SCIENCE DATA INFRASTRUCTURE FOR PRESERVATION - EARTH SCIENCE

Registry Framework Adapters
Implementation Guide

Current Version: Framework Version 3.1.0
Last Updated: 2 December 2014
Original Author: Simon Berriman

Table of Contents

Document Control 2
About this Document 2
- Purpose and Scope 2
- Who should read this document 2
- Prerequisite Reading 2

Introduction 3
Registry Adapters 3
Registry Base Classes within the Framework 3
- AbstractRegistry 3
- AbstractHTTPRegistry 3
- BaseSimpleRegistryImpl 4
- BaseACSRegistryImpl 4

Discovery via Java SPI 5
Deploying a New Adapter 6
Generic Adapters for Private SCIDIP-ES Registry Types 6

Document Control

Date	Version	Description	Author
31/07/2014	Framework Version 3.1.0	Initial version	Simon Berriman
05/08/2014	Framework Version 3.1.0	Updated for inclusion of generic adapters	Simon Berriman

About this Document

Purpose and Scope

This document provides details of the means by which the SCIDIP-ES Framework library discovers and interacts with Registries, and how to implement the necessary extra code required for using a non-public Registry.

Who should read this document

Developers who wish to understand and use the SCIDIP-ES Framework for communicating with SCIDIP-ES RepInfo Registry Service/s, and System Implementors creating a local environment including a non-public Registry.

Prerequisite Reading

It is assumed the reader has previously read the document 'Registry Framework – Developer's Overview'.

Introduction

As explained in 'Registry Framework – Developer's Overview', the SCIDIP-ES Framework is a library of common software designed to be built into other applications. Its purpose is to abstract interactions with one or more Registries within the SCIDIP-ES e-infrastructure on behalf of that application.

The Framework does not inherently know of any specific Registry. Registries are defined only at an abstract level within the model. The following sections detail how the Framework finds and uses a specific implementation of a Registry.

Registry Adapters

The implementation of the code which represents a specific deployment of a Registry is referred to as a 'Registry Adapter', and is a separate Java class, independent of the Framework. As a general principle, it is the Registry implementer's responsibility to provide the adapter class for any given Registry instance. Although within the SCIDIP-ES project both Registry designs use XML over HTTP, there is no specific limit on how a Registry could be designed and deployed. The adapter class insulates the Framework and the applications which use it from any given Registry's public API.

Registry Adapters are located at runtime using Java's Service Provider Interface (SPI), and so once an adapter has been correctly constructed and packaged as a JAR, it can be deployed in such a way so to be discovered without any further configuration. See the '' section for details.

The Framework contains base classes for the common or abstract operations carried out by all Registries of certain implementation types, right up to base classes for the two specific Registry technology types implemented within the SCIDIP-ES project; namely the ACS Registry and the APA Web Registry. If the implemented Registry is of one of these two types, then minimal code is required to construct a suitable adapter class. If an entirely new interpretation of the SCIDIP-ES Registry is to integrated, then a more involved adapter class will be required.

There also exists generic adapters for both of the SCIDIP-ES Registry types. These are designed for local or private registry deployments only, and allow a registry to be deployed without the need to write any code. These are explained in the section ''.

Registry Base Classes within the Framework

These classes are all within the '`eu.scidipes.common.framework.registry`' package in the Framework. Any one of them could be used as the base class for an adapter, however more work would be required with the 'Abstract...' classes than with the 'Base...' classes.

AbstractRegistry

This is little more than a skeleton implementation of the model 'Registry' interface. This should only be used as the base class for a Registry Adapter if the Registry in question is an entirely new implementation which does not even use HTTP as its API communication protocol.

AbstractHTTPRegistry

This extends from AbstractRegistry and implements the use of HTTP as the API communication protocol for the Registry. It breaks down the separate tasks required to be performed by function. This should only be used as the base class for a Registry Adapter if the Registry in question is not a compatible implementation of one of the two types produced within the SCIDIP-ES project.

BaseSimpleRegistryImpl

This extends from the AbstractHTTPRegistry, and provides all the task functions required by that class within the context of the API expected by a deployment of the APA Web Registry. This should be the base class for a Registry Adapter if the Registry in question is a deployment of this Registry type.

To implement an adapter using this as a base class, only two methods are required to be implemented to return the String values for a unique name and also for the base HTTP URL on which the Registy is deployed, as shown in the following example code:

```
package my.registry.adapter;

import info.digitalpreserve.exceptions.RIException;
import eu.scidipes.common.framework.registry.BaseSimpleRegistryImpl;

/**
 * Adapter for an APA (aka 'Web') Registry
 */
public class MyAPARegistry extends BaseSimpleRegistryImpl {

	/**
	 * Simple constructor for a APA Registry
	 *
	 * @throws RIException
	 *             if the base URL is illegal
	 */
	public MyAPARegistry() throws RIException {
	}

	/**
	 * Returns a unique string to identify this Registry.
	 * This should be short, but human readable.
	 */
	@Override
	public String getLocationUID() {
		return "MyAPAReg";
	}

	/**
	 * Returns the base HTTP URL with which this Registry can be located
	 * by the machine running application this adapter is loaded in to.
	 */
	@Override
	protected String getBaseURLPath() {
		return "http://apademo.scidip-es.eu";
	}
}
```

BaseACSRegistryImpl

This extends from the AbstractHTTPRegistry, and provides all the task functions required by that class within the context of the API expected by a deployment of the ACS Registry. This should be the base class for a Registry Adapter if the Registry in question is a deployment of this Registry type.

