RFC 101 Extension Bundles

Final

11 Pages

Abstract

This RFC describes a mechanism to install extension bundles into the OSGi framework.

0 Document Information

0.1 Table of Contents

0.2 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.`

0.3 Revision History

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

Revision	Date	Comments
Initial	10 04 2004	Initial draft Thomas Watson, IBM tjwatson@us.ibm.com
Draft 2	12 02 2004	Significant rewrite to include comments from Nice F2F meeting. Move to use bundle fragment concepts for extension bundles. Thomas Watson, IBM tjwatson@us.ibm.com
Draft 3	01 06 2005	Minor updates according to comments from Boca Raton F2F meeting. • Use system.bundle as the symbolic name of the system bundle • Use Fragment-Host directive “extension” to specify a bootclasspath vs. framework extension • Export-Package applies to both framework and bootclasspath extensions • Moved Security section to 6
Draft 4	03 03 2005	• Updates to make extension bundles act like normal fragment bundles as much as possible o Removed restriction on getEntry methods o Removed restriction on Bundle-Classpath o All other headers are treated just like fragment bundles (except Import-Package, Require-Bundle and Bundle-NativeCode) o Allow frameworks to support attaching extension bundles to an already active framework. Just like attaching fragments to already resolved host bundles. • Updated extension bundle lifecycle chart. • Update security section to include checks for AdminPermission[<bundle>, EXTENSIONLIFECYCLE]

Revision	Date	Comments
Final Review Draft	20 Mar 2005	Final comments from Munich CPEG meeting. • Define properties to indicate if the framework supports extension bundles. BJ Hargrave, hargrave@us.ibm.com
Final	27 May 2005	No Changes BJ Hargrave, hargrave@us.ibm.com

1 Introduction

OSGi provides a framework in which Java components known as *bundles* interact. Bundles carry code, resources, and manifest definitions known as *metadata* in JAR files. Bundles may be installed, updated and uninstalled at any time while the framework is active. The bundle lifecycle allows administrative actions to be performed within the framework dynamically without the need of native code to install new features or update existing ones.

There are parts of the framework execution environment which are static and cannot be administered using the bundle lifecycle. The VM configuration (host VM) used to run a framework is usually static. In other words the classes and resources available on the boot classpath of the host VM cannot be affected by the framework. By the time the first line of implementation code is run in the framework the VM has already statically defined the boot classpath and it can never be changed while the framework is active. The framework implementation classes and resources are usually static also. There is currently no defined way the framework can affect the framework implementation classes and resources by installing new bundles or updating existing ones.

The bundle lifecycle is a powerful concept that we would like to apply to other areas within a framework execution environment. This RFC describes a mechanism to install extension bundles into the framework. Extension bundles are special bundles that contain code that extend the host VM's boot classpath or the framework implementation.

2 Application Domain

The original intention for the framework was to provide a general-purpose, secure, and managed Java framework that supports the deployment of extensible and downloadable service applications known as bundles. We are now seeing the need to deploy and manage the framework's execution environment. Additional features are needed to enable the framework to deploy and manage the configuration of the framework execution environment.

3 Problem Description

The framework is run within a static execution environment. An execution environment is usually comprised of two statically configured components; a configured host VM which is used to run the framework within and a configured set of classes and resources that are used to implement the framework itself. There is currently no defined API that allows a framework to deploy or manage the execution environment.

4 Requirements

There is a need to leverage the bundle lifecycle architecture to deploy and manage a framework execution environment. This would allow an administrator to manage the execution environment of a framework just like any other bundle installed within the framework. A new type of bundle is needed to deliver extensions to the execution environment. This new type of *extension* bundle would be treated differently by the framework. There is a need for two different kinds of extension bundles; a boot classpath extension bundle and a framework extension bundle.

4.1 Boot ClassPath Extensions

A J2ME execution environment is generally a subset of the J2SE execution environment (javax.microedition is a bad exception). There are a few packages in the java.* namespace that are not included in J2ME from J2SE (e.g. java.sql). The Java specification states that all packages in the java namespace must be loaded from the boot classloader. This prevents any bundle from delivering the java.sql package to the framework. The VM will not allow classes from the java.sql package within the bundle to be loaded because the bundle's classloader is not the boot classloader.

There is a requirement to allow for boot classpath extension bundles to configure a host VM's boot classpath. A boot classpath extension bundle must be treated differently from "normal" bundles. The framework must use boot classpath extension bundles to configure the host VM's boot classpath.

