Framework Hooks

Final

41 Pages

Abstract

This RFC describes the means for a bundle to hook into the resolver, bundle events and accessing bundles with a bundle context.

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 11.1.

`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	08/02/10	At the F2F in Ottawa it was decided to remove the concept of composite from the core framework. This RFC now is used to specify a set of core framework hooks that can be used to build isolation models on top of the framework. Thomas Watson, IBM tjwatson@us.ibm.com
	08/18/10	– Bundle EventHook takes a collection of bundle context now – Bundle EventHook must be called once and only once per BundleEvent – The Bundle FindHook only affects the behavior of BundleContext.getBundles() not BundleContext.getBundle(long) – Describe dynamic resolution process. – Add begin/end to ResolverHook – Provide details on the resolve process and the interaction with the resolver hooks – Specify a failure when a resolver hook attempts to start a nested resolve process. – Imported javadoc.
	09/21/10	– Split ResolverHook into a factory and hook. Removed the restriction on the framework to have a single resovler process at a time.

Revision	Date	Comments
	09/22/10	– Minor updates to javadoc and use of triggered. – Added sections about exception handling – Add section that details how the framework obtains hook services (same as service hooks).
	10/07/10	– resolveBundles triggers only contains revisions for bundles in the INSTALLED state
	10/08/10	– Final version of RFC
	11/08/10	– Mark as Final to commence voting. BJ Hargrave

1 Introduction

This RFC details how a bundle can hook into the various operations with in the module and lifecycle layers of the core Framework to influence the resolve operations and influence access to bundles through bundle events and the bundle context.

2 Application Domain

This design is targeted at bundles which need to observe and manipulate select resolution operations and access to bundles through bundle events and the bundle context. In general these will be highly specialized bundles written by systems programmers. The design is not intended to be used by so-called “normal” application bundles.

3 Problem Description

One of the key features of the module layer is to perform the resolution process which “wires” requirements (Import-Package, Require-Bundle etc.) to capabilities (Export-Package, Bundle-SymbolicName/Bundle-Version etc.). The module layer provides no means for a bundle (external to the framework) to observe or manipulate the resolution process as it occurs. Certain specialized bundles need to be able to alter the output results of the resolution process. Such purposes may include things like bundle grouping or scoping for isolation, etc.

The lifecycle layer provides no means for a bundle (external to the framework) to observe or manipulate access to bundles. Certain specialized bundles need to be able to alter the output results of the lifecycle layer with respect to accessing bundle objects with the bundle event delivery operations and accessing bundle objects with the bundle context. Such purposes may include things like bundle grouping or scoping for isolation, etc.

4 Requirements

RFP 100 contains a number of requirements used for this RFC. Additional requirements have been added or reworded to address some of the issues that have been observed since the original RFP.

- The solution MUST work with the current lifecycle layer.
- The solution MUST work with the current module layer.
- The solution MUST allow certain bundles to reduce the list of candidates (exported packages) available to resolve import package constraint from a given bundle.
- The solution MUST allow certain bundles to reduce the list of candidates (bundles) available to resolve require bundle constraints from a given bundle.
- The solution MUST allow certain bundles to reduce the list of candidates (host bundles) available to resolve fragment host constraints from a given fragment.
- The solution MUST allow certain bundles to reduce the list of generic capability (RFC 154) candidates available to resolve generic requirements from a given bundle.
- The solution MUST allow multiple bundles with identical symbolic-name and version to be installed without error.
- The solution MUST allow certain bundles to reduce the list of singleton bundles which influence the resolvability of a given singleton bundle.

- The solution MUST allow certain bundles to reduce the list of bundles a bundle event is delivered to.
- The solution MUST allow certain bundles to reduce the list of bundles returned by a find bundle operation (BundleContext.getBundles()).
- The solution MUST be secured when java permissions are in effect.
- Security MUST not be used to provide the means to the solution. The solution MUST work without a security manager.

5 Technical Solution

The OSGi framework has built-in support for bundle resolution primitives and bundle lifecycle primitives. The resolution primitives are quite complex as well as powerful. The resolver built into the framework provides the basis for which the complete module layer is built upon. However, the resolver operates on information that is not completely visible to the bundles and in most cases bundles cannot effect the results of the resolver. For example, it is not possible for a bundle outside of the framework reduce the set of exported packages available to resolve a particular import package from a certain bundle.

