Red Hat Enterprise MRG 2

Grid Developer Guide

Developer-focused information for the Grid component of Red Hat Enterprise MRG

Edition 3

David Ryan
Red Hat Engineering Content Services

Cheryn Tan
Red Hat Engineering Content Services
cheryntan@redhat.com

Alison Young
Red Hat Engineering Content Services

Legal Notice

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701

Abstract

This book contains information on Aviary, an interface for MRG Grid job submission, management and queries. It is targeted at developers who are getting started with MRG Grid.

Table of Contents

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the Liberation Fonts set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keys and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a key, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from an individual key by the plus sign that connects each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to a virtual terminal.

The first example highlights a particular key to press. The second example highlights a key combination: a set of three keys pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find...** from the

> **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.
>
> The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.
>
> To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. Getting Help and Giving Feedback

2.1. Do You Need Help?

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal at http://access.redhat.com. Through the customer portal, you can:

- search or browse through a knowledgebase of technical support articles about Red Hat products.
- submit a support case to Red Hat Global Support Services (GSS).
- access other product documentation.

Red Hat also hosts a large number of electronic mailing lists for discussion of Red Hat software and

technology. You can find a list of publicly available mailing lists at https://www.redhat.com/mailman/listinfo. Click on the name of any mailing list to subscribe to that list or to access the list archives.

2.2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: http://bugzilla.redhat.com/ against the product **Red Hat Enterprise MRG.**

When submitting a bug report, be sure to mention the manual's identifier: *Grid_Developer_Guide*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Chapter 1. Overview

MRG Grid provides a web service interface for job submission, management and queries called Aviary. This interface is designed to remove some of MRG Grid's complexity and provide access using the common network data transport of HTTP. Aviary uses the Simple Object Access Protocol (SOAP) for request and response exchanges between MRG Grid, Aviary-enabled components, and web service clients. Web service clients can be developed using Java, Python, Ruby, or other languages.

Aviary is targeted at developers who wish to get started with MRG Grid. Developers can use Aviary without the depth of knowledge associated with MRG Grid's High Throughput Computing capabilities.

Chapter 2. API Types

2.1. SOAP and WSDL

The API types are described using the SOAP XML schema, and the operations use the Web Services Description Language (WSDL). This schema-based approach allows developers to use API types and operations in their preferred programming language. Some popular web service toolkits for client development are:

- Apache Axis or CXF for Java
- Suds for Python
- Savon for Ruby

2.2. Aviary Model

Entities of this API include ***job***, ***submission*** and ***attribute***.

A ***job*** is the basic unit of work and has a minimum set of attributes. These attributes include the full path of the command to be executed, command arguments, job owner, and requirements that provide information to MRG Grid. The requirements list enables matching with a resource that can execute the job.

A ***submission*** is an association of jobs under a common name key, such as ***my_submission_for_today***. Aviary can generate a submission name if one is not given.

An ***attribute*** describes aspects of a job. Some attributes can be set when the job is submitted or edited later when the job is still actively being processed in the MRG Grid job queue. MRG Grid will specify many job attributes after a submission but you can also provide custom attributes if they are meaningful to the execution of the application represented by a job.

Chapter 3. Aviary Installation and Configuration

3.1. Installation

RPM

Install the *condor-aviary* package for your platform. This will install the required software components, including the WSDL and schema files for Aviary. These files can be used to develop a remote web service client.

Important

Currently, due to a limitation in the underlying web service stack (Axis2/C), it is not possible to dynamically retrieve the WSDL and imported XSD over HTTP using the **`?wsdl`** URL syntax.

Source

Aviary can be included in a MRG Grid source build using the following variables when **`cmake`** is invoked:

```
-DWANT_CONTRIB:BOOL=TRUE -DWITH_AVIARY:BOOL=TRUE -DWITH_MANAGEMENT:BOOL=TRUE
```

3.2. Configuration

To enable Aviary use Remote Configuration to apply the following two features to your MRG Grid pool:

- ***`AviaryScheduler`*** - configuration to activate a component that provides the Aviary job submission and management capabilities
- ***`QueryServer`*** - configuration to activate a component that provides the Aviary job query capabilities

Refer to the *Remote Configuration* chapter in the *MRG Grid User Guide* for information on applying features to pools.

