Template version: proc-RFC_template-1_01_011016.dot

RFC 51 - XML Parser Service

Confidential, Draft
<rfc0051-xmlparserservice>

11 Pages

Abstract

This RFC defines a method to convert any JAXP compliant XML Parser into an OSGi Service. As an OSGi service, multiple XML Parsers can co-exist within the framework and using bundles can choose among them at runtime based on their characteristics.

0 Document Information

0.1 Table of Contents

0.2 Status

This document specifies an XML Parser Service for the Open Services Gateway Initiative, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

0.3 Acknowledgement

This document is made possible by the developers of the javax.xml.parsers API.

0.4 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

```
Source code is shown in this typeface.
```

All Page Within This Box

0.5 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	Apr 16, 2002	Initial Draft Pam Tobias, IBM, pam_tobias@us.ibm.com
Review Draft 1	May 16, 2002	Incorporated comments from Kevin Riff of Espial. Changed package name to org.osgi.util.xml.XMLParserActivator. It is now a utility class that can be used as is, subclassed, or not used at all. Pam Tobias, IBM, pam_tobias@us.ibm.com

1 Introduction

The Extensible Markup Language (XML) has become a popular method of describing data. As more bundles use XML to describe their data, a common XML Parser becomes more attractive in an embedded environment in order to save space. However, not all XML Parsers are equivalent in function, and not all bundles have the same requirements for an XML parser. The desired solution is one where multiple XML parsers can co-exist within an OSGi framework, and bundles can choose which XML parser to use at runtime, based on which parsers are available and the functionality provided by those parsers.

2 Motivation and Rationale

In the world of XML Parsers, there are primarily two types: SAX & DOM. A given XML Parser implementation may support either or both of these parser types by implementing the org.w3c.dom and/or org.xml.sax packages.

All Page Within This Box

In addition, parsers have characteristics such as whether they are validating or non-validating parsers and whether or not they are namespace aware. An application which uses a specific XML Parser must choose which parser to use, and code to that specific parser. If the parser has implemented JAXP, the application developer has more flexibility. JAXP is a package (javax.xml.parsers) which provides an abstraction layer between an application and a specific XML Parser implementation. Using applications can choose to use any JAXP compliant parser without changing any code simply by changing the contents of a properties file which specifies the parser class name before launching.

But in an OSGi environment, even more flexibility is desired. It would be desirable for bundles to be able to choose which parser to use at runtime and to choose based on the characteristics of that parser. This RFC defines a BundleActivator class which can be added to any JAXP compliant XML Parser. This class allows a parser to register itself as an OSGi service along with service properties that define its characteristics. With this added flexibility, bundles can use the OSGi service registry to locate XML Parsers that match a given set of requirements.

3 Technical Discussion

JAXP allows for the definition of factories to provide SAX & DOM Parsers through SAXParserFactories and DocumentBuilderFactories. But the limitation of using just the existing JAXP functionality is that only one of each type of parser factory can be registered. The XML Parser Service provides a way for multiple SAXParserFactories and DocumentBuilderFactories to be registered. It registers these factories in the OSGi Service Registry with service properties which describe whether the parser is validating or non-validating and whether it is namespace aware or not, and it can be extended by the parser to register any additional features provided by the parser. With this functionality, bundles can query the OSGi service registry for parsers supporting the specific functionality that they require.

If an XML Parser supports JAXP, then it can be converted to an OSGi aware bundle by adding a BundleActivator class which registers an XML Parser Service. The org.osgi.util.xml.XMLParserActivator class provides this function and can be compiled into any XML Parser, or it can be extended and customized if desired.

The details for this class are described in the javadoc below.

3.1 org.osgi.util.xml
Class XMLParserActivator

`org.osgi.util.xml.XMLParserActivator`

public class **XMLParserActivator**

XMLParserActivator is a BundleActivator class that allows any JAXP compliant XML Parser to register itself as an OSGi service. Multiple JAXP compliant parsers can concurrently register by using this BundleActivator class. Bundles who wish to use an XML parser can then use the framework's service registry to locate available XML Parsers with the desired characteristics such as validating and namespace-aware.

All Page Within This Box

The services that this bundle activator enables a bundle to provide are:

- `javax.xml.parsers.SAXParserFactory` (`SAXFACTORYNAME`)
- `javax.xml.parsers.DocumentBuilderFactory`(`DOMFACTORYNAME`)

XMLParserActivator assumes that it can find the class file names of the factory classes in the following files:

- `/META-INF/services/javax.xml.parsers.SAXParserFactory` is a file contained in a jar available to the runtime which contains the implementation class name of the SAXParserFactory
- `/META-INF/services/javax.xml.parsers.DocumentBuilderFactory` is a file contained in a jar available to the runtime which contains the implementation class name of the DocumentBuilderFactory

If either of the files does not exist, XMLParserActivator assumes that the parser does not support that parser type.