Additionally, it is also not possible to allow bundles to influence the access to other bundle objects from a certain bundle. For example, when a find bundle operation is performed using a bundle context (getBundles() method) it is not possible to reduce the collection of bundles returned. Also, it is not possible to hide bundle events from certain bundles.

Therefore, this framework hook specification provides a number of new mechanisms that closely interact with the built-in resolver and lifecycle layer of the framework. These interactions are *not* intended for use by application bundles. Modifying the behavior of the module and lifecycle layer requires developers to closely follow the semantics of the OSGi module and lifecycle model and this is often hard, requiring a significant amount of code.

5.1 Terminology

- Client – The bundle that finds bundles or receives events about bundles. Also, the bundle which declares requirements which need to be resolved.
- Handler – The bundle that registers a hook service uses this to:
  - view or modify the resolution process
  - view or modify find bundle operations
  - view or modify bundle event delivery
- Capability – Some feature that is declared with meta-data. For example, Export-Package.

- Provider – A bundle that provides a capability
- Consumer – A client bundle that requires a capability
- Resolver – The internal framework machinery that wires (resolves) constraints declared by a consumer to capabilities declared by a provider.
- Bundle Event Hook – A bundle event hook intercepts bundle events before they are delivered to the client. The hook can select to remove events for specific bundles, which effectively allows the hook to hide bundle events from a bundle.
- Bundle Find Hook – A bundle find hook intercepts the getBundle(s) call just before it returns the results to the client. The results can be influenced by removing bundle entries. The bundle find hook can be used to hide specific bundles for specific bundles.
- Resolver Hook – A resolver hook intercepts the resolve process in the following ways:
  - Remove specific capabilities available to resolve a specific client bundle
  - Remove specific singleton bundles from influencing the resolvability of another specific singleton bundle.
  - Remove specific bundles from a list of resolvable bundles to prevent specific bundles from resolving during a single resolution process.

5.2 Synopsis

A bundle that needs to hide specific capabilities available to wire to from other bundles can register a resolver hook by registering a Resolver Hook service with the framework. If a resolve operation is performed it will pass this resolve hook information about the resolve process as it is occurring. The resolver hook can then inspect the arguments and optionally remove candidates available to a consumer to influence the resolver solution.

A bundle that needs to hide bundle objects from other bundles can register a bundle find or event hook by registering a Bundle Find Hook service or a Bundle Event Hook service with the framework. If a bundle event is generated, it will pass this event to the bundle event hook. The event hook method can then inspect the arguments and optionally remove bundles from the event delivery list.

When a bundle uses the Bundle Context getBundle(long) or getBundles method, the Bundle Find Hook is notified with a list of discovered bundles. The hook can then remove any bundles it wants to hide from the target bundle.

5.3 Bundle Event Hook

To intercept events being delivered to bundles, a handler must register a bundle EventHook object as a service with the framework. The framework must then send all bundle events to all registered hooks. The calling order of the hooks is defined by the reversed compareTo ordering of their Service Reference objects. That is, the service with the highest ranking number is called first. Bundle Event hooks are called after the event is generated but before they are delivered to any of the registered bundle listeners. Before the return, the handler can remove bundles from the given list. This allows a bundle event hook to hide bundle events for specific bundles.

An event hook receives all events, INSTALLED, RESOLVED, STARTING, STARTED, STOPPING, STOPPED, UNRESOLVED, UPDATED, and UNINSTALLED, if and only if there one or more bundle listeners registered with the framework.

The bundle EventHook class has a single method:

- event(BundleEvent, Collection<BundleContext>) – A bundle event has been generated. The implementer of this method can optionally shrink the given collection of target bundles.

One of the parameters of the event method is a collection of target bundle contexts. The handler can shrink this list by removing bundle contexts. The collection and its iterator must therefore implement the remove method.

Removing a bundle context from the list of target bundle contexts will effectively hide the bundle event from the target bundle context and therefore any bundle listeners registered with that bundle context. The target bundle context can still get the bundle for the event with the BundleContext.getBundle(s) methods, though the Bundle Find Hook can be used to block this access to the bundle.

The event method must be called one and only one time for each bundle event generated, this included bundle events which are generated when there are no bundle listeners registered. The event method must be called on the same thread that is performing the action which generated the specified event. The specified collection includes bundle contexts with synchronous and asynchronous bundle listeners registered with them.

