Registry Service v.1.0.0

for the Planetary Data System

NASA Jet Propulsion Laboratory,
California Institute of Technology

10 November 2011

Table of Contents

1.1 Overview

About Registry Service

The Registry Service provides functionality for tracking, auditing, locating, and maintaining artifacts within the system. The service provides a REST-based interface for interacting with the service.

Please send comments, change requests and bug reports to the PDS Operator at pds_operator@jpl.nasa.gov.

1.2 Installation

Installation

This section describes how to install the Registry Service software contained in the *registry-service* package. The following topics can be found in this section:

- System Requirements
- Unpacking the Package
- Database Creation
- Deploying the Application
- Configuring the Application

System Requirements

This section details the system requirements for installing and operating the Registry Service.

Java Runtime Environment

The Registry Service was developed using Java and Jersey and will run on any platform with a supported Java Runtime Environment (JRE). The software was specifically developed under Java version 1.6 and has only been tested with this version. The following commands test the local Java installation in a UNIX-based environment:

```
% which java
/usr/bin/java

% java -version
java version "1.6.0_26"
Java(TM) SE Runtime Environment (build 1.6.0_26-b03-384-10M3425)
Java HotSpot(TM) 64-Bit Server VM (build 20.1-b02-384, mixed mode)
```

The first command above checks whether the *java* executable is in the environment's path and the second command reports the version. If Java is not installed or the version is not at least 1.6, Java will need to be downloaded and installed in the current environment. Consult the local system administrator for installation of this software. For the do-it-yourself crowd, the Java software can be downloaded from the Oracle Java

Download page. The software package of choice is the Java Standard Edition (SE) 6, either the JDK or the JRE package. The JDK package is not necessary to run the software but could be useful if development and compilation of Java software will also occur in the current environment.

Java Application Server

The Registry Service requires a Java application server for hosting the web application. The suggested application server for this release is Apache Tomcat with a minimal version of 6.0.20 through version 7.0.X. Consult the local system administrator for installation of this software. For the do-it-yourself crowd, the Apache Tomcat software can be downloaded from the Apache Tomcat page. Choose the version to download (6.0 or 7.0) from the menu on the left.

cURL Utility

Although it is generally a useful tool for interacting with the Registry Service, the cURL command-line application is required by the *RegistryConfig* script for populating the service with the supported object types. See the Configuration section for more information on configuring the service. If *cURL* is not installed on the local machine but *Wget* is, see the Using Wget section for converting *cURL* commands to *Wget* commands. If viewing this document in PDF form, see the appendix for details.

Database Server

The Registry Service comes prepackaged with the Apache Derby database. If another database solution is desired (i.e., MySQL), it should be installed and accessible by the Registry Service. See the Database Creation section if this is the first time you are installing the registry service and Configuration section for more information on configuring the service for a different database solution.

Unpacking the Package

Download the *registry-service* package from the PDS FTP site. The binary distribution is available in identical zip or tar/gzip packages. Unpack the selected binary distribution file with one of the following commands:

```
% unzip registry-service-1.0.0-bin.zip
or
% tar -xzvf registry-service-1.0.0-bin.tar.gz
```

Note: Depending on the platform, the native version of *tar* may produce an error when attempting to unpack the distribution file because many of the file paths are greater than 100 characters. If available, the GNU version of tar will resolve this problem. If that is not available or cannot be installed, the zipped package will work just fine in a UNIX environment.

The commands above result in the creation of the *registry-service-1.0.0* directory with the following directory structure:

- **README.txt**
  A README file directing the user to the available documentation for the project.

- **LICENSE.txt**
  The copyright notice from the California Institute of Technology detailing the restrictions regarding the use and distribution of this software. Although the license is strictly worded, the software has been classified as Technology and Software Publicly Available (TSPA) and is available for *anyone* to download and use.

- **registry-service-1.0.0.war**
  This is the Web ARchive (WAR) file containing the Registry Service software including all dependent JAR files.

- **bin/**
  This directory contains the batch and shell scripts for registering the supported object types.

- **lib**
  This directory contains jar files for working with the Derby Database.

- **conf/**
  This directory contains the policy files identifying the supported object types.

