RFC 191 Weaving Hook Enhancements

Final

22 Pages

Abstract

Addresses various enhancements to the Weaving Hook Service specification.

0 Document Information

0.1 License

0.2 Trademarks

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 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 11.1.

`Source code is shown in this typeface.`

0.6 Revision History

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

Revision	Date	Comments
	Jan 18 2013	*Accepted changes from Nov 19 and Nov 13 in preparation for the final draft.*

Revision	Date	Comments
	Nov 19 2012	*Removed requirement 4.2.4 for blacklisting weavers without import package permissions when adding dynamic imports. It was too strict since a weaver might decide to catch and not rethrow any security exceptions. Section 56.3.2 already covers the case where the security exception is thrown.* *Updated the Javadoc to clarify that it's the caller of woveClass.getDynamicImports().add(pkg), which may not be the weaver, that requires package permissions.* *John Ross, IBM jwross@us.ibm.com*
	Nov 13 2012	*Accepted changes from last revision.* *Added new enhancement for dynamically granting implied import package permissions to woven bundles with additional dynamic imports.* *John Ross, IBM jwross@us.ibm.com*
	Oct 11 2012	*Included BJ's javadoc changes.* *Minor wording change to Introduction.* *Additional entry added to Application Domain.* *Added new requirement.* *John Ross, IBM jwross@us.ibm.com*
	Oct 10 2012	*In section 5, clarified that the woven class listener is called in TRANSFORMING_FAILED and DEFINE_FAILED states.* *In section 5, corrected the definition of the isWeavingComplete() method to account for the new terminal states.* *Minor fixes to javadoc.* *John Ross, IBM jwross@us.ibm.com*
	Oct 09 2012	*Added state table and updated javadoc to account for the TRANSFORMING_FAILED and DEFINE_FAILED states.* *John Ross, IBM jwross@us.ibm.com*
	Oct 04 2012	*Minor javadoc updates. Added an additional considered alternative.* *Known outstanding issues:* 1. *Initial Spec Chapter* 2. *https://www.osgi.org/members/bugzilla/show_bug.cgi?id=2459* *John Ross, IBM jwross@us.ibm.com*

Revision	Date	Comments
	Oct 03 2012	*Added initial javadoc as section 10.* *John Ross, IBM jwross@us.ibm.com*
	Oct 02 2012	*Within section 5, clarified that the WovenClassListener is not called while the WovenClass is in the TRANSFORMING state since the WovenClass is mutable.* *John Ross, IBM jwross@us.ibm.com*
Initial	*Oct 01 2012*	*Initial draft.* *John Ross, IBM jwross@us.ibm.com*

1 Introduction

1.1 Woven Class Listener

This section originates from public Bug 144 and internal Bug 2430. There is no associated RFP. The scope of the bugs is much broader than the one in this RFC. In general, the bugs ask two questions: (1) how should Subsystems handle the DynamicImport-Package bundle manifest header and (2) how should Subsystems handle dynamic package requirements added by Weaving Hooks? This RFC narrows the scope to focus solely on how the Weaving Hook Service Specification may be enhanced to provide a foundation for (2) by defining a mechanism for observing the final set of dynamic package requirements before any attempt is made to wire them to capabilities.

1.2 Package Permission

This section originates from internal Bug 2466. There is no associated RFP. A security issue exists within the Weaving Hook Service specification since bundles containing woven classes with additional dynamic imports will not have the necessary permissions in order to import the packages. This RFC will address that issue.

2 Application Domain

2.1 Woven Class Listener

Chapter 3, section 9.2 of OSGi Core, Release 5, March 2012 defines the DynamicImport-Package bundle manifest header which allows dynamic package requirements to be wired to capabilities at class loading time.

Chapter 53 of OSGi Core, Release 5, March 2012 defines the Resolver Hook Service Specification allowing applications to influence the resolve operation and isolate the requirements and capabilities, including packages, of a collection of bundles.