Implementations of the Bundle Event Hook must ensure that target bundles continue to see a consistent set of bundle events. Bundle events can be used in a stat machine. Such state machines can get confused if some events are missed. For example, if a Bundle Tracker sees a STARTED event but is hidden from the corresponding STOPPED event then the tracker will still think the bundle is active. A simple solution is to stop the target bundle when the filter is put in place. However, when the bundle that implements the bundle event hook is stopped, it will of course no longer be able to filter events and the target bundle might see bundle events for bundles it was not aware of. As a best practice a bundle event hook should not hide an event from the bundle which the event is for.

5.4 Bundle Find Hook

The Bundle Find Hook is called when a target bundle searches the framework with the getBundles() methods on BundleContext. A registered Bundle Find Hook service gets a chance to inspect the returned set of bundles and can optionally shrink the set of returned bundles. The order in which the bundle find hooks are called is the reverse compareTo ordering of their Service References. That is, the service with the highest ranking number must be called first.

The bundle FindHook class has a single method:

- find(BundleContext, Collection<Bundle>) - The callback when a bundle calls the getBundle(long) or getBundles() methods on BundleContext. As parameters, it gets the bundle context of the calling bundle, and a set of bundles that will be returned. This list can be shortened by removing bundles from the given list.

The purpose of the Bundle Find Hook, is to limit the visibility of bundles to selected target bundles. For this reason the hook implementation can remove selected bundles from the result collection. The collection and its iterator must therefor implement the remove method. As a best practice a bundle find hook should not hide a bundle from itself.

5.5 Resolver Hook

A Resolver Hook Factory creates a resolver hook which is used by the framework for a single resolve process. The Resolver Hook is called during a resolution process. A Resolver Hook gets a chance to influence the outcome of a resolution process in the following ways.

- Limit the visibility of a capability to selected target bundles.
- Limit the effect a singleton bundle has on the resolvability of selected target bundles.
- Limit the set of bundles that will be available for resolution to a set of selected target bundles.

There are types of resolve processes that can be initiated.

1. A static bundle resolution operation. This resolve process is necessary any time one or more bundles transitions from the INSTALLED state to the RESOLVED state. During this resolve process the framework attempts to resolve static requirements specified by the bundles
2. A dynamic import resolution operation. This resolve process is necessary any time a request is made to wire a dynamic import for a bundle.

A resolver hook may influence the outcome of a resolve process by removing entries from shrinkable collections that are passed to the hook during a resolve process. A shrinkable collection is a collection that supports all remove operations. Any other attempts to modify a shrinkable collection will result in an UnsupportedOperationException being thrown.

5.5.1 Resolution Process Begins

When a resolve process begins the framework must ask each Resolver Hook Factory to provide a Resolver Hook which will be used for the duration of the resolve process. This is done by calling the begin method on a Resolver Hook Factory.

A resolver hook Factory is informed about a resolution process beginning with the following method:

- begin(Collection<BundleRevision> triggeres)

This method is called by the framework each time a resolve process begins to obtain a resolver hook instance. This resolver hook instance will be used for the duration of the resolve process. At the end of the resolve process the method ResolverHook.end() must be called by the framework and the framework must not hold any references of the resolver hook instance.

The triggers represent the collection of bundles which triggered the resolve process. This collection may be empty if the triggers cannot be determined by the framework. In most cases the triggers can easily be determined. Calling certain methods on bundle when a bundle is in the INSTALLED state will cause the framework to begin a resolve process in order to resolve the bundle. The following methods will start a resolve process in this case:

- start
- loadClass
- findEntries
- getResource
- getResources

In such cases the collection will contain the single bundle which the framework is trying to resolve. Other cases will cause multiple bundles to be included in the trigger bundles collection. When resolveBundles is called the collection of triggers must include all the current bundle revisions for bundles passed to resolveBundles which are in the INSTALLED state.

When FrameworkWiring.refreshBundles(Collection, org.osgi.framework.FrameworkListener...) is called the collection of triggers is determined with the following steps:

- If the collection of bundles passed is null then FrameworkWiring.getRemovalPendingBundles() is called to get the initial collection of bundles.
- The equivalent of calling FrameworkWiring.getDependencyClosure(Collection) is called with the initial collection of bundles to get the dependency closure collection of the bundles being refreshed.
- Remove any non-active bundles from the dependency closure collection.

For each bundle remaining in the dependency closure collection get the current bundle revision and add it to the collection of triggers.

The order in which the resolver hook factory begin methods are called is the reverse compareTo ordering of their Service References. That is, the service with the highest ranking number must be called first.

