Framework Launching

Final

15 Pages

Abstract

This RFC describes a proposed specification for a framework launching interface for 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 7.1.

`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	05 MAR 2008	Peter.Kriens@aQute.biz
Additional text	03 APR 2008	Added a large number of sections, mainly booting and more details about the processing of scripts. Also completely changed the API and added a problem description and requirements section. Changes are so massive that it was not that useful to track changes (and I forgot anyway)
Update	15 MAY 2008	More details in booting, minor update in the language aspects
Update	31 JUL 2008	Changed some method names, updated Javadoc and added to build
Update	5 AUG 2008	Updated javadoc section. Changed “boot” to “launch”.
Update	15 AUG 2008	Clarify the state transitions of the system bundle Thomas Watson tjwatson@us.ibm.com
Update	18 AUG 2008	Added a table, minute corrections, Peter Kriens
Update	27 AUG 2008	Update from CPEG discussions ● configuration properties are only set once at construction ● clarified state transitions ● Changed libraries configuration property to library.extensions ● Added org.osgi.framework.startlevel configuration property
Add concurrency package	09/09/08	Added the concurrency package proposed in bug 714. Reorganized the document. Clean up a few things. BJ Hargrave

Revision	Date	Comments
Update after CPEG mtg	09/18/08	Removed concurrency. BJ Hargrave
More updates from F2F mtg	10/09/08	● Imported section 4.7 “Framework Startup and Shutdown” from the core spec into the RFC. ● Specified the actions for framework (SystemBundle) init, startup and shutdown. ● Added org.osgi.framework.system.packages.extra option ● Added org.osgi.framework.storage.clean option ● Changed org.osgi.framework.root.certificates to org.osgi.framework.trust.repositories ● Added a security section Tom Watson
Update	10/29/2008	● Removed system properties check for framework configuration properties ● Updated to reflect the name change for the SystemBundle interface to the Framework interface. Tom Watson
Update	01/19/2009	Updates from Bedford F2F meeting ● Specify handling of org.osgi.framework.security when a security manager is already installed ● Document AllPermission requirement when constructing a Framework object. ● Document sections to be included in Core, Compendium, and Proposed sections of the specification.
Update	02/27/2009	Update to use the META-INF services approach to find a FrameworkFactory to construct Framework instances
Final	2009 April 21	Move command line design to RFC 147.

1 Introduction

This RFC is an solution for RFP 80 Framework booting. This RFC outlines the interfaces necessary to implement a generic start up solution for different implementations of frameworks.

2 Problem Description

This RFC addresses the problem of standardized external access. The purpose of this RFC is to allow any OSGi framework to launched, configured, and controlled externally from a console, telnet session, serial port or script.

2.1 Framework Launching

Every OSGi framework must invent its own technology for getting started. Therefore, code that may need to launch the framework from various places must write code that is proprietary for the particular OSGi vendors' implementation, if they wish to support more than one framework.

In enterprises, where the issues of vendor lock-in can cause a barrier to adoption of the system, this issue becomes magnified. While it should certainly be the case that each vendor can supply add-ons and extra features, the standard portions of all OSGi frameworks could be encapsulated in a Framework Launching specification. Having such a specification would increase the consistency and quality of compliant OSGi frameworks, and allay fears about vendor lock-in.

3 Requirements

3.1 Launching

- The solution should allow for scripting languages to behave similarly for all compliant OSGi frameworks.
- The solution must allow for starting, restarting and stopping compliant OSGi frameworks without prior knowledge of the framework.

- The solution should allow for starting compliant frameworks in the same or different processes (though it would be OK to fall back on the Java Process verbs if necessary)
- The solution should handle the problems associated with launching multiple OSGi frameworks inside the same JVM process space.
- The solution must be able to handle vendor specific extensions.

4 Technical Solution

The designs in this section are to be part of the Core Specifications.

Parts of this section are a copy of section 4.7 of the R4.1 specification with modifications for this RFC

4.1 The System Bundle

A Framework implementation must be started before any bundles can be started and any services can be provided. Frameworks must designate a FrameworkFactory class that is used to construct a Framework bundle instance.

