International Planetary Data Alliance (IPDA)

Registry Service Protocol

Version 1.0

Editor:
S. Hardman

Authors:
S. Hardman, P. Ramirez

Abstract

This document describes the protocol for the REST-based API of the Registry Service.

Status of this document

This document was generated by the Registry Development and Coordination Project (2011-2012).

Acknowledgments

This document is based on the PDS Registry Service Operation Guide.

Change Log

Revision	Date	Description
1.0	July 2012	Initial release.

Contents

Abstract 1
Status of this document 1
Acknowledgments 1
Change Log 2
Contents 3
1 Executive Summary 4
1.1 Purpose 4
1.2 Applicable Documents 4
2 Syntax 5
3 Registry Endpoints 6
3.1 Publish Artifacts 6
3.2 Query Artifacts 9
3.3 Update Status 11
3.4 Delete Artifacts 12
3.5 Ping 13
3.6 Report 13
Appendix: Acronyms 14

1 Executive Summary

This document captures the protocol for the Registry Service. As described in the Registry Service Design Specification [1], this protocol is implemented as a REST-based interface for the Registry Service.

1.1 Purpose

This document is intended for the reviewer of the protocol as well as the developer of the protocol (for implementation) and tester of the protocol (for quality assurance).

1.2 Applicable Documents

1) IPDA Registry Service Design Specification, Version 1.0, July 2012.

2) Registry Service API, June 2012.

2 Syntax

The interface is a REST-based interface over the HTTP protocol. The HTTP protocol is pretty basic in nature with respect to passing parameters. The following details the syntax for passing parameters:

```
http://<host>/<path>[?<parameter>=<value>[&
<parameter>=<value>…]]
```

As shown above the "&" symbol separates multiple clauses (parameter/value pairs) and may be interpreted as AND or OR depending on the underlying implementation. For this protocol, the underlying implementation should interpret it as an AND meaning that each clause must evaluate to true in order for a specific object to be included in the result set.

A request to the service is transmitted as an HTTP GET, POST or DELETE request. Parameter names in a request are case sensitive, meaning they should be represented in the case for which they are defined in the Registry Service API [2] document. Valid parameters are specific to a given endpoint. Values are also case sensitive.

The base URL, depicted as *http://<host>/<path>* in the example above, will vary based on the underlying implementation and its deployment. The main deployment of the Registry Service for IPDA, hosting the catalog-level metadata, will have the following as its base URL:

```
http://planetarydata.org/registry
```

For the examples that follow, the base URL is represented as <*base-url*>.

3 Registry Endpoints

This section provides an overview of the endpoints supported by the Registry Service. Details on the endpoints including supported parameters can be found in the Registry Service API [2] document. This document is available online for a given instance of the service at the following endpoint:

```
<base-url>/docs
```

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

3.1 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.

3.1.1 Extrinsics

In IPDA 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://planetarydata.org"
  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 <base-url>/extrinsics
```

By inspecting the HTTP Response Location Header returned from the above command, 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 <base-url>/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 <base-url>/extrinsics
```

3.1.2 Associations

In IPDA 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://planetarydata.org"
```

```
  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 <base-url>/associations
```

3.1.3 Services

In IPDA 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://planetarydata.org"
  name="PDS Service"
  description="This is a service to test adding a service
description to the registry">
  <serviceBinding
    name="IPDA 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 <base-url>/services
```

3.1.4 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 performed during installation.

3.2 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.

3.2.1 Extrinsics

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

```
% curl -X GET -H "Accept:application/xml" -v <base-
url>/extrinsics
```

The interface above accepts a number of parameters for filtering the return results. See the API [2] document 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 <base-
url>/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 <base-
url>/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 <base-url>/extrinsics/1234v1.0.json
```

3.2.2 Associations

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

```
% curl -X GET -H "Accept:application/xml" -v <base-
url>/associations
```

The interface above accepts a number of parameters for filtering the return results. See the API [2] document 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.

3.2.3 Services

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

```
% curl -X GET -H "Accept:application/xml" -v <base-
url>/services
```

The interface above accepts a number of parameters for filtering the return results. See the API [2] document 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.

3.2.4 Schemes and Nodes

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

```
% curl -X GET -H "Accept:application/xml" -v <base-
url>/schemes
```

The interface above accepts a number of parameters for filtering the return results. See the API [2] document 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 <base-
url>/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.

3.2.5 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 <base-
url>/events
```

The interface above accepts a number of parameters for filtering the return results. See the API [2] document 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.

3.2.6 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 <base-
url>/packages
```

The interface above accepts a number of parameters for filtering the return results. See the API [2} document 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.

3.3 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 <base-
url>/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.

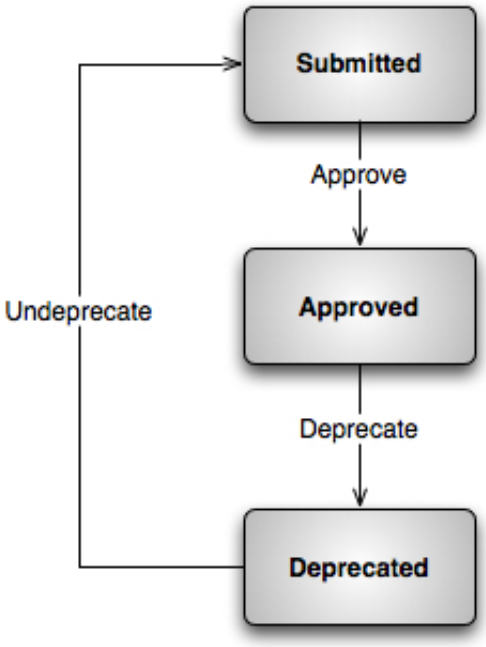

Figure 1: Registry Service Status States

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 <base-
url>/packages/<guid>/members/<action>
```

3.4 Delete Artifacts

The following command deletes the specific extrinsic from the service:

```
% curl -X DELETE -v <base-url>/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 <base-url>/packages/<guid>/members
```

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

```
% curl -X DELETE -v <base-url>/packages/<guid>
```

3.5 Ping

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

```
% curl -X GET -H "Accept:application/xml" -v <base-url>
```

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.

3.6 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 <base-
url>/report
```

Appendix: Acronyms

The following acronyms pertain to this document:

API – Application Programming Interface
ebXML – Electronic Business using XML
HTTP – HyperText Transfer Protocol
IPDA – International Planetary Data Alliance
PDS – Planetary Data System
REST – Representational State Transfer
XML – Extensible Markup Language