5.5.2 Resolution Process Ends

A resolver hook is informed about a resolution process ending with the following method:

- end() - This method is called once at the end of the resolve process. After the end method is called the resolve process has ended. The framework must not hold onto this resolver hook instance after end has been called.

The order in which the resolver hook end methods are called is the reverse compareTo ordering of their Service References. That is, the service with the highest ranking number must be called first.

5.5.3 Hide Capabilities

A Resolver Hook can hide capabilities using the following method:

- filterMatches(BundleRevision requirer, Collection<Capability> candidates) – A bundle (requirer) has a constraint which can be satisfied (matched) with the supplied collection of candidates (capabilities). The implementor of this method can optionally shrink the list of candidate capabilities.

One of the parameters of the filterMatches method is a list of capabilities. The handler can shrink this collection by removing capabilities. The collection and its iterator must therefore implement the remove method. Removing a capability from the list of candidates will effectively hide the capability from the target bundle. This will prevent the target bundle from getting wired to the capability.

The order in which the resolver hook filterMatches methods are called is the reverse compareTo ordering of their Service References. That is, the service with the highest ranking number must be called first.

5.5.4 Effect of Singleton Bundles

A Resolver Hook can change the effect of singleton bundles using the following method:

- filterSingletonCollisions(Capability singleton, Collection<Capability> collisionCandidates) – An osgi.bundle singleton capability has the same symbolic-name as the given collection of osgi.bundle capability candidates. The implementor of this method can optionally shrink the list of collision candidate capabilities.

One of the parameters of the filterSingletonCollisions method is a set of capabilities. The handler can shrink this collection by removing capabilities. The collection and its iterator must therefore implement the remove method. Removing a capability from the list of collision candidates will effectively hide the collision candidate from the target singleton bundle. This will allow the target singleton bundle to resolve regardless of the resolution state of the collision candidate.

The framework may call this method multiple times for the same singleton capability. For example, as a first pass a framework may want to determine if a singleton bundle is resolvable first based on the presence of other already resolved singleton capabilities. Later the framework may call this method again to determine which singleton capabilities from unresolved bundles to allow to resolve.

The order in which the resolver hook filterSingletonCollisions methods are called is the reverse compareTo ordering of their Service References. That is, the service with the highest ranking number must be called first.

5.5.5 Limit the Set of Resolvable Bundles

A Resolver hook can limit the set of bundles that will be allowed to resolve for a single resolve operation with the following method:

- filterResolvable(Collection<BundleRevision> candidates) – A resolve operation has been started. The given collection of candidate bundles are available to resolve (i.e. they are currently unresolved). The implementor of this method can optionally shrink the collection of candidate bundles.

The candidates parameters of the filterResolvable method is a collection of bundle revisions. The handler can shrink this collection by removing candidate bundle revisions. The collection and its iterator must therefore implement the remove method. Removing a candidate bundle from the collection of candidates will effectively prevent the bundle from resolving for the current resolve operation.

The order in which the resolver hook filterResolvable methods are called is the reverse compareTo ordering of their Service References. That is, the service with the highest ranking number must be called first.

For a dynamic import resolution process it is acceptable for the framework to pass an empty collection of resolvable candidates. This indicates that the framework will not cause any bundles to transition from INSTALLED to RESOLVED during a dynamic import package resolution.

5.5.6 The Resolve Process

The following steps outline the way a framework uses the resolver hooks during a resolve process.

1. Collect a snapshot of registered resolver hook factories that will be called during the current resolve process. Any hook factories registered after the snapshot is taken must not be called during the current resolve process. A resolver hook factory contained in the snapshot may become unregistered during the resolve process. The framework should handle this and stop calling the resolver hook instance provided by the unregistered hook factory for the remainder of the resolve process. Each registered resolver hook factory service in the snapshot will be obtained by the framework (by calling BundleContext.getService method using the system bundle context).

2. For each registered hook factory, call the begin() method to inform the hooks about a resolve process beginning and to obtain a Resolver Hook instance that will be used for the duration of the resolve process.

3. Determine the collection of unresolved bundle revisions that may be considered for resolution during the current resolution process and place each of the bundle revisions in a shrinkable collection **R**.

   a) For each resolver hook, call the filterResolveable method with the shrinkable collection **R**.

4. The shrinkable collection **R** now contains all the unresolved bundle revisions that may end up as resolved at the end of the current resolve process. Any other bundle revisions that got removed from the shrinkable collection **R** must not end up as resolved at the end of the current resolve process.