Chapter 4. Aviary Core Types

The XML schema defines core types that describe how Aviary operations are invoked and how results are interpreted.

4.1. *JobId*

A ***JobId*** is a unit of information that fully describes the identity of a job. It contains the following parameters:

- ***job*** - the local identifier for a job assigned to a specific scheduler. It is a string that encodes two positive integer numbers separated by a dot such as **1.0**, **84.3**, **2004.68**. The first is a reference to a local job grouping that may have multiple parts with attributes in common that are counted by the second number. A typical example is a group of jobs that share the same command but pass different arguments to the command, each job then writes its outputs to a different file.
- ***scheduler*** - a string that identifies which scheduler the job was submitted to.
- ***pool*** - a string that identifies a MRG Grid deployment. A deployment is an arena of schedulers, job execution resources and components that match jobs to those resources.
- ***submission*** - the ***SubmissionId*** related to the job.

4.2. *SubmissionId*

A ***SubmissionId*** is a unit of information that describes a submission in the following two parts:

- ***name*** - a string provided by the user or generated on behalf of the user at the time of submission. Submission names may be considered a way to associate and aggregate jobs in such a way that is meaningful to the developer. An example of a meaningful name may be ***my_submission_04302011***. As submissions are open-ended, a user can continue to add individual jobs to this aggregating name over time. This remains true though the individual jobs may have been scheduled and executed at different times by MRG Grid. For example, the jobs ***1.0***, ***28.0*** and ***2011.0*** could all be part of the submission named ***my_submission_04302011***.
- ***owner*** - a string containing the name of the original submitter.

4.3. *Attribute*

An ***attribute*** is type-coded information used by MRG Grid to evaluate, organize and execute job matching and processing. MRG Grid jobs have multiple attributes, some are user-specified before submission and many that are attached to a job by the MRG Grid infrastructure when added to the job queue. A MRG Grid job is the sum of its attributes. An ***attribute*** consists of:

- ***name*** - a string denoting the attribute name. Names can be predefined and understood by the MRG Grid infrastructure or a custom attribute name.
- ***type*** - an enumerated string with string, integer, float, expression or boolean values.
- ***value*** - the string form of the value.

4.4. *JobStatus*

A ***JobStatus*** exists in one of the following states:

- ***idle*** - the job is in a state where it is not ready or able to be assigned to a resource.

- ***running*** - the job is assigned to and running on a resource.
- ***held*** - the job exists in the MRG Grid queue but is held back from execution.
- ***completed*** - the job ran to completion.
- ***removed*** - the job was deleted from the job queue by a user.
- ***suspended*** - the job has its execution paused at the resource. This depends on a job's ability to respond to relevant platform-specific signals.
- ***transferring_output*** - the job has completed and is in the process of transferring output files back to the submission host.

4.5. *ResourceConstraint*

A ***ResourceConstraint*** is a basic quality that MRG Grid should consider when matching a new job to a resource. There are five basic constraints defined in Aviary which are:

- ***OS*** - ***Linux*** or ***Windows***.
- ***ARCH*** - for 32-bit platforms, ***INTEL***; or ***X86_64*** for 64-bit platforms. This is important when the executable needed by the job is compiled for a particular architecture.
- ***MEMORY*** - the expected total RAM required to execute the job.
- ***DISK*** - expected total disk space to execute the job.
- ***FILESYSTEM*** - the domain name representing a uniformly mounted network file system, as configured by a MRG Grid administrator.

4.6. *ResourceID*

A ***ResourceID*** describes location attributes for an Aviary endpoint such as the Scheduler or QueryServer.