- **doc/**
  This document directory contains a local web site with the Registry Service Guide, javadoc, unit test results and other configuration management related information. Just point the desired web browser to the *index.html* file in this directory.

- **examples/**
  This directory contains examples of artifact descriptions that can be registered with a service instance.

Database Creation

If this is the first time the Registry Service has been installed then a new registry database must be created. If however, you already have an existing registry skip down to the Configuration section. The following is a set of instructions that are dependent on the type of database chosen. Currently, there is support for Derby and MySQL.

Derby

The Derby database comes packaged with the registry service. The following commands will create the database and install the registry schema:

```
% cd registry-service-1.0.0
% java -Djava.ext.dirs=lib/ org.apache.derby.tools.ij
ij> connect 'jdbc:derby:RegistryDB;create=true;user=user';
ij> run 'conf/derby-registry-schema.ddl';
ij> exit;
```

The RegistryDB directory, which is the Derby, will now be present in the current working directory. The path to this directory should be used in the Configuration section. Note: Feel free to move this RegistryDB directory to anywhere on disk that is accessible by Tomcat and configured below.

MySQL

```
% mysqladmin -uroot -p create registry
% mysql registry -uroot -p -e"GRANT ALL ON registry.* TO registry@localhost \
  IDENTIFIED BY 'p@ssw0rd'"
% mysql -u root -p -D registry < conf/mysql-registry-schema.ddl
```

Deploying the Application

The Registry Service web application is packaged as a WAR file and is intended for installation under a standard Java Application Server. Prior to installation, the WAR file should be renamed from *registry-service-1.0.0.war* to *registry.war*. For a Tomcat server deployment, the WAR file is normally copied directly to the *$TOMCAT_HOME/webapps* directory or installed via the Manager interface. Once this step is complete, the application is ready for operation. Verify a successful installation by executing the command from the Ping portion of the Operation section.

When deploying the application via the Tomcat Manager interface, users have occasionally encountered a situation where the application appears to hang or generates the following stack trace:

```
javax.servlet.ServletException: java.lang.OutOfMemoryError: PermGen space
  com.sun.jersey.spi.container.servlet.WebComponent.service(WebComponent.java:424)
com.sun.jersey.spi.container.servlet.ServletContainer.service(ServletContainer.java:497)
com.sun.jersey.spi.container.servlet.ServletContainer.doFilter(ServletContainer.java:855)
com.sun.jersey.spi.container.servlet.ServletContainer.doFilter(ServletContainer.java:828)
com.sun.jersey.spi.container.servlet.ServletContainer.doFilter(ServletContainer.java:789)
  org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter.doFilterInternal
  (OpenEntityManagerInViewFilter.java:113)
org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:76)
```

If either of the above situations occur, stop and restart the Tomcat server to clear the problem.

Configuring the Application

The following steps configure the Registry Service for the target installation and ready the service for operation.

Database Configuration

By default, the Registry Service comes packaged with and configured to utilize Derby as the backend database. The Derby database home directory will default to the current working directory where the Tomcat server was launched. To permanently set the home directory of the database, add the following to the *CATALINA_OPTS* environment variable:

```
CATALINA_OPTS="-Dderby.system.home=/path/to/registrydb/home"
```

The *CATALINA_OPTS* environment variable is loaded from the Tomcat startup scripts. The Tomcat server will need to be restarted for this configuration to take effect.

The backend database can be changed from Derby to another database provider. As of the current release, MySQL is the only other supported database solution. To modify the configuration, edit the *applicationContext.xml* file located in the *$TOMCAT_HOME/webapps/registry/WEB-INF/classes* directory. The following line:

```
<context:property-placeholder location="classpath:derby.properties"/>

  should be changed to:

<context:property-placeholder location="classpath:mysql.properties"/>
```

The default configuration assumes that you have MySQL installed with a database named *registry* and will use a default user name and password as specified below. If you want to change the URL, database name, user name, and/or password you will need to edit the *mysql.properties* file located in the *$TOMCAT_HOME/webapps/registry/WEB-INF/classes* directory. The following lines pertain to the default configuration:

```
javax.persistence.jdbc.url=jdbc:mysql://localhost:3306/registry
javax.persistence.jdbc.user=registry
javax.persistence.jdbc.password=p@ssw0rd
```

Additionally, if you are using a version of MySQL older than 5.x you will need to change the dialect. To do this simply add a "#" before the first hibernate.dialect entry and remove the "#" from the second entry.