XMLParserActivator attempts to instantiate both the SAXParserFactory and the DocumentBuilderFactory. It registers each factory with the framework along with service properties:

- `PARSER_VALIDATING` - indicates if this factory supports validating parsers. It's value is a Boolean.
- `PARSER_NAMESPACEAWARE` - indicates if this factory supports namespace aware parsers It's value is a Boolean.

Individual parser implementations may have additional features, properties, or attributes which could be used as filters. These can be added by extending this class and overriding the `setSAXProperties` and `setDOMProperties` methods.

To incorporate this bundle activator into your XML Parser Bundle, do the following:

- If SAX parsing is supported, create a `/META-INF/services/javax.xml.parsers.SAXParserFactory` file containing the class name of your SAXParserFactory class.
- If DOM parsing is supported, Create a `/META-INF/services/javax.xml.parsers.DocumentBuilderFactory` file containing the class name of your DocumentBuilderFactory class.
- Create an OSGi manifest file which exports `org.w3c.dom`, `org.xml.sax` and `javax.xml.parsers`
- If your parser supports attributes, properties, or features that you would like to externalize so that other OSGi bundles can use them as search filters, extend this class and override `setSAXProperties` and `setDOMProperties`. It is a requirement that you call the original methods via super() to initialize the XML Parser Service properties properly.
- Compile this class into your package and install your new XML Parser Service bundle

Following is a simple example of how to locate and use a SAX Parser from another bundle

```
org.osgi.framework.ServiceReference sref =
```

All Page Within This Box

```
getServiceReference("javax.xml.parsers.SAXParserFactory");
 javax.xml.parsers.SAXParserFactory factory = bundlecontext.getService(sref);
 javax.xml.parsers.SAXParser parser = factory.newSAXParser();
 parser.parse(filename, handler);  // handler is where the input processing takes
place
```

In a complex environment, multiple parsers may have registered the services `javax.xml.parsers.SAXParserFactory` or `javax.xml.parsers.DocumentBuilderFactory`. In this case, the calling bundle may wish to filter which services are returned by the framework. For example, to locate a validating SAX parser:

```
 org.osg.framework.ServiceReference [] srefs =

getServiceReferences("javax.xml.parsers.SAXParserFactory",
                                                    "(parser.validating=true)");
 if (srefs!=null) {
    javax.xml.parsers.SAXParserFactory factory = bundlecontext.getService(srefs[0]);
    javax.xml.parsers.SAXParser parser = factory.newSAXParser();
    parser.parse(filename, handler);  // handler is where the input processing takes
place
 }
```

Field Summary

static java.lang.String	**DOMCLASSFILE** Fully qualified path name of DOM Parser Factory Class Name file
static java.lang.String	**DOMFACTORYNAME** Filename containing the DOM Parser Factory Class name Also used as the basis for the service_pid registration property.
static java.lang.String	**PARSER_NAMESPACEAWARE** Service property specifiying if factory is configured to support namespace aware parsers.
static java.lang.String	**PARSER_VALIDATING** Service property specifiying if factory is configured to support validating parsers.
static java.lang.String	**SAXCLASSFILE** Fully qualified path name of SAX Parser Factory Class Name file
static java.lang.String	**SAXFACTORYNAME** Filename containing the SAX Parser Factory Class name.

Constructor Summary

XMLParserActivator()

All Page Within This Box

Method Summary

void	**setDOMProperties**(javax.xml.parsers.DocumentBuilderFactory factory, java.util.Hashtable props) Set the DOM Parser Service Properties.
void	**setSAXProperties**(javax.xml.parsers.SAXParserFactory factory, java.util.Hashtable props) Set the SAX Parser Service Properties.
void	**start**(org.osgi.framework.BundleContext context) Called when this bundle is started so the Framework can perform the bundle-specific activities necessary to start this bundle.
void	**stop**(org.osgi.framework.BundleContext context) This method has nothing to do as all open service registrations will automatically get unregistered when the bundle stops.

Field Detail

3.1.1 SAXFACTORYNAME

public static final java.lang.String **SAXFACTORYNAME**

Filename containing the SAX Parser Factory Class name. Also used as the basis for the service_pid registration property.