- ***resource*** - a parameter describing the class of Aviary endpoint. It can be one of ***ANY***, ***COLLECTOR***, ***CUSTOM***, ***MASTER***, ***NEGOTIATOR***, ***SCHEDULER***, ***SLOT***
- ***pool*** - the name of the pool to which a Collector endpoint belongs.
- ***name*** - a modified version of the Grid daemon's name.
- ***sub_type*** - a parameter used to provide further classification to a ***resource***.

Chapter 5. Aviary Locator

Aviary provides a bootstrap WSDL interface for finding other Aviary SOAP endpoints called the *locator*. This is implemented within a **condor_collector**-specific plugin which receives generic ClassAds from Aviary endpoint publishers that contain a fully-formed base URL. This base URL can be used by a client to invoke their target services, but the client must still append the appropriate target operation to the end of the URL.

When a locator is configured, Aviary endpoints will bind to ephemeral ports on their respective hosts and publish their addressable URL to the locator plugin. Upon graceful exit, endpoints will notify the locator plugin through ad invalidation. For failed endpoints, the plugin monitors regular updates of the published ads and prunes them using a policy of missed update count over a configurable interval.

The default locator can be invoked at port 9000 on the Collector host (e.g., http://localhost:9000/services/locator/locate).

A sample command to run the Aviary locator is as follows:

Example 5.1. Sample Aviary locator command

```
$ ./locator.py --type CUSTOM --custom QUERY_SERVER --name user@hostname.com
```

Three arguments were provided in the sample command: the ***ResourceType***, ***SubType***, and ***Name***.

- ***--type*** – A required argument. The resource type can be either one of the following: ***ANY***, ***COLLECTOR***, ***CUSTOM***, ***MASTER***, ***NEGOTIATOR***, ***SCHEDULER*** or ***SLOT***.
- ***--custom*** – An optional argument. In the example given, the resource type specified is ***CUSTOM***, therefore the subtype of ***CUSTOM*** resources must also be specified. In this case, it is the ***QUERY_SERVER***. This field must always be the full string of the resource subtype.
- ***--name*** – An optional argument. It can be a fragment or the exact full name of the endpoint host. Exact matching can be used in the locate request by setting the XML attribute ***partialMatches=false***.

Note

For the full list of options available for the Aviary locator, run

```
$ ./locator.py --help
```

Chapter 6. Job Submission and Management

Job Management is used for job control and reporting. Methods in job management include job submission, hold, release and removal.

Table 6.1. Job Submission and Management Operations

Operation	Inputs	Outputs	Notes
submitJob	Job submission request fields are: • ***cmd*** - a string containing the absolute path to an executable or script • ***owner*** - a string identifying the submitter • ***iwd*** - the initial working directory where the job will be executed • ***args*** - an optional string containing arguments for the ***cmd*** • ***submission_name*** - an optional string identifying the submission that should be created or that this job is to be attached to • ***requirements*** - an optional list of **ResourceConstraints** that specify what type of resource this job should be targeted at • ***extra*** - optional list of ***Attributes*** that refine the request beyond basic fields or supersede the MRG Grid Attributes implied by other basic fields in this request	***'OK'*** and the ***JobId*** or an error containing diagnostic text if a problem was encountered.	MRG Grid users familiar with crafting specific ***attributes*** such as complex requirements may do so using the ***extra*** attribute field in conjunction with the **allowOverrides** XML attribute in the request.
holdJob	A single ***JobId*** and a hold reason in string format.	***'OK'*** or an error with text if the job is not found or parsed.	A hold is a temporary interruption of job execution against a resource; holds can be used to affect job attribute edits without needing to resubmit the job.
releaseJob	A single ***JobId*** and a	***'OK'*** or an error with	Releasing a job is