To implement an adapter using this as a base class, only two methods are required to be implemented to return the String values for a unique name and also for the base HTTP URL on which the Registy is deployed, as shown in the following example code:

```
package my.registry.adapter;

import info.digitalpreserve.exceptions.RIException;
import eu.scidipes.common.framework.registry.BaseACSRegistryImpl;

/**
 * Adapter for ACS Registry implementation
 */
public class MyACSRegistry extends BaseACSRegistryImpl {

	/**
	 * Simple constructor for a ACS Registry
	 *
	 * @throws RIException
	 *             if the base URL is illegal
	 */
	public MyACSRegistry() throws RIException {
	}

	/**
	 * Returns a unique string to identify this Registry.
	 * This should be short, but human readable.
	 */
	@Override
	public String getLocationUID() {
		return "MyACSReg";
	}

	/**
	 * Returns the base HTTP URL with which this Registry can be located
	 * by the machine running application this adapter is loaded in to.
	 */
	@Override
	protected String getBaseURLPath() {
		return "http://acsdemo.scidip-es.eu";
	}
}
```

Examples of both of the above can be easily exported from the 'Digital Preservation Services' project on SourceForge, from the path '`/SCIDIP-ES/software/common/registry-adapters`'; see the corresponding public view of the repository tree here:

https://sourceforge.net/p/digitalpreserve/code/HEAD/tree/SCIDIP-ES/software/common/registry-adapters/

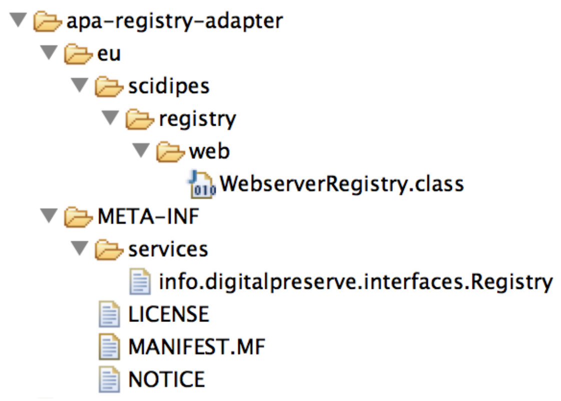
Discovery via Java SPI

In order for the Framework to automatically discover all adapters within its JVM, the adapter class needs to be made known. As previously mentioned, the Framework uses Java's built-in Service Provider Interface for class discovery. In order for a class which implements a specific interface to be found and initialised, it needs to be accompanied by a small, very specific text file.

The file has to be found on the classpath in the folder '`/META-INF/services/`' and must be named with the fully qualified classname of the interface being discovered, which in this case is '`info.digitalpreserve.interfaces.Registry`'. The contents of the file must be the fully qualified class of the new adapter class. So, in the case of the APA example class from the previous section, this would simply be the following text:

```
my.registry.adapter.MyAPARegistry
```

Deploying a New Adapter



```
apa-registry-adapter
    eu
        scidipes
            registry
                web
                    WebserverRegistry.class
    META-INF
        services
            info.digitalpreserve.interfaces.Registry
        LICENSE
        MANIFEST.MF
        NOTICE
```

With the adapter class and SPI text file correctly created and compiled, they need then to be packaged as a JAR for easy deployment. For reference, the core APA adapter's JAR has the following structure:

If the Registry is to be a public facing Registry, then the resultant JAR should be sent to the APA for inclusion on the public Switchboard. After the termination of the SCIDIP-ES project this JAR will need to be accompanied by any required forms and/or payment – please contact the APA for details.

If the Registry is private, or is for testing purposes only, the adapter can just be dropped in the to local classpath of the application which is using the Framework. For instance, for a service application running in Tomcat, this would mean simply copying the JAR in the web application's '`WEB-INF/lib`' folder and restarting the webapp.

Generic Adapters for Private SCIDIP-ES Registry Types

Within the 'Digital Preservation Services' project on SourceForge there exist two 'generic' registry adapters, one each for the ACS and APA Web registry types within the SCIDIP-ES project. These adapters can be found in the path '`/SCIDIP-ES/software/common/registry-adapters`' alongside the concrete implementations talked about previously.

These adapters are provided for a simpler means of deploying a Registry, but do come with some restrictions. They can only be deployed locally, in that they can not be used on the APA switchboard for public registries, and also there can only be a maximum of one of each type within any given Framework-based application's classpath. This is because each adapter looks for a specific properties file in the classpath which isn't included with the adapter, and must be created and supplied. For instance, in the case of the APA Web registry, the file it's generic adapter will look for is '`generic-web-registry.properties`'. The files must be placed in the root of the classpath and name the two String properties required, for example:

```
locationUID = MyAPAReg
baseURLPath = http://apademo.scidip-es.eu
```

The advantage to these adapters is that once compiled, changing the name and/or the base URL for an given registry is just a matter of editing the properties file and restarting the application.

For more insight into all the registry adapters and classes talked about in this document, it is recommended that you explore the source code on the SourceForge repository.