Chapter 56 of OSGi Core, Release 5, March 2012 defines the Weaving Hook Service Specification allowing applications to observe classes being defined from bundles, transform the byte codes, and add dynamic package requirements as necessary. The added package requirements must conform to the syntax specified for the DynamicImport-Package bundle manifest header.

Chapter 134 of OSGi Enterprise, Release 5, March 2012 defines the Subsystem Service Specification providing a declarative model for defining collections of resources, including bundles, whose requirements and capabilities, including packages, are isolated within a region as defined by a sharing policy. This isolation is made possible through the Resolver Hook Service Specification.

2.2 Package Permission

Chapter 2 of OSGi Core, Release 5, March 2012 defines the optional security layer of the OSGi framework. Section 2.4 discusses permissions, and, in particular, 2.4.1 discusses implied permissions. These are permissions automatically granted to bundles by the framework for normal operation.

Chapter 50 of OSGi Core, Release 5, March 2012 defines the Conditional Admin Service. In particular, section 50.3 discusses explicit permissions, and how they relate to local, system, and implied permissions. Local permissions are those granted to a bundle by itself, such as through the OSGI-INF/permissions.perm file. System permissions are those granted to bundles by a management agent through the Permission Admin and Conditional Permission Admin services. Local and system permissions are intersected in order to guarantee management agents cannot exceed the local permissions. The effective permissions are the result of the union of implied permissions and the intersection.

Chapter 56 of OSGi Core, Release 5, March 2012 defines the Weaving Hook Service Specification allowing applications to observe classes being defined from bundles, transform the byte codes, and add dynamic package requirements as necessary. When framework security is enabled, bundles containing woven classes with additional dynamic imports must have the corresponding package permissions.

3 Problem Description

3.1 Woven Class Listener

The Resolver Hook Service Specification allows for the isolation of package requirements and capabilities at resolution time. The Weaving Hook Service Specification allows for the transformation of class byte codes, often resulting in the addition of dynamic package requirements, at runtime. However, a reliable mechanism for determining the final set of dynamic package requirements before they are wired to capabilities does not exist. This makes it impractical for isolation engines, such as Subsystems, to configure a sharing policy that ensures any capability satisfying a dynamic package requirement will be allowed into the region.

3.2 Package Permission

Bundles whose classes are being woven cannot know in advance what additional dynamic imports will be added. This means there is no way to provide the local permissions necessary to import the packages at class loading time. Moreover, the weaver cannot provide these permissions using Permission or Conditional Permission Admin since the intersection of local and system permissions prohibits it. There is, therefore, currently no dynamic way of granting the necessary package import permissions to bundles containing woven classes with additional dynamic imports.

4 Requirements

4.1 Woven Class Listener

4.1.1 Dynamic package requirements added by Weaving Hook Services MUST be observable in their final state. That is, they MUST NOT change once observed.

4.1.2 Dynamic package requirements added by Weaving Hook Services MUST be observable before any attempt at wiring them to capabilities is made.

4.1.3 The solution MUST be backwards compatible.

4.1.4 Woven classes are not created when no weaving hooks are present; therefore, frameworks MUST NOT call woven class listeners if no weaving hooks are called.

4.2 Package Permission

4.2.1 The framework MUST grant implied import package permissions to a woven bundle with additional dynamic imports.

4.2.2 The weaving bundle MUST have import package permission for each dynamic import it added.

4.2.3 Implied import package permissions for the woven bundle MUST be granted after the TRANSFORMED state.

5 Technical Solution

5.1 Woven Class Listener

A new interface with the following signature is added to the org.osgi.framework.hooks.weaving package.

```
public interface WovenClassListener {

            void modified(WovenClass wovenClass);

}
```

The WovenClassListener follows the whiteboard pattern. Services registered under this interface will be called by the framework whenever a WovenClass enters the TRANSFORMED, DEFINED, TRANSFORMING_FAILED, and DEFINE_FAILED states. A listener does not receive a callback for the TRANSFORMING state because the WovenClass is mutable, and the listener is intended to be read only.