4.1.1 Framework Factory

A framework factory is used for creating Framework bundle instances. A framework implementation jar must contain the following resource:

```
/META-INF/services/org.osgi.framework.launch.FrameworkFactory
```

This UTF-8 encoded resource must contain the name of the framework implementation's FrameworkFactory implementation class. Space and tab characters, including blank lines, in the resource must be ignored. The number sign ('#' \u0023) and all characters following it on each line are a comment and must be ignored.

Launchers can find the name of the FrameworkFactory implementation class in the resource and then load and construct a FrameworkFactory object for the framework implementation. The FrameworkFactory implementation class must have a public, no-argument constructor. Java™ SE 6 introduced the `ServiceLoader` class which can create a FrameworkFactory instance from the resource.

The framework factory has a method to create new Framework instances. This method takes a single argument of type Map for configuration. If framework properties are not provided by the configuration argument, the created framework instance must use some reasonable default configuration appropriate for the current VM. For example, the system packages for the current execution environment should be properly exported. The specified configuration argument may be `null`. The created framework instance must copy any information needed from the specified configuration argument since the configuration argument can be changed after the framework instance has been created.

A newly constructed framework bundle must be in the INSTALLED state.

When constructing a Framework a `SecurityException` is thrown if the caller does not have `AllPermission`, and the Java Runtime Environment supports permissions.

4.2 Framework Initialization, Startup and Shutdown

4.2.1 Initialization

When the framework bundle is initialized, by calling the init() method, the following actions must occur before the init() method returns:

- Event handling is enabled. Events can now be delivered to listeners. Events are discussed in Events on page 97.
- If a security manager is configured (see org.osgi.framework.security configuration property) then a security manager is installed
- The Framework start level is set to 0
- A valid BundleContext is made available to the Framework bundle.
- Bundles installed in the Framework's persistent storage area are reified and their state is set to INSTALLED.
- The Framework services are made available in the service registry (e.g. PackageAdmin, ConditionalPermissionAdmin).
- The Framework enters the STARTING state.

The order in which these actions take place is not not important because nothing external to the framework can observe the initialization process, therefore the order is not specified and is implementation specific.

4.2.2 Startup

When a framework bundle is started, by calling the start() method, the following actions must occur:

1. If the framework bundle is not currently in the STARTING state then the framework bundle is initialized.
2. All installed bundles previously recorded as being started must be started as described in the Bundle.start method. Any exceptions that occur during startup must be wrapped in a BundleException and then published as a Framework event of type FrameworkEvent.ERROR. Bundles and their different states are discussed in The Bundle Object on page 81.
    - If the Framework implements the optional Start Level specification, this behavior is different. See Start Level Service Specification on page 203. The Framework start level is incremented to the value specified by the configuration property org.osgi.framework.startlevel.
    - Any bundles that specify an activation policy must be treated according to their activation policy, see Activation Policies on page 85.
3. The Frameworkbundle enters the ACTIVE state.
4. A Framework event of type FrameworkEvent.STARTED is broadcast.

4.2.3 Shutdown

The Framework will also need to be shut down on occasion. Shutdown can also be initiated by stopping the Framework bundle, covered in The Framework Bundle on page 96. When the Framework is shut down, the following actions must occur in the given order:

1. The system bundle enters the STOPPING state.
2. All ACTIVE bundles (except the system bundle) are stopped as described in the Bundle.stop method, except that their persistently recorded state indicates that they must be restarted when the Framework is next started is kept unchanged. Any exceptions that occur during shutdown must be wrapped in a BundleException and then published as a Framework event of type FrameworkEvent.ERROR. If the Framework implements the optional Start Level specification, this behavior is different. See Start Level Service Specification on page 203. During the shutdown, bundles with a lazy policy must not be activated even when classes are loaded from them.
3. The Framework start-level is set to 0.
4. Unregister the Framework services.
5. Event handling is disabled
6. All bundles (except system bundle) enter the INSTALLED state and all resources associated with Bundles should be freed (e.g. class loaders, open files etc.). The Framework should no longer hold references to the Bundle objects.
7. Any remaining resources held by the Framework should be freed (e.g. All threads used for event dispatching or to keep the VM alive should be shutdown).
8. The Framework bundle enters the RESOLVED state.

At this point the Framework bundle can be re-initialized or re-started.