	release reason in string format.	text if the job is not found or parsed	moving it out of the held state and back where it is ready to be schedule again with a resource.
removeJob	A single ***JobId*** and a remove reason in string format.	***'OK'*** or an error with text if the job is not found or parsed.	Job removal means that the job is prevented from executing to completion, note that its existence in the MRG Grid queue is still maintained on record.
suspendJob	A single ***JobId*** and a suspend reason in string format.	***'OK'*** or an error with text if the job is not found or parsed.	A job can be suspended depending on its ability to respond to platform-specific interrupt signals. A suspended job remains with a claimed resource until it is continued or removed.
continueJob	A single ***JobId*** and a continue reason in string format.	***'OK'*** or an error with text if the job is not found or parsed.	A job that has been suspended will be signalled to continue executing on its original resource host.
setJobAttribute	A single ***JobId*** and a single ***Attribute***.	***'OK'*** or an error with text is the job is not found or parsed.	Attributes are predefined by MRG Grid or can be user-created, for example a name/type/value shorthand combination: ▸ ***JobPrio*/*INTEGER*/*2*** would be shorthand for the job attribute predefined by MRG Grid to control job priority, set to a value of 2 giving it higher priority than the default of 0 ▸ ***Recipe*/*STRING*/*secret sauce*** would be shorthand for a custom job attribute provided by a user, meaningful to only their application and irrelevant to the MRG Grid infrastructure

Important

Users of condor_submit should keep the following differences between **condor_submit** and Aviary job submission in mind:

- The use of **$(*expr*)** syntax for evaluating expressions at submission time is not supported in Aviary. However, the related **$$(*expr*)** syntax for evaluating expressions at match time is supported.
- Unlike **condor_submit**, Aviary will not filter the keyword **error**, which is a ClassAd reserved word. In Aviary, use **err=** instead of **error=**.

6.1. submitJob Extra Attributes

The ***extra*** attributes are an advanced feature provided so that a submitter can apply more fine-grained details to their job advertisement. Details are expressed in the ClassAd language, more information is available in the ClassAd chapter of the *MRG Grid User Guide*. Attributes are described above in Section 4.3, "***Attribute***".

Consider the following attributes expressed in ClassAd syntax:

```
Output = /tmp/myjob.out
```

In Aviary the same details are expressed as:

```
<extra>
 <name>Out</name>
 <type>STRING</type>
 <value>/tmp/myjob.out</value>
</extra>
```

A user can override any of the basic job attributes by setting the **allowOverrides** XML attribute on the **submitJob** element to ***true***. Required attributes are:

- **cmd**
- **owner**
- **iwd**

An additional job attribute type **requirements** can be specified. It will be given a default value if none is specified.

If a different value is submitted for a basic attribute in the extra list and **allowOverrides** is set to ***false***, it will be ignored. In this case the value in the **submitJob** basic attribute will be used. For example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:job="http://job.aviary.grid.redhat.com">
 <soapenv:Header/>
 <soapenv:Body>
  <job:SubmitJob allowOverrides="true">
   <cmd>/bin/sleep</cmd>
   <args>40</args>
   <owner>condor</owner>
   <iwd>/tmp</iwd>
   <requirements>
    <type>OS</type>
    <value>LINUX</value>
   </requirements>
   <extra>
    <name>Owner</name>
    <type>STRING</type>
    <value>somebody</value>
   </extra>
  </job:SubmitJob>
 </soapenv:Body>
</soapenv:Envelope>
```

The value ***somebody*** is used instead of ***condor*** as an override of the basic attribute has been specified.

As extra attributes are designed to provide tuning for job submission, below is a slightly more complex use case. In the example, there is a job that can use all of the CPUs on a machine where HTCondor is configured for dynamic slots. Dynamic slots are explained in the *MRG Grid User Guide*.

The basic requirements field is simplified for common use cases so a more detailed requirements value is needed for matching like in the following example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:job="http://job.aviary.grid.redhat.com">
 <soapenv:Header/>
 <soapenv:Body>
  <job:SubmitJob allowOverrides="true">
   <cmd>/bin/sleep</cmd>
   <args>40</args>
   <owner>condor</owner>
   <iwd>/tmp</iwd>
   <extra>
    <name>Requirements</name>
    <type>EXPRESSION</type>
    <value>&amp;quot;(TARGET.TotalCpus =!= UNDEFINED)
    &amp;amp;&amp;amp; ((Target.PartitionableSlot =?= TRUE) ||
    (TARGET.DynamicSlot =?= TRUE)&amp;quot;</value>
   </extra>
   <extra>
    <name>RequestCpus</name>
    <type>EXPRESSION</type>
    <value>TotalCpus </value>
   </extra>
  </job:SubmitJob>
 </soapenv:Body>
</soapenv:Envelope>
```

