider.ResourcePropertyElementProducer` interface) uses a `org.w3c.dom.Element` as the return type.
- *Transforms* - An API for performing arbitrary post-processing on the output XML of element producers known as *element transforms*. Element transforms implement the `org.globus.mds.usefulrp.rpprovider.ResourcePropertyEle-
mentTransform` interface and can be configured to run against any given `ResourcePropertyElementProducer` via a configuration file setting. An example of an element transform is the included sample `XSLTFileElementTransform`, which will take the output of a `ResourcePropertyElementProducer` and apply an arbitrary XSL transform to it, using the XSLT file specified by the configuration argument.

1. Locating the code

The RPPProvider Framework code is part of the UsefulRP package of the ws-mds (CVS module) and can be seen under the `usefulrp/rpprovider/` directory. The producers are located in the `rpprovider/producers/` directory, and the transforms are located in the `rpprovider/transforms/` directory.

Chapter 6. Producers

1. The Current RPProviders

[The first major structural change - historical, do we need to mention that in new version?] is that an RPProvider package was created within the UsefulRP package. The intent of this [new] package was to provide some basic and flexible ways to insert arbitrary data into the [Index service](#) that are exposed as RPs. In particular, this package has a set of 'producers' that are used to get data into the Index service. For the first draft of the changes, we now have the following producers:

1. *External Process Element Producer* - this producer executes any arbitrary command or script that emits XML and it will be published as an RP.
2. *File Element Producer* - this producer publishes the contents of a file on local disk to the Index service as an RP.
3. *JVM Info Element Producer* - this producer is a useful testing mechanism that emits information about the host JVM as an RP.
4. *Scheduler Info Element Producer* - this producer executes the scheduler (batch) provider script for GRAM. This is a replacement for the inflexible code that previously executed the scheduler provider script.
5. *URL Element Producer* - this producer retrieves data from a configurable web based site and publishes the retrieved data into the Index as an RP.

The above producers allow flexibility regarding what data can be published in the Index service, how to publish that data done and exposing data as RPs from a variety of sources.

2. External Process Element Producer

TODO: make a similar section for each producer

To use the External Process Element Producer, add an entry in the following form to your \$GLOBUS_LOCA-TION//etc/globus_wsrf_mds_index/rp-provider-config.xml file.

```
<ns1:resourcePropertyName xsi:type="xsd:QName"
  xmlns:mds="http://mds.globus.org/glue/ce/1.1"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">mds:MyTestRP</ns1:resourcePropertyName>
<ns1:resourcePropertyImpl xsi:type="xsd:string"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">org.globus.mds.usefulrp.rpprovider.Si
<ns1:resourcePropertyElementProducers xsi:type="ns1:resourcePropertyElementProducerCon
  <ns1:className xsi:type="xsd:string"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">org.globus.mds.usefulrp.rpprovide
  <ns1:arguments xsi:type="xsd:string"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">/PATH/TO/MY/PROGRAM</ns1:argument
  <ns1:arguments xsi:type="xsd:string"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">DUMMY-ARGUMENT</ns1:arguments>
  <ns1:transformClass xsi:type="xsd:string"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">org.globus.mds.usefulrp.rpprovide
  <ns1:transformArguments xsi:type="xsd:string"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">transformDummyArgument</ns1:trans
```

```
<ns1:period xsi:type="xsd:int" xmlns:xsd="http://www.w3.org/2001/XMLSchema">30000
</ns1:resourcePropertyElementProducers>
```

Configuration settings for the External Process Element Producer

resourcePropertyName	Specifies the name of the <i>RP</i> that you'd like the data to be accessed by.
resourcePropertyImpl	Describes how the RP will be handled. The SingleValueResourcePropertyProvider is suitable for most uses.
resourcePropertyElementProducers	This element is required and wraps the following elements. For this type of RPPProvider, the className must be specified as org.globus.mds.usefulrp.rpprovider.producers.ExternalProcessElementProducer.

Configuration settings for resourcePropertyElementProducers

first argument	The first arguments element is the name of the script or program that you'd like to have executed. This script or program is expected to generate XML data that will be included in the Index.
second argument	The second arguments element is an arbitrary piece of data that will be provided to the process specified in the first argument as a (command line) argument.
transformClass	This element is optional (and may be excluded), but is provided in the case that you would like a post processing transform to occur on the output of the data generated by the process.
transformArguments	If the transformClass element was specified, the transformArguments element may be specified, which will be provided to the transformClass as a (command line like) argument.
element	The period element is required and specifies the interval at which to re-execute this external process and update the information it generates in the Index service.