The following states are added to the WovenClass.

- TRANSFORMING – This is the initial state. Weaving hooks are being called and in the process of transforming the woven class.
- TRANSFORMED – The last weaving hook has been called, and no exceptions were thrown.
- DEFINED – This is a terminal state. The class has been defined, and the bundle wiring has been updated.

- TRANSFORMING_FAILED – This is a terminal state. A weaving hook threw an exception.
- DEFINE_FAILED – This is a terminal state. A class definition failure occurred.

Source State	Target State	Event	Procedure
-	TRANSFORMING	A bundle class load request was made.	• isWeavingComplete == false. • getDefinedClass == null. • Woven class mutable. • Notify weaving hooks.
TRANSFORMING	TRANSFORMED	All weaving hooks have been notified.	• isWeavingComplete == false. • getDefinedClass == null. • Woven class immutable. • Notify woven class listeners.
TRANSFORMING	TRANSFORMING_FAILED	A weaving hook threw an exception.	• isWeavingComplete == true. • getDefinedClass == null. • Woven class immutable. • Notify woven class listeners.
TRANSFORMED	DEFINED	All woven class listeners have been notified.	• isWeavingComplete == true. • getDefinedClass != null. • Update bundle wiring. • Woven class immutable. • Notify woven class listeners.
TRANSFORMED	DEFINE_FAILED	All woven class listeners have been notified. Class definition failure.	• isWeavingComplete == true. • getDefinedClass == null. • Woven class immutable. • Notify woven class listeners.

A new method with the following signature is added to WovenClass in order retrieve the new state.

```
public int getState();
```

The WovenClass.isWeavingComplete method is redefined as follows.

```
public boolean isWeavingComplete() {

        return   ((getState()   &   (DEFINED   |   TRANSFORMING_FAILED   |
        DEFINE_FAILED)) != 0;

}
```

This solution will allow isolation engines, such as Subsystems, to observe the final set of dynamic package requirements and act upon them before any attempt is made to wire them to capabilities.

5.2 Package Permission

When a weaver adds a dynamic package (WovenClass.getDynamicImports().add(somepkg)), the framework must grant the woven bundle the "implied" (see 50.3) permission to import that package. However, the weaver must possess permission to import that package. The framework must check this during the add method via SecurityManager.checkPermission(new PackagePermission(somepkg,"import")). If the weaver has the permission, the woven will be granted the "implied" permission for all the packages in the dynamic import list immediately before defining the class (that is, the permissions are granted after the TRANSFORMED state). If the weaver does not have the permission, a SecurityException is thrown which can cause the weaver to be blacklisted.

Since the checkPermission used the simple (no filter, no exporting bundle) PackagePermission variant to check the permission of the weaver, weavers must be granted permissions for importing package that can be implied this way. This generally means the weaver must be granted simple (no filter) PackagePermissions to import the package. This is true even if the weaver bundle is in fact the exporter of the package and thus would only require permission to export the package. In order for the framework to grant the implied permission to import the package, the weaver must have the explicit permission to import the package.

6 JMX API

There are no known JMX API considerations.

7 Initial Spec Chapter

7.1 Woven Class Listener

This section will affect the Weaving Hook Service Specification, Chapter 56 of OSGi Core, Release 5, March 2012.

7.2 Package Permission

This section will affect Implied Permissions, Chapter 2, Section 4.1, and the Weaving Hook Service Specification, Chapter 56, of OSGi Core, Release 5, March 2012.

8 Considered Alternatives

8.1 Woven Class Listener

One alternative is to leave the Weaving Hook Service Specification unmodified and let Subsystems register a Weaving Hook Service with the lowest possible service ranking. This was rejected as unreliable since there is no guarantee that the set of dynamic package imports would be final. Other applications could register services at the same service ranking, get called later, and modify the set.