5. For each bundle revision **B** left in the shrinkable collection **R** that represents a singleton bundle, do the following:

   a) Determine the collection of available capabilities that have a name space of osgi.bundle, are singletons, and have the same symbolic name as the singleton bundle revision **B** and place each of the matching capabilities into a shrinkable collection **S**.

   b) Remove the osgi.bundle capability provided by the bundle revision **B** from the shrinkable collection **S**. A singleton bundle cannot collide with itself.

   c) For each resolver hook call the filterSingletonCollisions method with the osgi.bundle capability provided by bundle revision **B** and the shrinkable collection **S**.

   d) The shrinkable collection **S** now contains all the singleton osgi.bundle capabilities that can influence the ability of bundle revision **B** to resolve.

6. During a resolve process a framework is free to attempt to resolve any or all bundles contained in the shrinkable collection **R**. For each bundle revision **B** left in the shrinkable collection **R** which the framework attempts to resolve the following steps must be followed:

   a) For each requirement **T** specified by bundle revision **B,** determine the collection of capabilities that satisfy (or match) the requirement and place each matching capability into a shrinkable collection C. A capability is considered to match a particular requirement if its attributes satisfy a specified requirement and the requirer bundle has permission to access the capability.

   b) For each resolver hook, call the filterMatches method with the bundle revision **B** and the shrinkable collection **C**.

   c) The shrinkable collection **C** now contains all the capabilities that may be used to satisfy the requirement **T**. Any other capabilities that got removed from the shrinkable collection **C** must not be used to satisfy requirement **T.**

7. For each resolver hook, call the end method to inform the hook about a resolve process ending.

8. For each hook factory obtained by the framework in the first step; unget the service instance (by calling BundleContext.ungetService using the system bundle context).

In cases where the a shrinkable collection becomes empty the framework is required to call the remaining hooks.

The above steps are meant to illustrate how the resolve hooks are used by the framework. They are not normative. The nature of the resolve process and the resolve algorithm can require some back tracking by the framework implementation. Because of this it is acceptable for the framework to call methods on the ResolverHook multiple times with similar or identical parameters during a single resolve process. This is true for all methods except the begin and end methods. The begin and end methods must be called once and only once for each resolve process.

Resolver hooks are low level. Implementations of the resolver hook must be careful not to create an unresolvable state which is very hard for a developer or a provisioner to diagnose. Resolver hooks also must not be allowed to start another resolve process (e.g. by calling Bundle.start() or FrameworkWiring.resolveBundles). The framework must detect this and throw an IllegalStateException from the resolve process. In cases where a BundleException can be thrown (e.g. Bundle.start()) the IllegalStateException must be the cause of the BundleException and the BundleException must be of type BundleException.RESOLVE_ERROR. In cases where an exception cannot be propagated to a caller (e.g. dynamic import resolution) a FrameworkEvent of type ERROR must be published.

5.6 Ordinary Services

All hooks are treated as ordinary services. If the framework uses them, their Service References will show that the system bundle is using them, and if a hook is a Service Factory, then the actual instance will be properly created.

The only speciality of the service hooks (defined in chapter 12 of R4.2 specification) is that the framework must not use them for the other hooks themselves. That is, the service Event and service Find Hooks can not be used to hide the other hook services from the framework.

5.7 Exception handling

If a hook implementation method throws an exception the framework must publish a FrameworkEvent of type error and continue to call the remaining hooks for the method being called. The framework may stop calling the hook which threw the exception for future operations but the framework is not obligated to.

5.8 Bundle Symbolic Name and Version

In R4.0 the concept of a bundle symbolic name and version were formalized. The unique content of a bundle can be identified by its unique bundle symbolic name and version. The R4.0 specification also caused an installation exception if the same bundle (with identical bundle symbolic name and version) was installed multiple times. This restriction is being lifted in this release to allow the same bundle to be installed multiple times (using unique location strings).

A new framework configuration option (org.osgi.framework.bsnversion) has been added to enable this feature. The values of this property can be “multiple” or “single”. The value “multiple” specifies the framework will allow multiple bundles to be installed having the same symbolic name and version. The value “single” specifies the framework will only allow a single bundle to be installed for a given symbolic name and version. It will be an error to install a bundle or update a bundle to have the same symbolic name and version as another installed bundle. The default value is “single”. Note that this default may change in a future specification.

6 Java Doc

OSGi Javadoc

10/8/10 11:17 AM

Package Summary	***Page***

Package org.osgi.framework.hooks.bundle

Framework Bundle Hooks Package Version 1.0.