If **`allowOverrides`** had not been specified ***`true`***, the **`Requirements`** field would be given a default value of ***`true`***.

Chapter 7. Job Data Queries

Table 7.1. Job Queries for Data

Operation	Inputs	Outputs	Notes
getJobStatus	Zero to many ***JobId***s.	Returns the current status for each ***JobId*** input, or an error indicating that the job could not be parsed or found.	The most efficient query as it returns the least amount of data per job.
getJobSummary	Zero to many ***JobId***s.	Returns a summary for each ***JobId*** input, or an error indicating that the job could not be parsed or found.	Summary returned includes: • command • command arguments • scheduler local time when job was added to job queue • scheduler local time of last update to job status • job status • reason why job was held, released or removed
getJobDetails	Zero to many ***JobId***s.	Returns all ***Attributes*** for each ***JobId*** input, or an error indicating the job could not be parsed or found.	A potentially expensive operation, it is possible to request all the attributes for all the jobs tracked in MRG Grid. If performance is a concern consider judicious use of summaries for certain job sets.
getJobData	A single ***JobId***, the type of data file content requested (ERR, OUT, LOG), the maximum number of bytes to be returned and whether the file should be read from the front to back.	Returns the file content requested if successful.	Each job can specify an error file (ERR), a log file (LOG) or an output file (OUT); the log file is used by MRG Grid to monitor the job progress.
getSubmissionSummary	Zero to many ***SubmissionId***s.	For each valid submission returned, these job totals will be listed: • completed • held • idle • removed • running • suspended • transferring_output	Individual job summaries can be included in the response by setting the XML attribute ***includeJobSummaries*** to true in the request.

getSubmissionIDs	The inputs for this query will determine whether a lexical search or a search based on submission age is performed: • size • a ***ScanMode*** of BEFORE or AFTER (optional) • a ***SubmissionID*** offset (optional)	The list of submission IDs returned from this query will be limited by the page size that was requested. If there are more IDs to be returned beyond the ***size*** limit then that amount will be reflected in the ***remaining*** value.	This query returns a lexical sort of submission IDs by default, but can also be used to return IDs that are sorted by their age in increasing or decreasing order. The use of a submission ID offset can be thought of as a random-access pointer to particular regions of the overall ID list. This offset can be applied as appropriate for lexical or age queries.

7.1. Queries And Scale Considerations

Submissions in Aviary are a logical grouping of one or more jobs. They make a convenient entity for an organized, top-down view of the jobs in a pool. There are two query operations noted in Table 7.1, “Job Queries for Data” that can return lists of ***SubmissionIDs***. They can be used together to help a client scale the management of submissions as their volume increases.

- ***getSubmissionIDs*** - this operation helps clients discover unknown or unloaded submission IDs and limit the number of IDs returned. Therefore, it is ideal for situations where a client needs to learn about new submissions from a known point-in-time, or incrementally load older submission IDs.
- ***getSubmissionSummary*** - this operation can accept zero or more submission IDs and return either a summary for the submission itself or optionally also include summaries for each job within that submission. It is better suited to tracking a set of known submission IDs.

Although ***getSubmissionSummary*** with no input arguments can return all the submissions known to a Scheduler, this way of updating submission data becomes less and less efficient as the number of submissions grow. Since IDs returned from the ***getSubmissionIDs*** operation can be used as arguments for ***getSubmissionSummary***, it is more efficient to collect IDs first then query the summary data (including possibly job data) if requested by a user.