4.2 Framework Extensions

Framework implementation classes and resources are usually static and cannot be extended. There is a requirement to deliver optional parts of the framework as framework extension bundles. One example is the optional framework support for URL stream handler services. This is a good example of an option framework feature that is self contained and could be easily delivered as a separate component. One reason it cannot be delivered as a bundle is because it must register implementation objects with the java.net.URL#setURLStreamHandlerFactory and java.net.URLConnection#setContentHandlerFactory methods. This method must be called early by the framework implementation before any bundles are loaded to prevent any other code from calling these methods. Once these methods are called the objects cannot be unset. It is impossible to implement this in a bundle because the objects must never become unstable or unusable which would prevent bundles from being stopped, updated and uninstalled.

There is a requirement to allow for framework extension bundles to contribute to the implementation of the framework. The framework must use framework extension bundles to contribute content to the implementation of the framework.

5 Technical Solution

This RFC introduces the concept of an extension bundle.

5.1 Extension Bundle

An extension bundle must be treated differently than "normal" bundles by the framework. Extension bundles use the concept of fragment bundles from RFC 70. A fragment bundle appends to the classpath of a host bundle. Extension bundles append to the classpath of either the framework implementation or the boot classpath. Conceptually an extension bundle is a fragment to the framework implementation or the boot classpath.

5.1.1 Extension Bundle Fragments

An extension bundle is a fragment bundle. Fragment bundles specify a host bundle using the Fragment-Host manifest header (see RFC 70). An extension bundle specifies a Fragment-Host using an OSGi architected symbolic name to specify the system bundle as its fragment host. The following symbolic name is architected by the OSGi specification:

- system.bundle – The symbolic name for the System Bundle (or framework implementation). Fragments of the system.bundle are extension bundles.

The following directive is architected by the framework for Fragment-Host to specify the type of extension bundle.

- extension – A string taking one of the values of “framework” or “bootclasspath”. The default value is “framework”.
    - framework – indicates the fragment is a framework extension bundle.
    - bootclasspath – indicates the fragment is a bootclasspath extension bundle.

The following example uses the Fragment-Host manifest header to specify a framework implementation extension bundle.

```
Fragment-Host: system.bundle
```

The following example uses the Fragment-Host manifest header to specify a boot classpath extension bundle.

```
Fragment-Host: system.bundle; extension:=bootclasspath
```

If a fragment bundle specifies the “extension” directive with a symbolic name other than “system.bundle” then the directive is ignored by the framework.

5.1.2 Extension Bundle Classloader

An extension bundle does not have a bundle classloader associated with it. The classes and resources delivered by an extension bundle are loaded by the boot classloader (boot classpath extension bundles) or by the framework implementation classloader (framework extension bundles).

5.1.3 Extension Bundle Metadata

The metadata of an extension bundle is contained in a bundle manifest file like a normal bundle. Because extension bundles use the framework implementation classloader or the boot classpath classloader there are some limitations that must be placed on extension bundles. All manifest headers of an extension bundle are processed by the Framework just like a fragment bundle (See RFC 70) except for the following headers.

5.1.3.1 Import-Package and Require-Bundle

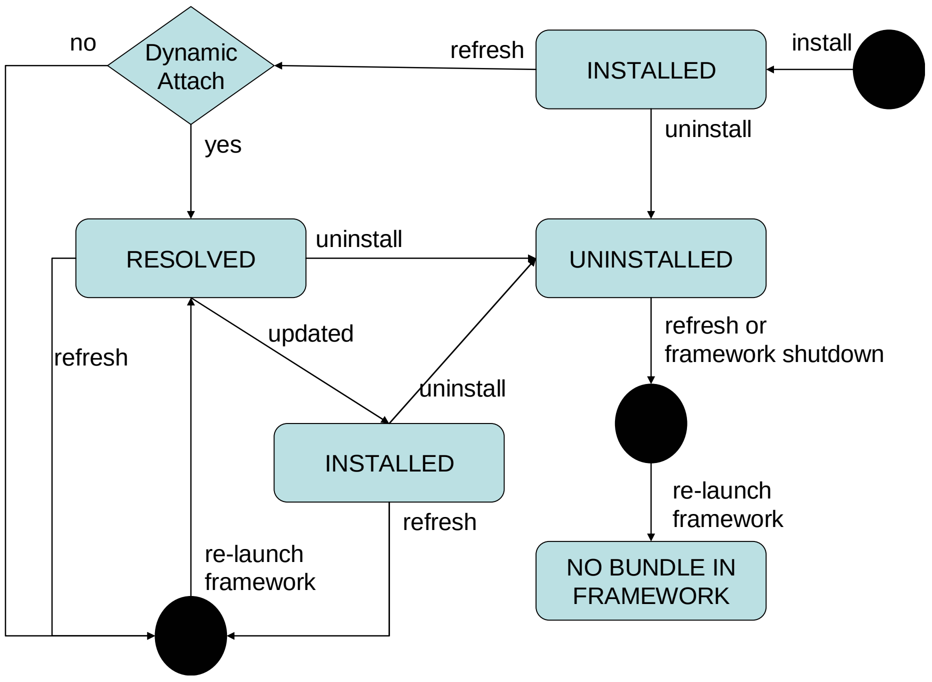
An extension bundle cannot specify any dependencies using Import-Package or Require-Bundle. This implies that an extension bundle cannot be unresolvable. If an extension bundle specifies an Import-Package or Require-Bundle header then a BundleException must be thrown indicating an install error.