Another alternative is to add additional API to Subsystems allowing for the dynamic modification of sharing policies. Either Weaving Hook or Subsystem providers would be responsible for synchronizing the dynamic package requirements with the sharing policies. This was rejected as unreliable, complex, and burdensome. Either Weaving Hook providers responsible for adding the dynamic imports would also need to update Subsystems, or the Subsystem provider would need to know all of the dynamic imports that are going to be added beforehand. There is no desire to couple the Weaving Hook and Subsystems API.

Still another alternative is to allow Subsystems users to specify a list of packages that must always be imported as part of a subsystem's configuration. This was rejected for the same reasons as the second alternative.

The proposed solution is the best alternative because it (1) is dynamic in that it handles all possible additional package imports and (2) places the burden on the Subsystems implementer, not the user.

An additional alternative discussed involved having the WovenClassListener become a more powerful version of WeavingHook. Listeners would receive notifications of all WovenClass states, including TRANSFORMING. This would allow users to register a single service in order to observe the full lifecycle of a WovenClass. This alternative was rejected because a mutator already existed (i.e. WeavingHook), there was no desire to deprecate it, and having two mutators would be potentially confusing. Plus there seemed to be no compelling reason that required a single listener for observing all of the states. More comfort was found in the separation of concerns where weaving hooks can fail a class load but woven class listeners cannot.

9 Security Considerations

9.1 Woven Class Listener

Implementers of WovenClassListener must have ServicePermission[..WovenClassListener,REGISTER] in order to register a service.

No admin permissions are necessary since the WovenClass will be immutable.

9.2 Package Permission

See 5.2.

10 Javadoc

OSGi Javadoc
11/19/12 11:22 AM

Package Summary		Page
org.osgi.framework.hooks.weaving	Framework Weaving Hooks Package Version 1.1.	Error: Reference source not found

Package org.osgi.framework.hooks.weaving

Framework Weaving Hooks Package Version 1.1.

See:
    Description

Interface Summary		***Page***
WovenClass	A class being woven.	Error: Reference source not found
WovenClassListener	OSGi Framework Woven Class Listener Service.	Error: Reference source not found