See:
Description

Interface Summary		Page
EventHook	OSGi Framework Bundle Event Hook Service.	*16*
FindHook	OSGi Framework Bundle Context Hook Service.	*17*

Package org.osgi.framework.hooks.bundle Description

Framework Bundle Hooks Package Version 1.0.

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.bundle; version="[1.0,2.0)"
```

Interface EventHook

org.osgi.framework.hooks.bundle

`public interface` **`EventHook`**

OSGi Framework Bundle Event Hook Service.

Bundles registering this service will be called during framework lifecycle (install, start, stop, update, and uninstall bundle) operations.

Version:
$Id: 18ea1ec1f14f47410a43e99be4da3b2583149722 $
ThreadSafe

Method Summary		*Page*
`void`	**`event`**`(org.osgi.framework.BundleEvent event, Collection<org.osgi.framework.BundleContext> contexts)` Bundle event hook method.	*16*

Method Detail

event

```
void event(org.osgi.framework.BundleEvent event,
           Collection<org.osgi.framework.BundleContext> contexts)
```

Bundle event hook method. This method is called prior to bundle event delivery when a bundle is installed, resolved, started, stopped, unresolved, or uninstalled. This method can filter the bundles which receive the event.

This method must be called by the framework one and only one time for each bundle event generated, this included bundle events which are generated when there are no bundle listeners registered. This method must be called on the same thread that is performing the action which generated the specified event. The specified collection includes bundle contexts with synchronous and asynchronous bundle listeners registered with them.

Parameters:

`event` - The bundle event to be delivered

`contexts` - A collection of Bundle Contexts for bundles which have listeners to which the specified event will be delivered. The implementation of this method may remove bundle contexts from the collection to prevent the event from being delivered to the associated bundles. The collection supports all the optional `Collection` operations except `add` and `addAll`. Attempting to add to the collection will result in an `UnsupportedOperationException`. The collection is not synchronized.

Interface FindHook

org.osgi.framework.hooks.bundle

```
public interface FindHook
```

OSGi Framework Bundle Context Hook Service.

Bundles registering this service will be called during framework bundle find (get bundles) operations.

Version:
$Id: 2f5bd0fa306e792eb88f254975e301b20d7f969d $
ThreadSafe

Method Summary		*Page*
`void`	**`find`**`(org.osgi.framework.BundleContext context, Collection<org.osgi.framework.Bundle> bundles)` Find hook method.	*17*

Method Detail

find

```
void find(org.osgi.framework.BundleContext context,
          Collection<org.osgi.framework.Bundle> bundles)
```

Find hook method. This method is called during the bundle find operation (for example, `getBundle` and `org.osgi.framework.BundleContext.getBundles()` methods). This method can filter the result of the find operation.

Parameters:

`context` - The bundle context of the bundle performing the find operation.
`bundles` - A collection of Bundles to be returned as a result of the find operation. The implementation of this method may remove bundles from the collection to prevent the bundles from being returned to the bundle performing the find operation. The collection supports all the optional `Collection` operations except `add` and `addAll`. Attempting to add to the collection will result in an `UnsupportedOperationException`. The collection is not synchronized.

Package org.osgi.framework.hooks.resolver

Framework Resolver Hooks Package Version 1.0.

See:
  Description

Interface Summary		***Page***
ResolverHook	OSGi Framework Resolver Hook instances are obtained from the OSGi `Framework Resolver Hook Factory` service.	*19*
ResolverHookFactory	OSGi Framework Resolver Hook Factory Service.	*22*

Package org.osgi.framework.hooks.resolver Description

Framework Resolver Hooks Package Version 1.0.

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.resolver; version="[1.0,2.0)"
```

Interface ResolverHook

org.osgi.framework.hooks.resolver

`public interface` **`ResolverHook`**

OSGi Framework Resolver Hook instances are obtained from the OSGi `Framework Resolver Hook Factory` service.

A Resolver Hook instance is called by the framework during a resolve process. A resolver hook may influence the outcome of a resolve process by removing entries from shrinkable collections that are passed to the hook during a resolve process. A shrinkable collection is a `Collection` that supports all remove operations. Any other attempts to modify a shrinkable collection will result in an `UnsupportedOperationException` being thrown.

The following steps outline the way a framework uses the resolver hooks during a resolve process.