4.3 Example using the Framework Bundle

The current specification provides a very detailed description of the Framework Bundle. The instantiated object represents this Framework bundle in an unstarted state (INSTALLED). However, after init has been called and the Framework bundle is in the STARTING state, the Framework bundle must be able to provide a valid Bundle Context that can be used to install bundles and perform other operations. Any bundles installed in the Framework must not be started before the Framework bundle itself is started. The Framework bundle can only be configured once at construction time. The Framework Bundle object can be started, stopped, and started again, ad nauseum.

The following is a short example of creating a Felix framework, adding bundles to it, launching the framework and then waiting for it to stop. This assumes that some bundle (for example a shell) will initiate the stop command to the system bundle.

- `f = new org.apache.felix.FelixFactory.newFramework`
- `f init`
- `context = f bundleContext`
- `$context installBundle http://www.aQute.biz/repo/aQute/sample.jar`
- `f start`
- `f waitForStop`

This example makes the main thread wait for the framework to finish.

4.4 Life Cycle Issues

A framework must never do System.exit(n) when stopped. It is the responsibility of the configurator to exit.

A framework can make repeated start/stop cycles. The semantics of stopping a framework are described in the core specification. It is not possible to change the properties. The options in the start and stop methods are ignored.

The following state diagram shows the activities and transitions (>) that should occur. The number after the state is the start level, where n is the set start level).

	INSTALLED	RESOLVED (0)	STARTING (0)	ACTIVE (n)	STOPPING (0)
init	>RESOLVED, >STARTING(0)	>STARTING (0)	noop	noop	noop
start	init, >RESOLVED, >STARTING (0), >ACTIVE (n)	init, >STARTING (0), >ACTIVE (n)	>ACTIVE (n)	noop	>RESOLVED (0), init, >STARTING (0), >ACTIVE (n)
stop	noop	noop	>STOPPING (0), >RESOLVED (0)	>STOPPING (0), >RESOLVED (0)	noop
update	noop	noop	>STOPPING (0), >RESOLVED (0), init, >STARTING (0),	>STOPPING (0), >RESOLVED (0), init, >STARTING (0), >ACTIVE (n)	noop

4.4.1 The init method

When init is called the framework bundle transitions to the STARTING state. No other bundles are active in the newly initialized framework. If a StartLevel service is available then the active start level value of the framework must be 0.

4.4.2 The start method

When the start method is called on a Framework bundle in the INSTALLED, RESOLVED or STARTING state then the Framework bundle transitions into the ACTIVE state. If the current Framework bundle state is INSTALLED or RESOLVED then init must be called before transitioning into the ACTIVE state. Calling this method on an ACTIVE Framework bundle has no effect. If a StartLevel service is available then the active start level value is incremented to the beginning start level.

4.4.3 The stop method

If the Framework bundle is in the STARTING or ACTIVE state then the Framework bundle transitions to the RESOLVED state on another non-deamon thread. If the StartLevel service is available then the active start level value is decremented to 0. After stop has been called the Framework bundle no longer has a valid BundleContext. All resources used by the framework must be released (e.g. All threads must be stopped, all files should be closed, all Bundle class loaders should be dereferenced by the framework and allowed to be garbage collected).

4.4.4 The update method

If the Framework bundle is in the STARTING or ACTIVE state then, on another non-daemon thread, Framework bundle transitions into the RESOLVED state and then transitions back to the original state (ACTIVE or STARTING). If the start level service is available then the active start level is decremented to 0 and then incremented back to the original active start level value.

Calling update on a Framework bundle in the INSTALLED or RESOLVED state has no effect.

The framework must invalidate the Framework bundle's BundleContext from the previous launch. A new BundleContext object must be obtained from Bundle.getBundleContext to interact with the relaunched framework.

4.4.5 Framework generated objects

Normal framework generated objects (other than the Framework Bundle) associated with an instance of the Framework are only valid as long as the Framework bundle is in the STARTING, ACTIVE or STOPPING state. After the Framework bundle has been stopped and has transitioned into the RESOVED state the Framework is allowed to discard all instances of the framework generated objects (e.g. Bundle, ServiceReference, etc.) and invalidate them. Any methods called on Bundle objects from a stopped Framework Bundle will have undefined behavior. If a Framework Bundle is started again then new Bundle instances must be obtained for any installed bundles.