Package org.osgi.framework.hooks.weaving Description

Framework Weaving Hooks Package Version 1.1.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.framework.hooks.weaving; version="[1.1,2.0)"
```

Interface WovenClass

org.osgi.framework.hooks.weaving

`public interface WovenClass`

A class being woven. This object represents a class being woven and is passed to each `org.osgi.framework.hooks.weaving.WeavingHook` for possible modification. It allows access to the most recently transformed class file bytes and to any additional packages that should be added to the bundle as dynamic imports.

Upon entering one of the terminal states, this object becomes effectively immutable.

Version:

Id

NotThreadSafe

Field Summary		***Page***
`int`	**DEFINE_FAILED** The woven class failed to define.	rror: Reference source not found
`int`	**DEFINED** The woven class has been defined.	Error: Reference source not found
`int`	**TRANSFORMED** The woven class has been transformed.	Error: Reference source not found
`int`	**TRANSFORMING** The woven class is being transformed.	Error: Reference source not found
`int`	**TRANSFORMING_FAILED** The woven class failed to transform.	Error: Reference source not found

Method Summary		Page
org.osgi.framework.wiring.BundleWiring	**getBundleWiring**() Returns the bundle wiring whose class loader will define the woven class.	rror: Reference source not found
byte[]	**getBytes**() Returns the class file bytes to be used to define the named class.	Error: Reference source not found
String	**getClassName**() Returns the fully qualified name of the class being woven.	Error: Reference source not found
Class<?>	**getDefinedClass**() Returns the class defined by this woven class.	Error: Reference source not found
List<String>	**getDynamicImports**() Returns the list of dynamic import package descriptions to add to the bundle wiring for this woven class.	Error: Reference source not found
ProtectionDomain	**getProtectionDomain**() Returns the protection domain to which the woven class will be assigned when it is defined.	Error: Reference source not found

`int`	`getState()` Returns the current state of this woven class.	Error: Reference source not found
`boolean`	`isWeavingComplete()` Returns whether weaving is complete in this woven class.	Error: Reference source not found
`void`	`setBytes(byte[] newBytes)` Set the class file bytes to be used to define the `named` class.	Error: Reference source not found

Field Detail

TRANSFORMING

`public static final int TRANSFORMING = 1`

The woven class is being transformed.

The woven class is in this state while `weaving hooks` are being called. The woven class is mutable so the `class bytes` may be `modified` and `dynamic imports` may be added. If a weaving hook throws an exception the state transitions to `TRANSFORMING_FAILED`. Otherwise, after the last weaving hook has been successfully called, the state transitions to `TRANSFORMED`.

Since:
1.1

TRANSFORMED

`public static final int TRANSFORMED = 2`

The woven class has been transformed.

The woven class is in this state after `weaving hooks` have been called and before the class is defined. The woven class cannot be further transformed. The woven class is in this state while defining the class. If a failure occurs while defining the class, the state transitions to `DEFINE_FAILED`. Otherwise, after the class has been defined, the state transitions to `DEFINED`.

Since:
1.1

DEFINED

`public static final int DEFINED = 4`

The woven class has been defined.

The woven class is in this state after the class is defined. The woven class cannot be further transformed. This is a terminal state. Upon entering this state, this object is effectively immutable, the `bundle wiring` has been updated with the `dynamic import requirements` and the class has been `defined`.

Since:
1.1

TRANSFORMING_FAILED

`public static final int TRANSFORMING_FAILED = 8`

The woven class failed to transform.

The woven class is in this state if a `weaving hook` threw an exception. The woven class cannot be further transformed or defined. This is a terminal state. Upon entering this state, this object is effectively immutable.

Since:

1.1

DEFINE_FAILED

`public static final int DEFINE_FAILED = 16`

The woven class failed to define.

The woven class is in this state when a failure occurs while defining the class. The woven class cannot be further transformed or defined. This is a terminal state. Upon entering this state, this object is effectively immutable.

Since:

1.1

Method Detail

getBytes

`byte[] getBytes()`

Returns the class file bytes to be used to define the `named` class.

While in the `TRANSFORMING` state, this method returns a reference to the class files byte array contained in this object. After leaving the `TRANSFORMING` state, this woven class can no longer be transformed and a copy of the class file byte array is returned.

Returns:

The bytes to be used to define the `named` class.

Throws:

`SecurityException` - If the caller does not have `AdminPermission[bundle,WEAVE]` and the Java runtime environment supports permissions.

setBytes

`void setBytes(byte[] newBytes)`

Set the class file bytes to be used to define the `named` class. This method must not be called outside invocations of the `weave` method by the framework.

