RFC 204 - Framework Extension Activators

Draft

9 Pages

Abstract

The OSGi Core specification version 4 introduced framework extension bundles as a way to deliver optional parts of the Framework implementation. The OSGi Core specification version 4.2 and 4.3 introduced various framework hooks that allowed bundles on top of the framework to augment certain behaviors of the framework. For example, the service, bundle and resolver hooks together may be used to implement a scoping model for isolating groups of bundles with in the same framework. In many scenarios it is desired to have these hooks be available (registered) as early as possible to guarantee consistent behavior for the complete lifecycle of the framework. Currently there is no standard to ensure the hook implementations are available when a framework is initialized because hook implementations are registered as services by normal bundles which can only be activated after a framework has been initialized.

This RFC specifies extension bundle activators which can be used by a framework extension to hook into the initialization process of the framework, including the ability to register framework hook services.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

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 10.1.

```
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
Initial	Aug 27 2013	Thomas Watson, IBM tjwatson@us.ibm.com Initial version
	Sept 4 2013	Thomas Watson, IBM tjwatson@us.ibm.com – Modified requirements based on CPEG feedback – Added the technical section

1 Introduction

The OSGi Core specification version 4 introduced framework extension bundles as a way to deliver optional parts of the Framework implementation. The OSGi Core specification version 4.2 and 4.3 introduced various framework hooks that allowed bundles on top of the framework to augment certain behaviors of the framework. For example, the service, bundle and resolver hooks together may be used to implement a scoping model for isolating groups of bundles with in the same framework. In many scenarios it is desired to have these hooks be available (registered) as early as possible to guarantee consistent behavior for the complete lifecycle of the framework. Currently there is no standard to ensure the hook implementations are available when a framework is initialized because hook implementations are registered as services by normal bundles which can only be activated after a framework has been initialized.

This RFC specifies extension bundle activators which can be used by a framework extension to hook into the initialization process of the framework, including the ability to register framework hook services.

2 Application Domain

The basic framework provides complete visibility for any bundle to any other bundle, service or capability. In certain use cases it is important to provide a notion of isolation or scope to a group of bundles. The Enterprise specification has defined subsystems as a standard way to isolate groups of bundles.

In environments that include an isolation model, such as subsystems, framework hooks are used extensively to provide the implementation of the isolation model. In order to provide a consistent behavior the framework hook service implementations must be registered before any operation occurs that requires isolation. For example, resolution operations, interactions with the service registry etc.

3 Problem Description

In order for framework hook service implementations to provide consistent behavior they must be registered with the service registry and available before any framework operations occur where they need to influence the behavior. For service and bundle hooks this is typically not an issue if the hook implementations are registered by a bundle at a very low start-level (for example start-level 1). This allows them to be available before any other normal bundles interact with the service registry or the set of installed bundles.

Resolver hooks participate in the resolve process of the framework. The specification allows resolution to occur at any time. Resolution may occur before even starting the first bundle (including the bundle that implements a resolver hook). This implies that bundles implementing resolver hooks must do some kind of consistency check each time they come on line to see if the wiring has changed since the last time they were active. One way to do this would involve the resolve hook implementation persisting the complete wiring graph on shutdown and then comparing the graph to the current wiring graph on restart. This is likely to be complicated and an expensive operation.

Conceptually framework hook implementations extend the framework and can be considered part of the framework. In many cases it is desirable to make the hooks available as soon as the framework is initialized. One way to solve this issue is to introduce extension bundle activators which can be declared by a framework extension. This would allow the activator to be called during framework initialization to establish the framework hook as early as possible. It would also allow the framework hook to remain available as late as possible during the framework shutdown process.

4 Requirements

1. A framework extension must be able hook into the initialization process of the framework and gain access to the system bundle's context~~to specify an extension bundle activator~~.
2. A bootclasspath extension must NOT be able to ~~specify an extension bundle activator~~hook into the initialization process of the framework.
3. ~~An extension bundle activator must be specified in an extension bundle's manifest using a new header. The existing Bundle-Activator header must not be used. [design choice]~~
4. ~~Framework configurations that have framework extensions with extension bundle activators the initialization process (Framework.init()) must call the BundleActivator.start(BundleContext) method for each framework extension that is resolved.~~
5. During ~~framework~~ the initialization process, if existing framework extensions are installed and require resolution then the resolve operation must be scoped to only include the system bundle and its fragments. This resolution operation must occur before ~~calling the first extension bundle activator start method~~allowing framework extensions to have access to the system bundle's context.
6. If a framework extension is allowed to be installed and resolve dynamically after framework initialization~~,~~ without a framework restart, then the extension bundle must have access to the system bundle's context ~~activator start method must be called~~ as soon as the extension bundle is resolved (the current CT implies that dynamic resolution is required without a restart).
7. A framework extension must be able to hook into the ~~During~~ the framework shutdown process~~ the extension bundle activator BundleActivator.stop(BundleContext) method must be called for each framework extension that is resolved.~~ ~~The stop methods are called~~ This must happen after all other bundles are stopped ~~decrementing the start-level to 0~~ and before firing one of the FrameworkEvent.STOPPED events which Framework.waitForStop is listening for~~. [Called after all other bundles are stopped]~~.
8. Uninstalling and updating a framework extension must NOT ~~result in the extension bundle activator stop method being called~~remove the extension from the framework shutdown process~~. Extension bundle activator stop methods are ONLY called during framework shutdown. [~~ The actual effects of update/uninstall do not take effect ~~the lifecycle for the activator~~ until the framework is shutdown~~] .~~

5 Technical Solution

A new header is introduced for framework extensions to allow a bundle activator to be specified. The following sections will be merged into the Extension Bundles chapter (3.15)

5.1 Framework Extension Activator

A framework extension may hook into the Framework initialization and shutdown process by specifying an Extension Bundle Activator. The BundleActivator interface defines methods that the Framework invokes for Extension Bundle Activators when the Framework is initialized and shutdown.

To inform the OSGi environment of a fully qualified class name serving as its Extension Bundle Activator, a framework extension developer must declare an ExtensionBundle-Activator manifest header in the framework extension bundle's manifest file. The following is an example of an ExtensionBundle-Activator:

```
ExtensionBundle-Activator: com.acme.Activator
```

The class acting as an Extension Bundle Activator must implement the BundleActivator interface, be declared public, and have a public default constructor so an instance of it may be created with `Class.newInstance`.

Supplying an Extension Bundle Activator is optional and only valid for extensions of type framework.

5.1.1 Initialization

The last step during Framework initialization (Section 4.2.4) is to call each Extension Bundle Activator. The start method of each Extension Bundle Activator declared by resolved framework extensions is called to inform the framework extension of the initializing framework. While calling Extension Bundle Activator start methods the framework must be in the STARTING state and have a valid bundle context. The order in which framework extension activators are called is not specified and should not be relied upon. Any exception thrown by an extension bundle activator start method should result in a FrameworkEvent of type ERROR.

XXX – ERROR event not much value here unless the framework has built in logging of such errors, should we fail init? Equinox would log such errors to a log and continue framework init. I'm not a big fan of failing initialization because it puts the framework persistent storage area into an unlaunchable state with no way to recover except to use a clean launch. This will inevitably result a severe bug report when someone installs a bad extension into a product like Eclipse.

During the initialization process a framework must attempt to resolve all installed framework extensions. All resolve operations that occur during initialization must be scoped to only include the system bundle and its fragments. This is necessary to avoid resolution operations which change the wiring of normal bundles before the Extension Bundle Activators are called.

5.1.2 Update and Uninstall

Unlike normal bundles, updating or uninstalling an extension bundle does not take effect at runtime until the framework is shutdown and restarted. The old content of the bundle must remain attached to the system bundle until the framework is shutdown. This extends to the Extension Bundle Activators that have had the start method called. All started Extension Bundle Activators must not have the stop method called until the framework is shutdown.

5.1.3 Installing

When a framework extension is installed a Framework may allow the extension to become resolved dynamically, without a Framework restart. If a framework extension is allowed to be installed and resolve dynamically after framework initialization, then the Extension Bundle Activator start method must be called as soon as the extension bundle is resolved. This must happen before the resolved bundle event is fired for the extension bundle.

5.1.4 Shutdown

Section 4.2.6 discusses the Framework shutdown process. Extension Bundle Activators are called during the shutdown process after all other bundles are stopped. The Extension Bundle Activator stop methods are called after the start-level moves to 0 and before disabling event handling. While calling Extension Bundle Activator stop methods the framework must be in the STOPPING state and have a valid bundle context.

The Framework must guarantee that if a BundleActivator.start method has executed successfully for a framework extension, that same BundleActivator object must be called with its BundleActivator.stop method when the

framework is shutdown. After calling the stop method, that particular BundleActivator object must never be used again. Extension Bundle Activators that threw an exception during start must not be called on shutdown.

6 Data Transfer Objects

None for this RFC

7 Javadoc

None at this time. A constant will need to be defined to specify the ExtensionBundle-Activator header.

8 Considered Alternatives

Considered adding a FrameworkWiring.getChangeCount method that could be used by bundles with resolver hooks that could be used to determine if some resolve operation occurred while they were not available or registered. This solution is really only useful for frameworks that cache their resolution state between framework sessions. Otherwise the change count would be different for each restart and require a resolver hook implementation to always do a consistency check.

9 Security Considerations

Framework extensions will be handed the BundleContext of the system bundle without performing any security check. This is acceptable because framework extensions must have all permissions in order to be installed successfully.

10 Document Support

10.1 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

10.2 Author's Address

Name	Thomas Watson
Company	IBM
Address	
Voice	
e-mail	tjwatson@us.ibm.com

10.3 Acronyms and Abbreviations

10.4 End of Document