5.1.3.2 Bundle-NativeCode

An extension bundle cannot deliver native code. If an extension bundle specifies a Bundle-NativeCode header then a BundleException must be thrown indicating an install error.

5.1.4 Extension Bundle Lifecycle

An extension bundle lifecycle behaves differently from normal bundles installed in the framework. The following chart shows the flow of an extension bundle’s lifecycle:



The following steps describe the lifecycle of an extension bundle:

1. When an extension bundle is installed it enters the INSTALLED state.

2. If the framework supports dynamically attaching fragments to an already resolved host (see RFC 70) then the extension bundle is allowed to enter the RESOLVED state at the frameworks discretion.

3. If the framework does not support dynamically attaching fragments to already resolved hosts then an extension bundle is not allowed to enter the RESOLVED state. If the extension bundle is refreshed then the framework must shutdown, the host VM must terminate and the framework must be re-launched.

4. If a RESOLVED extension bundle is refreshed then the framework must shutdown, the host VM must terminate and framework must be re-launched.

5. When a RESOLVED extension bundle is updated or UNINSTALLED it is not allowed to re-enter the RESOLVED state. If the extension bundle is refreshed then the framework must shutdown, the host VM must terminate and framework must be re-launched.

5.2 Boot Classpath Extension Bundles

A boot classpath extension bundle appends classes and resources to the boot classpath of the host VM. The classpath elements that are appended to the boot classpath are specified by the Bundle-Classpath header of the extension bundle. Extension Bundle-Classpath elements are appended to the boot classpath in the order in which the extension bundles are installed. That is ascending bundle id order.

How a framework configures the host VM to append the extension bundles classpath elements is implementation specific.

5.3 Framework Extension Bundles

A framework extension bundle appends classes and resources to the classloader of the framework. The classpath elements that are appended to the framework classpath are specified by the Bundle-Classpath header of the extension bundle. Extension Bundle-Classpath elements are appended to the framework implementation classpath in the order in which the extension bundles are installed. That is ascending bundle id order.

How a framework configures itself to append the extension bundles classpath elements is implementation specific.

5.4 Extension Bundles Unsupported

In some execution environments it may be impossible to support extension bundles. In such environments the framework must throw a BundleException from the BundleContext#installBundle methods when there is an attempt to install an extension bundle. The thrown BundleException must have a nested exception of type UnsupportedOperationException.

5.5 Detecting Framework Support for Extension Bundles

Support for extension bundles is optional, so a framework implementation must provide a means for detecting if extension bundles are supported. Therefore, the framework must set the following system properties to “true” if each type of extension bundle is supported.

```
org.osgi.supports.framework.extension
org.osgi.supports.bootclasspath.extension
```

If the property is not set or the value is unrecognized, then the value defaults to “false”.

6 Security

In an environment that has Java 2 security enabled the framework must perform additional security checks before allowing an extension bundle to be installed, updated or uninstalled. The following security checks must be performed:

1. In order for an extension bundle to successfully install or update the framework must check that the extension bundle has the permission AllPermission assigned to it. This means that the permissions of an extension bundle must be setup before the extension bundle is installed or updated.

2. The caller must have the permission AdminPermission[<bundle>, EXTENSIONLIFECYCLE] when installing, updating or uninstalling an extension bundle.

The permission AllPermission must be granted to extension bundles because they are loaded under the ProtectionDomain of either the boot classpath or the framework implementation. Both of these

ProtectionDomains have AllPermission granted to them. An extension bundle must not be allowed to install unless it already has been granted AllPermission.

7 Considered Alternatives

Use an Extension-Bundle header to specify the type of extension bundle (framework or boot-classpath). This has been replaced with the concept of fragment bundles for the framework implementation and the boot classpath.

8 Document Support

8.1 References

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

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

8.2 Author’s Address

Name	Thomas Watson
Company	IBM
Address	11501 Burnet Rd, Austin, TX 78758 USA
Voice	+1 512 838 4533
e-mail	tjwatson@us.ibm.com

Name	BJ Hargrave
Company	IBM
Address	11501 Burnet Rd, Austin, TX 78758 USA
Voice	+1 512 838 9938
e-mail	hargrave@us.ibm.com

8.3 Acronyms and Abbreviations

8.4 End of Document