3.1.2 DOMFACTORYNAME

public static final java.lang.String **DOMFACTORYNAME**

Filename containing the DOM Parser Factory Class name Also used as the basis for the service_pid registration property.

3.1.3 SAXCLASSFILE

public static final java.lang.String **SAXCLASSFILE**

Fully qualified path name of SAX Parser Factory Class Name file

---

3.1.4 DOMCLASSFILE

public static final java.lang.String **DOMCLASSFILE**

Fully qualified path name of DOM Parser Factory Class Name file

---

3.1.5 PARSER_VALIDATING

public static final java.lang.String **PARSER_VALIDATING**

Service property specifiying if factory is configured to support validating parsers. The value is a Boolean.

All Page Within This Box

3.1.6 PARSER_NAMESPACEAWARE

`public static final java.lang.String` **`PARSER_NAMESPACEAWARE`**

Service property specifiying if factory is configured to support namespace aware parsers. The value is a Boolean.

Constructor Detail

3.1.7 XMLParserActivator

`public` **`XMLParserActivator`**`()`

Method Detail

3.1.8 start

```
public void start(org.osgi.framework.BundleContext context)
          throws java.lang.Exception
```

Called when this bundle is started so the Framework can perform the bundle-specific activities necessary to start this bundle. This method can be used to register services or to allocate any resources that this bundle needs.

This method must complete and return to its caller in a timely manner.

This method attempts to register a SAX and DOM parser with the OSGI service registry.

Parameters:
`context` - The execution context of the bundle being started.
Throws:
`java.lang.Exception` - If this method throws an exception, this bundle is marked as stopped and the Framework will remove this bundle's listeners, unregister all services registered by this bundle, and release all services used by this bundle.
See Also:
`Bundle#start`

3.1.9 stop

```
public void stop(org.osgi.framework.BundleContext context)
          throws java.lang.Exception
```

This method has nothing to do as all open service registrations will automatically get unregistered when the bundle stops.

Parameters:
`context` - The execution context of the bundle being stopped.
Throws:
`java.lang.Exception` - If this method throws an exception, the bundle is still marked as stopped, and the Framework will remove the bundle's listeners, unregister all services registered by the bundle, and release all services used by the bundle.
See Also:
`Bundle#stop`

All Page Within This Box

3.1.10 setSAXProperties

```
public void setSAXProperties(javax.xml.parsers.SAXParserFactory factory,
                             java.util.Hashtable props)
```

Set the SAX Parser Service Properties. By default, the following properties are set:

- SERVICE_DESCRIPTION
- SERVICE_PID
- PARSER_VALIDATING - instantiates a parser and queries it to find out whether it is validating or not
- PARSER_NAMESPACEAWARE - instantiates a parser and queries it to find out whether it is namespace aware or not

This method can be subclassed to add additional SAX2 features and properties, but this method should be invoked via super() prior to adding your own properties.

Parameters:
`factory` - - the SAXParserFactory object
`props` - - Hashtable of service properties.

3.1.11 setDOMProperties

```
public void setDOMProperties(javax.xml.parsers.DocumentBuilderFactory factory,
                             java.util.Hashtable props)
```

Set the DOM Parser Service Properties. By default, the following properties are set:

- SERVICE_DESCRIPTION
- SERVICE_PID
- PARSER_VALIDATING
- PARSER_NAMESPACEAWARE

This method can be subclassed to add additional DOM Level 2 attributes, but this method should be invoked via super() prior to adding your own properties.

Parameters:
`factory` - - the DocumentBuilderFactory object
`props` - - Hashtable of service properties.

All Page Within This Box

4 Security Considerations

Bundles which include XMLParserActivator must have permission to register the `javax.xml.parsers.SAXParserFactory` and `javax.xml.parsers.DocumentBuilderFactory` services. They must also have java.io.FilePermission to read the `/META-INF/services/javax.xml.parsers.SAXParserFactory` and `/META-INF/services/javax.xml.parsers.DocumentBuilderFactory` files.

5 Document Support

5.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Org.xml.sax package

[3]. Org.w3c.dom package

[4]. Javax.xml.parsers package

5.2 Author's Address

Name	Pam Tobias
Company	IBM
Address	11400 Burnet Road, Austin TX 78758 USA
Voice	512-346-1483
e-mail	Pam_tobias@us.ibm.com

5.3 Acronyms and Abbreviations

- DOM – Document Object Model
- JAXP – Java API for XML Parsers

All Page Within This Box

- SAX – The Simple API for XML
- XML – Extensible Markup Language

5.4 End of Document