Before:

```
# For use with MySQL 5+
hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
# For use with older versions of MySQL. See hibernate documentation.
#hibernate.dialect=org.hibernate.dialect.MySQLInnoDBDialect
```

After:

```
# For use with MySQL 5+
#hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
# For use with older versions of MySQL. See hibernate documentation.
hibernate.dialect=org.hibernate.dialect.MySQLInnoDBDialect
```

No matter which database solution is configured as the backend database, the schema that supports the Registry Service is created by default when launching the service for the first time.

Home Configuration

In a distributed environment with multiple Registry Service instances, the registry home value identifies the source of a registry entry when it has been replicated to another registry instance. By default, the registry home is configured as *http://localhost:8080/registry*. This should be modified to represent the Registry Service endpoint of the local installation. Meaning that instead of *localhost*, the fully qualified name of the local machine should be specified (e.g., node.nasa.gov). To modify the configuration, edit the *applicationContext.xml* file located in the *$TOMCAT_HOME/webapps/registry/WEB-INF/classes* directory. Modify the following line with the new endpoint:

```
<bean id="idGenerator"
class="gov.nasa.pds.registry.model.naming.DefaultIdentifierGenerator"
p:home="http://localhost:8080/registry"/>
```

```
  should be changed to (if hosted at node.nasa.gov on port 80):

<bean id="idGenerator"
class="gov.nasa.pds.registry.model.naming.DefaultIdentifierGenerator"
p:home="http://node.nasa.gov/registry"/>
```

Object Type Configuration

Once the Registry Service is installed and running, the list of supported object types must be registered with the service. The list of objects types corresponds with the types of products that a given instance of the Registry Service will support. The *RegistryConfig* and *RegistryConfig.bat* scripts default to a Registry Service endpoint of *http://localhost:8080/registry*. If necessary, modify the appropriate script for the local environment so that it corresponds with the endpoint of the target installation. In addition, this script should be executed prior to applying security to the service URLs since it does not account for a secured interface. Execute the script from the *bin* directory in order to register the full set of object types:

```
% cd registry-service-1.0.0/bin
% ./RegistryConfig
```

The output from this command should show the registration of the Core object types and PDS object types. Since the configuration files referenced in the configuration script are slightly larger they are sent in chunks. Each configuration will get associated with a Registry Package and can be found by following the location link that comes in the header of the response. The output from the command should look something like the following:

```
* About to connect() to localhost port 8080 (#0)
*   Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> POST /registry/configure?name=Core+Objects&\
description=This+configures+the+core+set+of+registry+objects HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8l
zlib/1.2.3
> Host: localhost:8080
> Accept: */*
> Content-type:application/xml
> Content-Length: 5295
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
< HTTP/1.1 201 Created
< Server: Apache-Coyote/1.1
< Location: http://localhost:8080/registry/packages/\
urn:uuid:bd6e4f7b-dfb0-443c-b845-3378077b1016
< Content-Type: text/plain
< Transfer-Encoding: chunked
< Date: Mon, 21 Mar 2011 19:55:52 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
urn:uuid:bd6e4f7b-dfb0-443c-b845-3378077b1016

* About to connect() to localhost port 8080 (#0)
*   Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> POST /registry/configure?name=PDS+Objects&\
description=This+configures+PDS+object+types HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8l
zlib/1.2.3
> Host: localhost:8080
> Accept: */*
> Content-type:application/xml
> Content-Length: 18320
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
< HTTP/1.1 201 Created
< Server: Apache-Coyote/1.1
< Location: http://localhost:8080/registry/packages/\
urn:uuid:a07ad134-42ad-4781-9cbd-826bb9a8dfec
< Content-Type: text/plain
< Transfer-Encoding: chunked
< Date: Mon, 21 Mar 2011 19:55:53 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
urn:uuid:a07ad134-42ad-4781-9cbd-826bb9a8dfec
```

Verify successful configuration by executing the command from the Report portion of the Operation section. The output from this command should look something like the following:

```
<ns2:report xmlns:ns2='http://registry.pds.nasa.gov' registryVersion='1.0.0'
packages='2'
classificationNodes='67' classificationSchemes='1' services='0' extrinsics='0'
associations='68'
serverStarted='2011-08-28T12:45:43.514-07:00' status='OK'/>
```