3. File Element Producer

To use the File Element Producer, add an entry in the following form to your \$GLOBUS_LOCATION//etc/globus_wsrp_mds_index/rp-provider-config.xml file.

```
<ns1:resourcePropertyName xsi:type="xsd:QName"
  xmlns:frp="http://my.ns.domain/file"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">frp:MyFileRP</ns1:resourcePropertyName>
<ns1:resourcePropertyImpl xsi:type="xsd:string"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">org.globus.mds.usefulrp.rpprovider.SingleValueResourcePropertyProvider</ns1:resourcePropertyImpl>
<ns1:resourcePropertyElementProducers
  xsi:type="ns1:resourcePropertyElementProducerConfig">
  <ns1:className xsi:type="xsd:string"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">org.globus.mds.usefulrp.rpprovider.ExternalProcessElementProducer</ns1:className>
  <ns1:arguments xsi:type="xsd:string"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">/PATH/TO/MY/XML/FILE</ns1:arguments>
  <ns1:transformClass xsi:type="xsd:string" xsi:nil="true"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"/>
  <ns1:period xsi:type="xsd:int"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">60</ns1:period>
```

</ns1:resourcePropertyElementProducers>

Chapter 7. Transforms

[FIXME list and fully describe _each_ transform for rppprovider]

Transforms allow for post processing of the XML to be stored in the Index service.

[FIXME It was apparent that we'd need this step for the GLUE data, since a requirement became a complete re-organization of the data, based on site admin configurations. - this is 'historical' information and not necessary for people to know now?]

This [FIXME refer to it by actual name] particular transform involves parsing the GLUE XML document and bubbling up some elements to the SubCluster level, as well as re-organizing all of the Host elements into specified SubClusters.

Rather than demanding that all GLUE based information providers (i.e. Hawkeye, Ganglia, CluMon, etc) be able to do this organization, this post processing step can handle it regardless of what the input GLUE data looks like.

This is significant and allows site administrators to have more control than ever before on how to visualize their Cluster and SubClusters into meaningful groupings.

Architecturally, this involved adding a step in the UsefulRP package that routed all GLUE data to a post-processing phase, should it be configured for use.

By default, there is no post-processing that occurs for the compute element side of the data, however, we always post process the scheduler element data (to be consistent with the legacy code that converts the batch provider schema to the GLUE schema).

Finally, aside from GLUE specific data and transforms, there is a transform that exists that can optionally post process any arbitrary XML data with an XSLT transform. This allows for a natural and flexible post-processing step for any RP. Again, this is disabled by default, but can be configured for use if desired.

Chapter 8. APIs

1. Programming Model Overview

TBD

2. Component API

- [RPProvider Framework API](http://www.isi.edu/~mdarcy/globus/ws-mds/javadocs/org/globus/mds/usefulrp/rpprovider/package-summary.html)¹

¹ <http://www.isi.edu/~mdarcy/globus/ws-mds/javadocs/org/globus/mds/usefulrp/rpprovider/package-summary.html>

UsefulRP Admin Command

Name

`mds-gluerp-configure` -- generate a GLUE resource property configuration file with default values

`mds-gluerp-configure`

Tool description

`mds-gluerp-configure` is a simple utility tool for generating a configuration file for the GLUE resource property provider implementation. It can create a configuration with suitable default values for both cluster and scheduler information providers.

Command syntax

The basic syntax for **`mds-gluerp-configure`** is:

```
mds-gluerp-configure [scheduler info provider name | keyword none]
                    [cluster info provider name | keyword none] [outputFile - defaults to
                    gluerp-config.xml in the current dir]
```

where:

[scheduler info provider name]	One of the following keywords (ignoring case): fork, pbs, or none
[cluster info provider name]	One of the following keywords (ignoring case): ganglia, clumon, nagios, or none
[outputFile]	If this argument is specified, <code>mds-gluerp-configure</code> will write the configuration to file name and path, instead of the default.

Limitations

This command does not generate the correct configuration file for Nagios. However, you can use this command for HawkEye or Ganglia and modify for Nagios. Details are found [here](#).

Chapter 9. Domain-specific interface for UsefulRP/RPPProvider

TBD - describe domain-specific interface for usefulrp

Chapter 10. Configuring UsefulRP

1. Configuration overview

The system administrator first enables a given service or service resource to use the `org.globus.mds.usefulrp.rpprovider.ResourcePropertyProviderCollection` operation provider by adding the fully qualified Java class name to the provider's parameter value in the service descriptor of a service or resource's `server-config.wsdd` file.

Lastly, the administrator must add a new parameter named `rpProviderConfigFile` and for its corresponding value, specify a full (absolute) OS-native file path to a valid `ResourcePropertyProviderConfig` configuration file. The `ResourcePropertyProviderConfig` file contains all required information for generating one or more Resource Properties for the hosting service or resource.

At service startup, the `ResourcePropertyProviderCollection` operation provider code is initialized and attempts to process the configuration entries found in the file specified by the `rpProviderConfigFile` parameter into a set of one or more background execution tasks (threads) that periodically update the contents of configured Resource Properties with the results of the executing information providers. By default, if there are errors that occur during the first execution of a provider, the timer that controls that provider will be canceled and a warning message output to the container log.