1. Collect a snapshot of registered resolver hook factories that will be called during the current resolve process. Any hook factories registered after the snapshot is taken must not be called during the current resolve process. A resolver hook factory contained in the snapshot may become unregistered during the resolve process. The framework should handle this and stop calling the resolver hook instance provided by the unregistered hook factory for the remainder of the resolve process.
2. For each registered hook factory call the `ResolverHookFactory.begin(Collection)` method to inform the hooks about a resolve process beginning and to obtain a Resolver Hook instance that will be used for the duration of the resolve process.
3. Determine the collection of unresolved bundle revisions that may be considered for resolution during the current resolution process and place each of the bundle revisions in a shrinkable collection **`R`**.
    a. For each resolver hook call the `filterResolvable(Collection)` method with the shrinkable collection **`R`**.
4. The shrinkable collection **`R`** now contains all the unresolved bundle revisions that may end up as resolved at the end of the current resolve process. Any other bundle revisions that got removed from the shrinkable collection **`R`** must not end up as resolved at the end of the current resolve process.
5. For each bundle revision `B` left in the shrinkable collection **`R`** that represents a singleton bundle do the following:
    a. Determine the collection of available capabilities that have a name space of `osgi.bundle`, are singletons, and have the same symbolic name as the singleton bundle revision **`B`** and place each of the matching capabilities into a shrinkable collection **`S`**.
    b. Remove the `osgi.bundle` capability provided by bundle revision **`B`** from shrinkable collection **`S`**. A singleton bundle cannot collide with itself.
    c. For each resovler hook call the `filterSingletonCollisions(Capability, Collection)` with the `osgi.bundle` capability provided by bundle revision **`B`** and the shrinkable collection **`S`**
    d. The shrinkable collection **`S`** now contains all singleton `osgi.bundle` capabilities that can influence the ability of bundle revision **`B`** to resolve.
6. During a resolve process a framework is free to attempt to resolve any or all bundles contained in shrinkable collection **`R`**. For each bundle revision **`B`** left in the shrinkable collection **`R`** which the framework attempts to resolve the following steps must be followed:
    a. For each requirement **`T`** specified by bundle revision **`B`** determine the collection of capabilities that satisfy (or match) the requirement and place each matching capability into a shrinkable collection **`C`**. A capability is considered to match a particular requirement if its attributes satisfy a specified requirement and the requirer bundle has permission to access the capability.
    b. For each resolver hook call the `filterMatches(BundleRevision, Collection)` with the bundle revision **`B`** and the shrinkable collection **`C`**.
    c. The shrinkable collection **`C`** now contains all the capabilities that may be used to satisfy the requirement **`T`**. Any other capabilities that got removed from the shrinkable collection **`C`** must not be used to satisfy requirement **`T`**.
7. For each resolver hook call the `end()` method to inform the hooks about a resolve process ending.

In all cases, the order in which the resolver hooks are called is the reverse compareTo ordering of their Service References. That is, the service with the highest ranking number must be called first. In cases where a shrinkable collection becomes empty the framework is required to call the remaining registered hooks.

Resolver hooks are low level. Implementations of the resolver hook must be careful not to create an unresolvable state which is very hard for a developer or a provisioner to diagnose. Resolver hooks also must not be allowed to start another synchronous resolve process (e.g. by calling `org.osgi.framework.Bundle.start()` or

`FrameworkWiring.resolveBundles(Collection)`). The framework must detect this and throw an `IllegalStateException`.

Version:
$Id: 49bcead9ff37ad2c864c0b0f2ca510f0be04f835 $

See Also:
`ResolverHookFactory`

ThreadSafe

Method Summary		Page
void	`end()` This method is called once at the end of the resolve process.	21
void	`filterMatches(BundleRevision requirer, Collection<Capability> candidates)` Filter matches hook method.	21
void	`filterResolvable(Collection<BundleRevision> candidates)` Filter resolvable candidates hook method.	20
void	`filterSingletonCollisions(Capability singleton, Collection<Capability> collisionCandidates)` Filter singleton collisions hook method.	20

Method Detail

filterResolvable

```
void filterResolvable(Collection<BundleRevision> candidates)
```

Filter resolvable candidates hook method. This method may be called multiple times during a single resolve process. This method can filter the collection of candidates by removing potential candidates. Removing a candidate will prevent the candidate from resolving during the current resolve process.

Parameters:
`candidates` - the collection of resolvable candidates available during a resolve process.

filterSingletonCollisions

```
void filterSingletonCollisions(Capability singleton,
                               Collection<Capability> collisionCandidates)
```

Filter singleton collisions hook method. This method is called during the resolve process for the specified singleton. The specified singleton represents a singleton capability and the specified collection represent a collection of singleton capabilities which are considered collision candidates. The singleton capability and the collection of collision candidates must all use the same name space.