4.5 Properties

The configurator can set the following properties. In addition, it can also add framework implementation dependent properties.

org.osgi.framework.version	set by framework implementation
org.osgi.framework.vendor	set by framework implementation
org.osgi.framework.language	set by configurator, but framework should provide a default when not set
org.osgi.framework. executionenvironment	set by configurator, but framework should provide a default when not set
org.osgi.framework.processor	set by configurator, but framework should provide a default when not set
org.osgi.framework.os.version	set by configurator, but framework should provide a default when not set
org.osgi.framework.os.name	set by configurator, but framework should provide a default when not set
org.osgi.supports. framework.extension	set by framework
org.osgi.supports. bootclasspath.extension	set by framework
org.osgi.supports.framework. fragment	set by framework to true
org.osgi.supports.framework. requirebundle	set by framework to true
org.osgi.framework. bootdelegation	set by configurator, framework provides empty default.
org.osgi.framework.system. packages	set by configurator, but framework should provide a default when not set

org.osgi.framework.system.packages.extra	Set by the configurator. Framework environment property identifying extra packages which the Framework bundle must export from the current execution environment. This property is useful for configuring extra system packages in addition to the system packages calculated by the framework.
org.osgi.framework.security	Specifies the the type of security manager the framework must use. If not specified then the framework will not set the VM security manager. The following types are defined ● osgi – Enables a security manager that supports all security aspects of the OSGi R4.0 specification (including postponed conditions). If specified and there is a security manager already installed then a SecurityException must be thrown when the Framework is initialized.
org.osgi.framework.storage	A valid file path in the file system to a directory. If the specified directory does not exist then the framework will create the directory. If the specified path exists but is not a directory or if the framework fails to create the storage directory then framework initialization must fail. The framework is free to use this directory as it sees fit. This area can not be shared with anything else. If this property is not set, it must use a reasonable platform default.
org.osgi.framework.storage.clean	Specifies if and when the storage area for the framework should be cleaned. Default value is none. The possible values are ● onFirstInit - the framework storage area will be cleaned before the Framework bundle is initialized for the first time. Subsequent inits, starts or updates of the Framework bundle will not result in cleaning the framework storage area. ● none - the framework storage area should not be cleaned.
org.osgi.framework.library.extensions	A comma separated list of additional library file extensions that must be searched for. If not set then only the library name returned by System.mapLibraryName(String) will be used to search. This is needed for certain operating systems which allow more than one extension for a library. For example AIX allows library extensions of .a and .so, but System.mapLibraryName(String) will only return names with the .a extension.
org.osgi.framework.command.execpermission	Specifies an optional OS specific command to set file permissions on extracted native code. On some operating systems it is required that native libraries be set to executable. This optional property allows you to specify the command. For example, on a UNIX style OS you could have the following value: org.osgi.framework.command.execpermission="chmod +rx ${abspath}" The ${abspath} is used to substitute the actual file path by the framework. [fullpath][fullpath]
org.osgi.framework.trust.repositories	This property is used to configure trust repositories for the framework. The value is a `java.io.File#pathSeparator` separated list of valid file paths in the file system to files that contain key stores of type JKS. The framework will use the key stores as trust repositories to authenticate certificates of trusted signers. The key stores are only used as read-only trust repositories to access public keys. No passwords are required to

	access the key stores' public keys.
org.osgi.framework.windowsystem	Set by the configurator but the framework should provide a reasonable default.
org.osgi.framework.startlevel	Specifies the beginning start level of the framework (See StartLevel service specification for more information).

4.6 Security

The Framework implementations typically require a broad scope of permissions in order to function. For example, the permission to create class loaders and set the security manager. In most cases Framework implementations require AllPermission to function properly.

Permissions granted to the Framework are controlled by the existing security manager, policy and class loader used to load the Framework implementation. Client code that constructs a Framework Bundle may have a more narrow set of permissions granted to it than the framework implementation. In this case the Framework Implementation should set a bundle's local permissions to be the intersection of permissions granted to the code which constructs the Framework and the permissions declared by the bundle's permission resource (OSGI-INF/permissions.perm). The permissions of the client code can be obtained when the Framework is constructed by saving the AccessControlContext in the Framework Constructor (e.g. `AccessController#`*`getContext`*`()`).