1.3 Operation

Operation

This section describes how to operate the Registry Service software. The following topics can be found in this section:

- Interface
- Publish Artifacts
- Query Artifacts
- Update Status
- Delete Artifacts
- Ping
- Report
- Controlled Access

Interface

The Registry Service provides a REST-based interface accessible via HTTP for interacting with the service. Details on the REST-based interface can be found in the API section. If viewing this document in PDF form, the API section is not available. The API documentation is available from any deployed instance of the Registry Service by accessing */registry/docs/* from the host application server. Because the REST-based interface operates over HTTP, there are several options for interacting with the Registry Service:

- Registry User Interface

  The registry-ui component offers a Graphical User Interface (GUI) for interacting with the service.

- Web Browser

  Any standard web browser (e.g., Firefox, Safari, Internet Explorer, etc.) will allow interaction with the query and retrieval interfaces of the service.

- cURL

  The cURL utility offers the most flexible means for interacting with the service. The utility comes installed on most UNIX-based platforms and is available for download for the Windows platform. The examples in the sections that follow utilize *cURL* to interact with the service. If *cURL* is not installed on the local machine but *Wget* is, see the Using Wget section for converting *cURL* commands to *Wget* commands. If viewing this document in PDF form, see the appendix for details.

The Registry Service also allows messaging (acceptance of and return of content descriptions) in the form of XML or JavaScript Object Notation (JSON). More on this in the examples below. Each PDS Node will have their own installation of the Registry Service with its own service endpoint. Because of this, the examples below use *http://localhost:8080* as the generic endpoint for the service.

Publish Artifacts

The Registry Service supports a wide range of artifacts for registration with the service. In ebXML terms, artifacts are referred to as Registry Objects. The following subsections provide examples for each of the supported Registry Object types.

Extrinsics

In PDS terms, an extrinsic can be a data product, document, element definition, mission description, schema, etc. Within the ebXML terminology this maps to an Extrinsic Object which is simply a way for a particular instantiation of a registry to extend its model. The following is an example of an extrinsic description in XML form:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<extrinsicObject xmlns="http://registry.pds.nasa.gov"
  guid="1234v1.0"
  lid="1234"
  name="Product 1234 v1"
  objectType="Product"
  description="This is a new version test product 1234 v1" >
  <slot name="first-name">
    <value>John</value>
  </slot>
  <slot name="last-name">
    <value>Doe</value>
  </slot>
  <slot name="phone">
    <value>(818)123-4567</value>
    <value>(818)765-4321</value>
  </slot>
</extrinsicObject>
```

The extrinsic description above is contained in the *new_product.xml* file, which can be found in the */examples* directory of the software distribution package. The following command registers this extrinsic with the service:

```
% curl -X POST -H "Content-type:application/xml" -v -d @new_product.xml \
http://localhost:8080/registry/extrinsics
```

A successful registration with the above command would produce the following output to standard out:

```
* About to connect() to localhost port 8080 (#0)
*   Trying ::1... Connection refused
*   Trying fe80::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST registry/extrinsics HTTP/1.1
> User-Agent: curl/7.16.3 (powerpc-apple-darwin9.0) ...
> Host: localhost:8080
> Accept: */*
> Content-type:application/xml
> Content-Length: 598
>
< HTTP/1.1 201 Created
< Server: Apache-Coyote/1.1
< Location: http://localhost:8080/registry/extrinsics/1234v1.0
< Content-Type: application/xml
< Content-Length: 0
< Date: Wed, 14 Apr 2010 20:33:32 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
urn:uuid:53451c5e-1809-4799-8dd8-060672f3e0e1
```

By inspecting the HTTP Response Location Header one can see the URL to the registered extrinsic. This header is a standard way for exchanging information about a newly created resource using HTTP. The last line of the response is the global unique identifier that the service assigned to the registered extrinsic. The following example details how to publish a new version of the above extrinsic to the service:

```
% curl -X POST -H "Content-type:application/xml" -v -d @new_product_v2.xml \
http://localhost:8080/registry/extrinsics/logicals/1234
```

The value of *1234* in the example above represents the logical identifier of the original published extrinsic, which must be specified in order for the service to recognize it as a new version. The following is an example of a extrinsic description in JSON form:

```
{"description":"This is a new version test product 5678 v1",
 "name":"Product 5678 v1",
 "objectType":"Product",
 "lid":"5678",
 "slot":[{"name":"last-name","value":["Doe"]},
         {"name":"phone","value":["(818)123-4567","(818)765-4321"]},
         {"name":"first-name","value":["Jane"]}]}
```

The extrinsic description above is contained in the *json_product.txt* file. The following command registers this extrinsic with the service:

```
% curl -X POST -H "Content-type:application/json" -v -d @json_product.txt \
http://localhost:8080/registry/extrinsics
```

Associations

In PDS terms, an association is a relationship between two registered artifacts. The following is an example of an association description in XML form:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<association xmlns="http://registry.pds.nasa.gov"
  sourceObject="1234v1.0"
  targetObject="1234v2.0"
  associationType="associatedTo"/>
```

The association description above is contained in the *new_association.xml* file. The following command registers this association with the service:

```
% curl -X POST -H "Content-type:application/xml" -v -d @new_association.xml \
http://localhost:8080/registry/associations
```

Services

In PDS terms, a service is an electronic resource available within the system. A service can be as simple as a web site or as intricate as the Registry Service that is described in this documentation. The following is an example of a service description in XML form:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<service xmlns="http://registry.pds.nasa.gov"
  name="PDS Service"
  description="This is a service to test adding a service description to the registry">
  <serviceBinding
    name="PDS Main Site"
    description="This is the PDS main web site"
    accessURI="http://pds.jpl.nasa.gov">
    <specificationLink
```

```
       name="HTTP Specification Link"
       description="This is a link to the HTTP specification."
       specificationObject="urn:uuid:HTTPSpecificationDocument">
         <usageDescription>
           Use a browser to access the PDS site. The acceptable browsers are
           listed in the usage parameters.
         </usageDescription>
         <usageParameter>Firefox</usageParameter>
         <usageParameter>Safari</usageParameter>
         <usageParameter>Internet Explorer</usageParameter>
         <usageParameter>Chrome</usageParameter>
     </specificationLink>
   </serviceBinding>
 </service>
```

The service description above is contained in the *new_service.xml* file. The following command registers this service with the service:

```
% curl -X POST -H "Content-type:application/xml" -v -d @new_service.xml \
http://localhost:8080/registry/services
```

Schemes and Nodes

In order for the above artifacts to be accepted for registration by the service, the service must be preloaded with the list of supported object types. This procedure is detailed in the Configuration portion of the Installation section. For more information on scheme and node registration see the Scheme and Node Registration section of the documentation. If viewing this document in PDF form, see the appendix for details.

Query Artifacts

Although the Registry Service does not offer an advanced query interface, it does offer interfaces for discovering and retrieving artifact descriptions. The URLs shown in the examples below will work in a web browser.

Extrinsics

The following command retrieves a paged list of registered extrinsics (products) from the service:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/extrinsics
```

The interface above accepts a number of parameters for filtering the return results. See the API section for a detailed list of the parameters. The following command retrieves the latest extrinsic with logical identifier *1234* from the service:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/extrinsics/logicals/1234
```

In order to retrieve the earliest extrinsic with logical identifier *1234*, append */earliest* to the URL in the example above. In order to retrieve the latest extrinsic with logical identifier *1234*, append */latest* to the URL in the example above. The following command retrieves the specific extrinsic with guid *1234*, but in JSON form:

```
% curl -X GET -H "Accept:application/json" -v \
http://localhost:8080/registry/extrinsics/1234v1.0
```

The example above will not work in a browser because it is not possible to set the HTTP Accept Header via a browser, but the following command will work in a browser by encoding the return type with a suffix in the URL:

```
% curl -X GET -v \
http://localhost:8080/registry/extrinsics/1234v1.0.json
```

Associations

The following command retrieves a paged list of registered associations from the service:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/associations
```

The interface above accepts a number of parameters for filtering the return results. See the API section for a detailed list of the parameters. In order to retrieve a specific association, append the global unique identifier

(/<guid>) for that association to the URL in the example above.