Seen below is a sample service descriptor for the WS MDS `DefaultIndexService` which shows how to configure the service to use the `ResourcePropertyProviderCollection` operation provider and specifies the `rpProviderConfigFile` location used for configuring the sample `GLUEResourceProperty` that the `ResourcePropertyProviderCollection` will process.

```
<service name="DefaultIndexService" provider="Handler" use="literal" style="document">
  <parameter name="providers"
    value="org.globus.mds.usefulrp.rpprovider.ResourcePropertyProviderCollection
    org.globus.wsrfl.impl.servicegroup.ServiceGroupRegistrationProvider
    GetRPPProvider
    GetMRPPProvider
    QueryRPPProvider
    DestroyProvider
    SetTerminationTimeProvider
    SubscribeProvider
    GetCurrentMessageProvider"/>
  <parameter name="rpProviderConfigFile" value="/YOUR-GLOBUS-LOCATION-HERE/etc/globus_ws
  <parameter name="scope" value="Application"/>
  <parameter name="allowedMethods" value="*/>
  <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
  <parameter name="className" value="org.globus.mds.index.impl.DefaultIndexService"/>
  <wsdlFile>share/schema/mds/index/index_service.wsdl</wsdlFile>
</service>
```

1.1. Configuration File Format

The configuration file format for the `ResourcePropertyProviderCollection` operation provider is simply the XML-serialized form of the `ResourcePropertyProviderConfig` stub object, as defined in the schema file [rpprovider.xsd](#)¹.

Below is a sample configuration file which configures the GLUE Resource Property provider with element producers using Ganglia to provide cluster information and PBS for scheduler information. This sample configures the provider to generate cluster information using Ganglia on the localhost with the default Ganglia port, and configures PBS as the scheduler information provider. The period of execution is set to 300 seconds for each element producer, but may be configured separately if desired. This configuration mirrors a common information provider setup in the GT4 GRAM `ManagedJobExecutable` service. Using the `RPProvider` Framework, it is possible to generate this information in other services as well.

```
<ns1:ResourcePropertyProviderConfigArray
  xsi:type="ns1:ResourcePropertyProviderConfigArray"
  xmlns:ns1="http://mds.globus.org/rpprovider/2005/08"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ns1:resourcePropertyProviderConfiguration xsi:type="ns1:resourcePropertyProviderConf
    <ns1:resourcePropertyName xsi:type="xsd:QName" xmlns:mds="http://mds.globus.org/gl
    <ns1:resourcePropertyImpl xsi:type="xsd:string">org.globus.mds.usefulrp.rpprovider.G
    <ns1:resourcePropertyElementProducers xsi:type="ns1:resourcePropertyElementProduce
      <ns1:className xsi:type="xsd:string">org.globus.mds.usefulrp.glue.GangliaElementPr
      <ns1:arguments xsi:type="xsd:string">localhost</ns1:arguments>
      <ns1:arguments xsi:type="xsd:string">8649</ns1:arguments>
      <ns1:period xsi:type="xsd:int">300</ns1:period>
      <ns1:transformClass xsi:type="xsd:string">org.globus.mds.usefulrp.rpprovider.trans
    </ns1:resourcePropertyElementProducers>
    <ns1:resourcePropertyElementProducers xsi:type="ns1:resourcePropertyElementProduce
      <ns1:className xsi:type="xsd:string">org.globus.mds.usefulrp.rpprovider.producers.
      <ns1:arguments xsi:type="xsd:string">libexec/globus-scheduler-provider-pbs</ns1:ar
      <ns1:transformClass xsi:type="xsd:string">org.globus.mds.usefulrp.rpprovider.trans
      <ns1:period xsi:type="xsd:int">300</ns1:period>
    </ns1:resourcePropertyElementProducers>
  </ns1:resourcePropertyProviderConfiguration>
</ns1:ResourcePropertyProviderConfigArray>
```

It is possible to configure the `GLUEResourceProperty` provider to use alternate mechanisms for providing scheduler information by changing the `arguments` field that follows the `SchedulerInfoElementProducer` parameter to a string with a `GLOBUS_LOCATION` relative-path that indicates the GRAM scheduler adapter to use, for example, `libexec/globus-scheduler-provider-fork`.

TBD: It is also possible to pass parameters to the `GLUESchedulerElementTransform` that control even more advanced post-processing and sorting of results when generating GLUE 1.1 XML, e.g. Teragrid resorting code.

¹ http://viewcvs.globus.org/viewcvs.cgi/ws-mds/usefulrp/schema/schema/mds/usefulrp/rpprovider.xsd?rev=1.2.6.1&only_with_tag=wsmds_usefulrp_update_4_0_branch&content-type=text/vnd.viewcvs-markup

Chapter 11. Environment variable interface

TBD

Chapter 12. Debugging

Log output from UsefulRP 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](#).

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.

2. Enabling verbose logging

As described in the above section, configuration files need to be edited to enable logging at different levels. For example, to see all logging for the CAS server, the following lines need to be added:

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

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

Chapter 13. Related Documentation

[TBD describe]

Glossary

I

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

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

R

resource properties A resource is composed of zero or more resource properties which describe the resource. For example, a resource can have the following three resource properties: Filename, Size, and Descriptors. The resource properties are defined in the web service's WSDL interface description.