This same principle holds true for the job-related query operations. Since most users would be typically be interested in a job's status (RUNNING or not), it makes more sense to get the job summary once then update its status by calls to ***getJobStatus***. By the same token, it would be perhaps the less common case that they need to look at the detailed information provided by ***getJobDetails***.

Chapter 8. Security

Aviary supports job submissions, management and queries over a secure SSL connection with mutual authentication. To enable this advanced feature, use remote configuration to add the **`SSLEnabledAviaryScheduler`** and **`SSLEnabledQueryServer`** features. These features contain parameters for specifying the file locations of server certificates, key files, and a certificate authority (CA) file or directory. These certificates are expected to be in PEM format, the default used by OpenSSL.

Chapter 9. Client Examples

Aviary clients can be developed using a variety of programming languages. Below are code examples of client actions using SOAP XML, Ruby and Python.

9.1. SOAP XML

The following example shows the request and response SOAP XML for a job submission.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:job="http://job.aviary.grid.redhat.com">
   <soapenv:Header/>
   <soapenv:Body>
      <job:SubmitJob allowOverrides="false">
         <cmd>/bin/sleep</cmd>
         <!--Optional:-->
         <args>40</args>
  <owner>ownername</owner>
         <iwd>/tmp</iwd>
         <!--Optional:-->
  <submission_name>my_submission</submission_name>
         <!--Zero or more repetitions:-->
         <requirements>
            <type>OS</type>
            <value>LINUX</value>
         </requirements>
         <!--Zero or more repetitions:-->
         <extra>
            <name>MYDATA</name>
            <type>STRING</type>
     <value>the data</value>
         </extra>
      </job:SubmitJob>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <n:SubmitJobResponse xmlns:n="http://job.aviary.grid.redhat.com">
         <id>
            <job>247.0</job>
            <pool>localhost</pool>
     <scheduler>username@localhost.localdomain</scheduler>
            <submission>
               <name>my_submission</name>
        <owner>username</owner>
            </submission>
         </id>
         <status>
            <code>OK</code>
            <text/>
         </status>
      </n:SubmitJobResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

9.2. Ruby

The following example shows a Ruby Savon web service client that generates a basic submission.

```
# uses Savon http://savonrb.com/
require 'rubygems'
# httpi >= 0.9.2
require 'httpi'
# savon >= 0.9.1
require 'savon'
require "openssl"

client = Savon::Client.new do |wsdl|
  wsdl.document = "/var/lib/condor/aviary/services/job/aviary-job.wsdl"
  wsdl.endpoint = "http://localhost:9090/services/job/submitJob"
end

xml =  Builder::XmlMarkup.new
xml.cmd("/bin/sleep")
xml.args("40")
xml.owner("condor")
xml.iwd("/tmp")

response = client.request :job, "SubmitJob" do
    soap.namespaces["xmlns:job"] = "http://job.aviary.grid.redhat.com"
    soap.body = xml.target!
end
```

9.3. Python

The following example shows a Python Suds web service client that invokes a job query operation based on user input. This operation also takes an optional ***JobID*** argument.

```
# uses Suds - https://fedorahosted.org/suds/
import logging
from suds import *
from suds.client import Client
from sys import exit, argv, stdin
import time

# enable these to see the SOAP messages
#logging.basicConfig(level=logging.INFO)
#logging.getLogger('suds.client').setLevel(logging.DEBUG)

# change these for other default locations and ports
job_wsdl = 'file:/var/lib/condor/aviary/services/query/aviary-query.wsdl'

cmds = ['getJobStatus', 'getJobSummary', 'getJobDetails']

cmdarg = len(argv) > 1 and argv[1]
cproc =  len(argv) > 2 and argv[2]
job_url = len(argv) > 3 and argv[3] or "http://localhost:9091/services/query/"

if cmdarg not in cmds:
    print "error unknown command: ", cmdarg
    print "available commands are: ",cmds
    exit(1)

client = Client(job_wsdl);
job_url += cmdarg
client.set_options(location=job_url)

# enable to see service schema
#print client

# set up our JobID
if cproc:
    jobId = client.factory.create("ns0:JobID")
    jobId.job = cproc
else:
    # returns all jobs
    jobId = None

try:
    func = getattr(client.service, cmdarg, None)
    if callable(func):
        result = func(jobId)
except Exception, e:
    print "invocation failed: ", job_url
    print e
    exit(1)

print result
```