Services

The following command retrieves a paged list of registered services from the service:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/services
```

The interface above accepts a number of parameters for filtering the return results. See the API section for a detailed list of the parameters. In order to retrieve a specific service, append the global unique identifier (/<guid>) for that service to the URL in the example above.

Schemes and Nodes

The following command retrieves a paged list of registered schemes from the service:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/schemes
```

The interface above accepts a number of parameters for filtering the return results. See the API section for a detailed list of the parameters. In order to retrieve a specific scheme, append the global unique identifier (/<guid>) for that scheme to the URL in the example above.

The following command retrieves the list of nodes associated with a specific scheme:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/schemes/<guid>/nodes
```

In order to retrieve a specific node, append the global unique identifier (/<guid>) for that node to the URL in the example above.

Events

The service tracks auditable events for each registered artifact including submission, approval, deprecation, etc. The following command retrieves a paged list of events from the service:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/events
```

The interface above accepts a number of parameters for filtering the return results. See the API section for a detailed list of the parameters. In order to retrieve events for a specific object, append the global unique identifier (/<guid>) for the affected object to the URL in the example above.

Packages

When Harvest Tool registers a bundle or collection or products with the service, it precedes the registration with the registration of a package that all of the registered products will be associated with. The following command retrieves a paged list of packages from the service:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/packages
```

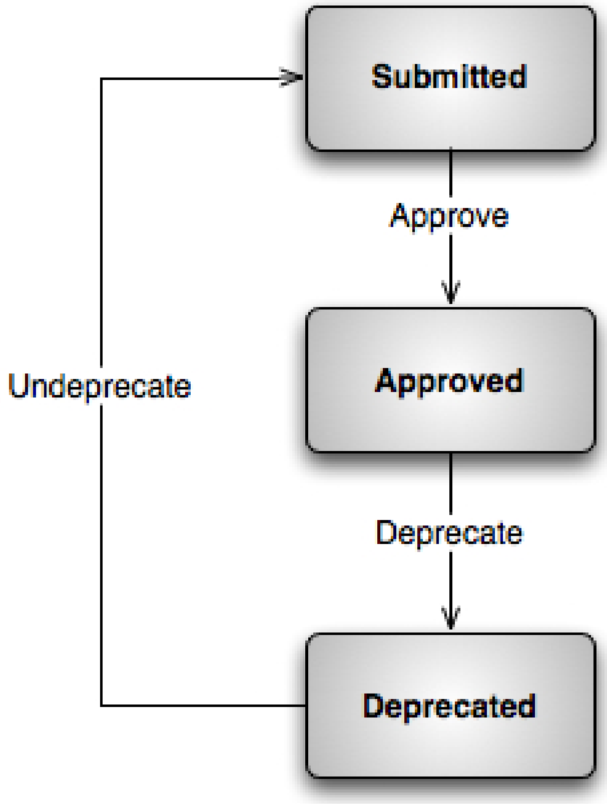
The interface above accepts a number of parameters for filtering the return results. See the API section for a detailed list of the parameters. In order to retrieve a specific package, append the global unique identifier (/<guid>) for that package to the URL in the example above.

Update Status

When extrinsics are successfully registered with the service they are given a status of *Submitted*. The status for a specific extrinsic can be modified with the following command:

```
% curl -X POST -H "Content-type:application/xml" -v \
http://localhost:8080/registry/extrinsics/<guid>/<action>
```

Valid values for <action> include *approve*, *deprecate* and *undeprecate*. The following diagram details the relationship of the status state with the above actions.



As mentioned above, Harvest Tool associates all registrations with a package. The status for the entire package, including its members, can be modified with the following command:

```
% curl -X POST -H "Content-type:application/xml" -v \
http://localhost:8080/registry/packages/<guid>/members/<action>
```

Delete Artifacts

The following command deletes the specific extrinsic from the service:

```
% curl -X DELETE -v \
http://localhost:8080/registry/extrinsics/<guid>
```

The same format applies to the other registry objects as well (e.g., associations, services, etc.). As mentioned above, Harvest Tool associates all registrations with a package. An entire package, including its members, can be deleted with the following command:

```
% curl -X DELETE -v \
http://localhost:8080/registry/packages/<guid>/members
```

The above command does not delete the package itself. The package can be deleted using the following:

```
% curl -X DELETE -v \
http://localhost:8080/registry/packages/<guid>
```

Ping

The following command checks to see if the registry service is up and running:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/
```

The above command will return the list of links to the service's endpoints and an HTTP status of 200. From a web browser, the command returns a welcome message. Make sure to include the trailing slash on the above command.