4.7 Starting Procedure

This RFC does not propose a proper shell script for the launch procedure. The Bundle object approach to framework creation is sufficient to allow almost any Java compatible script language to be used. An example of such a script language is the shell as proposed in this document (tsl), but any script language will work: Jython, Jruby, Beanshell, Bex, Groovy, etc.

For example in Groovy

```
 factoryClazz = "org.apache.felix.framework.FelixFactory"
 lib = "file:jar/felix.jar"

 ////// Generic
 storage = new File("osgi-storage").getAbsolutePath()
 storage.mkdirs()

 properties = [
 'org.osgi.framework.system.packages':"org.osgi.framework,\
    org.osgi.service.packageadmin, \
    org.osgi.service.startlevel,\
    javax.sql",
 'org.osgi.framework.storage' : storage.absolutePath,
'org.osgi.service.http.port'  : '8080'
 ]

 this.class.classLoader.rootLoader.addURL( new URL(lib) )

 clazz = Class.forName(factoryClazz)
 factory = clazz.newInstance()
 frameworkBundle = factory.newFramework( new Object[] {properties})
 frameworkBundle.init()
 ctx     = frameworkBundle.bundleContext
 servlet = ctx.installBundle("http://www.osgi.org/repository/servlet.jar")
 osgi    = ctx.installBundle("http://www.osgi.org/repository/osgi.jar")
 webrpc  = ctx.installBundle("http://www.aqute.biz/uploads/Code/aQute.webrpc.jar")
 suduko  = ctx.installBundle("http://www.aqute.biz/uploads/Code/aQute.sudoku.jar")
 http    = ctx.installBundle("http://www.knopflerfish.org/repo/jars/http/http_all-
2.1.0.jar")

 http.start()
 webrpc.start()
 suduko.start()

 frameworkBundle.start()
```

The shell language, outlined in RFC 147, can also be used in the same way. However, this is combined in a launcher program that gets the framework library and framework bundle class from the command line paramaters:

```
 org.osgi.framework.system.packages ="
       org.osgi.framework,
       org.osgi.service.packageadmin,
       org.osgi.service.startlevel,
       javax.sql"
 org.osgi.framework.storage = $user.home/osgi
 org.osgi.service.http.port = 8080

 start
 ctx = bundleContext
 addCommand ctx $ctx
 servlet = installBundle http://www.osgi.org/repository/servlet.jar
 osgi    = installBundle http://www.osgi.org/repository/osgi.jar
 webrpc  = installBundle http://www.aqute.biz/uploads/Code/aQute.webrpc.jar
 suduko  = installBundle http://www.aqute.biz/uploads/Code/aQute.sudoku.jar
 http    = installBundle http://www.knopflerfish.org/repo/jars/http/http_all-
2.1.0.jar
```

This design has the tremendous advantage that each organization can use its own script language. Due to the abstraction of the framework bundle, it is trivial to create a launcher that can be combined with any compliant framework. It is therefore not deemed wise to standardize the script syntax for the launch process, there are already a sufficient number around.

However, it is advantageous to take advantage of the command provider interface. This model is described in RFC 147, but it is based on services. Using the script language approach as defined in RFC 147, it is quite easy to search the service registry for new commands. The groovy meta model or the undefined command catch functions can make this quite transparent.

5 Alternatives

5.1 Considered setParentBundle

This section was denied because it was deemed to premature. An other RFC will look at nested frameworks.

setParentBundle(Bundle) – If a framework is embedded in another framework, then it must give the child framework the bundle object of its representation in the parent. That is, if you embed a framework in an OSGi framework, the parent is the bundle object of the code that manages the embedding. This bundle must be registered in the service registry as a Bundle service with the property: org.osgi.framework.parent=true. Singleton services like thread IO and URL handlers should use this service to synchronize their behavior with the ancestor frameworks. This method can be repeatedly called when the framework is not started.

5.2 Removed ThreadFactory and AsyncExecutor

A ThreadFactory and AsyncExecutor proposal was rejected. The design for this is archived at https://www.osgi.org/members/svn/build/trunk/archive/org.osgi.service.concurrent

6 Security Considerations

7 Document Support

7.1 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

7.1 Author’s Address

Name	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drezery
Voice	+33 633982260
e-mail	Peter.Kriens@aQute.biz

7.2 Acronyms and Abbreviations

7.3 End of Document