Chapter 10. More Information

Reporting Bugs

Follow these instructions to enter a bug report:

1. Create a Bugzilla account.
2. Log in and click on Enter A New Bug Report.
3. You will need to identify the product (Red Hat Enterprise MRG), the version (2.2), and whether the bug occurs in the software (component=grid) or in the documentation (component=Grid_Developer_Guide).

Further Reading

Red Hat Enterprise MRG and MRG Grid Product Information

http://www.redhat.com/mrg

***MRG Grid User Guide* and other Red Hat Enterprise MRG manuals**

http://docs.redhat.com/docs/en-US/index.html

HTCondor Manual

http://research.cs.wisc.edu/htcondor/manual/

Red Hat Knowledgebase

https://access.redhat.com/knowledge/search

Revision History

Revision 3.0-2	**Wed Jan 30 2013**	**David Ryan**

Various edits including BZ#756254 for Suspend/Continue actions for jobs.

Revision 3.0-1	**Fri Jan 26 2013**	**David Ryan**

BZ#882822 - Updating HTCondor product name.

Revision 3.0-0	**Fri Sep 14 2012**	**Cheryn Tan**

Prepared for publishing (MRG 2.2).

Revision 2-4	**Fri Aug 10 2012**	**Cheryn Tan**

Rebuild with Publican 3.0.

Revision 2-3	**Thu Jul 17 2012**	**Cheryn Tan**

BZ#702492 - Added admonition on job submission for condor_submit users.

Revision 2-1	**Tue Feb 28 2012**	**Tim Hildred**

Updated configuration file for new publication tool.

Revision 2-0	**Tue Dec 6 2011**	**Alison Young**

Prepared for publishing

Revision 1-9	**Mon Nov 21 2011**	**Alison Young**

BZ#722996 - extra attribute usage in Aviary submits
BZ#754451 - 2.1 engineering review and feedback

Revision 1-8	**Tue Nov 15 2011**	**Alison Young**

BZ#752406 - change RHEL versions

Revision 1-7	**Fri Nov 11 2011**	**Alison Young**

BZ#722996 - updated example code

Revision 1-5	**Thu Oct 20 2011**	**Alison Young**

BZ#722996 - updated security chapter

Revision 1-4	**Wed Oct 19 2011**	**Alison Young**

BZ#722996 - Advanced extra attribute usage in Aviary submits
BZ#732390 - Missing submission as parameter of jobID

Revision 1-3	**Wed Sep 07 2011**	**Alison Young**

Prepared for publishing

Revision 1-2	**Tue Aug 23 2011**	**Alison Young**

BZ#731649 - Change getSubmissionSummaries to getSubmissionSummary

Revision 1-1	**Thu Jun 23 2011**	**Alison Young**

Prepared for publishing

Revision 1-0	**Thu Jun 23 2011**	**Alison Young**

Prepared for publishing

Revision 0.1-5 **Thu Jun 02 2011** **Alison Young**

BZ#674385 - Minor updates

Revision 0.1-4 **Mon May 09 2011** **Alison Young**

Minor XML updates

Revision 0.1-3 **Thu May 05 2011** **Alison Young**

BZ#674385- Restructured book and inserted additional source content provided

Revision 0.1-2 **Wed Mar 30 2011** **Alison Young**

Inserted Submit, Hold and Release method descriptions in Job Management

Revision 0.1-1 **Thu Mar 3 2011** **Alison Young**

Added skeleton chapters and sections

Revision 0.1-0 **Thu Mar 3 2011** **Alison Young**

Initial creation of book by publican