Report

The following command details the status of the service along with registered counts by Registry Object type:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/report
```

Controlled Access

A given instance of the Registry Service may be configured to control access to specific URLs utilizing the software of the Security Service. If this is the case, the *curl* application can be used to obtain an authentication cookie as follows:

```
% curl -X POST -H "Content-type:application/xml" -v -d @new_product.xml \
https://localhost:8443/registry/extrinsics -u username:password \
-k -c tomcat_cookie.txt
```

The cookie file *tomcat_cookie.txt* can then be passed to the next *curl* command:

```
% curl -X DELETE -H "Content-type:application/xml" -v -d @new_product_v2.xml \
https://localhost:8443/registry/extrinsics/1234v1.0 -k -b tomcat_cookie.txt
```

The example above also applies to a *DELETE* request.

1.4 Appendix A - Using Wget

Using Wget

Although cURL is the command-line application of choice for interacting with the Registry Service, some machines may only have Wget installed or some users may prefer *Wget* over *cURL*. The functionality of each application is essentially the same. This section describes the differences between the two applications.

The following command provides an example of using *cURL* to register a product with the Registry Service:

```
% curl -X POST -H "Content-type:application/xml" -v -d @new_product.xml \
http://localhost:8080/registry/extrinsics
```

The same action can be achieved using *Wget* with the following command:

```
% wget --post-file new_product.xml --header "Content-type:application/xml" -v \
http://localhost:8080/registry/extrinsics
```

The following command provides an example of using *cURL* to retrieve content from the Registry Service:

```
% curl -X GET -H "Accept:application/xml" -v \
http://localhost:8080/registry/extrinsics
```

The same action can be achieved using *Wget* with the following command:

```
% wget --header "Accept:application/xml" -v -O - \
http://localhost:8080/registry/extrinsics
```

The *Wget* application works fine for actions requiring GET or POST requests but does not offer an equivalent function for DELETE requests.

1.5 Appendix B - Scheme and Node Registration

Scheme and Node Registration

This section describes classification scheme and classification node registration in more detail. At this point in time, only one classification scheme is supported by the Registry Service and that is the ObjectType scheme. This scheme is utilized by the service for determining the allowable object types (artifacts) for registration. These object types are represented by the classification nodes, which are children of the ObjectType classification scheme.

The standard scheme and associated nodes are registered during the installation of the service. That procedure is detailed in the Configuration portion of the Installation section. The rest of this section gives a little insight into the content of the scheme and node descriptions and how they are registered with the service. The following is an example of a scheme description in XML form:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<classificationScheme xmlns="http://registry.pds.nasa.gov"
  guid="urn:registry:classificationScheme:ObjectTypeScheme:Test"
  name="TestObjectType"
  description="This is the canonical object type classification
  that is one of the core registry objects"
  isInternal="true"
  nodeType="UniqueCode"/>
```

The scheme description above is contained in the *new_scheme.xml* file, which can be found in the */examples* directory of the software distribution package. The following command registers this scheme with the service:

```
% curl -X POST -H "Content-type:application/xml" -v -d @new_scheme.xml \
http://localhost:8080/registry/schemes
```

The following is an example of a node description in XML form:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<classificationNode xmlns="http://registry.pds.nasa.gov"
  guid="urn:nasa:pds:profile:regrep:ObjectType:ProductTest"
  lid="urn:nasa:pds:profile:regrep:ObjectType:ProductTest"
  name="Product Test Node"
  description="This is the classification node for testing."
  parent="urn:registry:classificationScheme:ObjectTypeScheme:Test"
  code="ProductTest"/>
```

The node description above is contained in the *new_node_product.xml* file. The following command registers this node with the service:

```
% curl -X POST -H "Content-type:application/xml" -v -d @new_node_product.xml \
http://localhost:8080/registry/\
schemes/urn:registry:classificationScheme:ObjectTypeScheme:Test/nodes
```