While in the `TRANSFORMING` state, this method replaces the reference to the array contained in this object with the specified array. After leaving the `TRANSFORMING` state, this woven class can no longer be transformed and this method will throw an `IllegalStateException`.

Parameters:

`newBytes` - The new classfile that will be used to define the `named` class. The specified array is retained by this object and the caller must not modify the specified array.

Throws:

`NullPointerException` - If newBytes is `null`.

`IllegalStateException` - If state is `TRANSFORMED`, `DEFINED`, `TRANSFORMING_FAILED` or `DEFINE_FAILED`.

`SecurityException` - If the caller does not have `AdminPermission[bundle,WEAVE]` and the Java runtime environment supports permissions.

getDynamicImports

`List<String> getDynamicImports()`

Returns the list of dynamic import package descriptions to add to the `bundle wiring` for this woven class. Changes made to the returned list will be visible to later `weaving hooks` called with this object. The returned list must not be modified outside invocations of the `weave` method by the framework.

After leaving the `TRANSFORMING` state, this woven class can no longer be transformed and the returned list will be unmodifiable.

If the Java runtime environment supports permissions, the caller must have `AdminPermission[bundle,WEAVE]` and `PackagePermission[package,IMPORT]` to modify the returned list.

Returns:

A list containing zero or more dynamic import package descriptions to add to the bundle wiring for this woven class. This list must throw `IllegalArgumentException` if a malformed dynamic import package description is added.

See Also:

"Core Specification, Dynamic Import Package, for the syntax of a dynamic import package description."

isWeavingComplete

`boolean isWeavingComplete()`

Returns whether weaving is complete in this woven class. Weaving is complete after the class is defined.

Returns:

`true` if `state` is `DEFINED`, `TRANSFORMING_FAILED` or `DEFINE_FAILED`; `false` otherwise.

getClassName

`String getClassName()`

Returns the fully qualified name of the class being woven.

Returns:

The fully qualified name of the class being woven.

getProtectionDomain

`ProtectionDomain getProtectionDomain()`

Returns the protection domain to which the woven class will be assigned when it is defined.

Returns:

The protection domain to which the woven class will be assigned when it is defined, or `null` if no protection domain will be assigned.

getDefinedClass

`Class<?> getDefinedClass()`

Returns the class defined by this woven class. During weaving, this method will return `null`. Once weaving is `complete`, this method will return the class object if this woven class was used to define the class.

Returns:

The class associated with this woven class, or `null` if weaving is not complete, the class definition failed or this woven class was not used to define the class.

getBundleWiring

`org.osgi.framework.wiring.BundleWiring getBundleWiring()`

Returns the bundle wiring whose class loader will define the woven class.

Returns:

The bundle wiring whose class loader will define the woven class.

getState

`int getState()`

Returns the current state of this woven class.

A woven class can be in only one state at any time.

Returns:

Either `TRANSFORMING`, `TRANSFORMED`, `DEFINED`, `TRANSFORMING_FAILED` or `DEFINE_FAILED`.

Since:

1.1

Interface WovenClassListener

org.osgi.framework.hooks.weaving

`public interface WovenClassListener`

OSGi Framework Woven Class Listener Service.

Bundles registering this service will receive notifications whenever a `woven class` completes a `state` transition. Implementers will therefore be unable to modify the woven class in contrast with `weaving hooks`.

Receiving a woven class in the `TRANSFORMED` state allows listeners to observe the modified `byte codes` before the class has been `DEFINED` as well as the additional `dynamic imports` before the `bundle wiring` has been updated.

Woven class listeners are synchronously `called` when a woven class completes a state transition. The woven class processing will not proceed until all woven class listeners are done.

If the Java runtime environment supports permissions, the caller must have `ServicePermission[WovenClassListener,REGISTER]` in order to register a listener.

Since:
    1.1

Version:
    $Id$

ThreadSafe

Method Summary		***Page***
`void`	**`modified`**`(WovenClass wovenClass)` Receives notification that a `woven class` has completed a state transition.	rror: Reference source not found

Method Detail

modified

`void modified(WovenClass wovenClass)`

Receives notification that a `woven class` has completed a state transition.

The listener will be notified when a woven class has entered the `TRANSFORMED`, `DEFINED`, `TRANSFORMING_FAILED` and `DEFINE_FAILED` states.

If this method throws any exception, the Framework must log the exception but otherwise ignore it.

Parameters:
    `wovenClass` - The woven class that completed a state transition.

11 Document Support

11.1 References

[1]. OSGi Enterprise, Release 5, March 2012

[2]. OSGi Core, Release 5, March 2012

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

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

11.2 Author's Address

Name	John Ross
Company	IBM
Address	11501 Burnet Road, Building 903, Office 4C-002, Austin, TX 78758
Voice	+1 512 973 2216
e-mail	jwross@us.ibm.com

11.3 Acronyms and Abbreviations

11.4 End of Document