Currently only capabilities with the name space of `osgi.bundle` can be singletons. In that case all the collision candidates have the name space of `osgi.bundle`, are singletons, and have the same symbolic name as the specified singleton capability.

In the future, capabilities in other name spaces may support the singleton concept. Hook implementations should be prepared to receive calls to this method for capabilities in name spaces other than `osgi.bundle`.

This method can filter the list of collision candidates by removing potential collisions. Removing a collision candidate will allow the specified singleton to resolve regardless of the resolution state of the removed collision candidate.

Parameters:
`singleton` - the singleton involved in a resolve process
`collisionCandidates` - a collection of singleton collision candidates

filterMatches

```
void filterMatches(BundleRevision requirer,
                   Collection<Capability> candidates)
```

Filter matches hook method. This method is called during the resolve process for the specified requirer. The collection of candidates match a requirement for the requirer. This method can filter the collection of matching candidates by removing candidates from the collection. Removing a candidate will prevent the resolve process from choosing the removed candidate to satisfy a requirement for the requirer.

All of the candidates will have the same name space and will match a requirement of the requirer.

If the Java Runtime Environment supports permissions then the collection of candidates will only contain candidates for which the requirer has permission to access.

Parameters:

`requirer` - the bundle revision which contains a requirement
`candidates` - a collection of candidates that match a requirement of the requirer

end

```
void end()
```

This method is called once at the end of the resolve process. After the end method is called the resolve process has ended. The framework must not hold onto this resolver hook instance after end has been called.

Interface ResolverHookFactory

org.osgi.framework.hooks.resolver

`public interface` **`ResolverHookFactory`**

OSGi Framework Resolver Hook Factory Service.

Bundles registering this service will be called by the framework during a bundle resolver process to obtain a `resolver hook` instance which will be used for the duration of a resolve process.

Version:
$Id: 08a6fba0f95499d08047282124b88256d33e1ce4 $
See Also:
`ResolverHook`
ThreadSafe

Method Summary		Page
`ResolverHook`	**`begin`**`(Collection<BundleRevision> triggers)` This method is called by the framework each time a resolve process begins to obtain a `resolver hook` instance.	*22*

Method Detail

begin

`ResolverHook` **`begin`**`(Collection<BundleRevision> triggers)`

This method is called by the framework each time a resolve process begins to obtain a `resolver hook` instance. This resolver hook instance will be used for the duration of the resolve process. At the end of the resolve process the method `ResolverHook.end()` must be called by the framework and the framework must not hold any references of the resolver hook instance.

The triggers represent the collection of bundles which triggered the resolve process. This collection may be empty if the triggers cannot be determined by the framework. In most cases the triggers can easily be determined. Calling certain methods on `bundle` when a bundle is in the `INSTALLED` state will cause the framework to begin a resolve process in order to resolve the bundle. The following methods will start a resolve process in this case:

- `start`
- `loadClass`
- `findEntries`
- `getResource`
- `getResources`

In such cases the collection will contain the single bundle which the framework is trying to resolve. Other cases will cause multiple bundles to be included in the trigger bundles collection. When `resolveBundles` is called the collection of triggers must include all the current bundle revisions for bundles passed to resolveBundles which are in the `INSTALLED` state.

When `FrameworkWiring.refreshBundles(Collection, org.osgi.framework.FrameworkListener...)` is called the collection of triggers is determined with the following steps:

- If the collection of bundles passed is null then `FrameworkWiring.getRemovalPendingBundles()` is called to get the initial collection of bundles.
- The equivalent of calling `FrameworkWiring.getDependencyClosure(Collection)` is called with the initial collection of bundles to get the dependency closure collection of the bundles being refreshed.

- Remove any non-active bundles from the dependency closure collection.
- For each bundle remaining in the dependency closure collection get the current bundle revision and add it to the collection of triggers.

Parameters:

`triggers` - an unmodifiable collection of bundles which triggered the resolve process. This collection may be empty if the collection of trigger bundles cannot be determined.

Returns:

a resolver hook instance to be used for the duration of the resolve process. A `null` value may be returned which indicates this resolver hook factory abstains from the resolve process.

Package org.osgi.framework.wiring

Framework Wiring Package Version 1.0.

See:
Description

Interface Summary		*Page*
BundleRevision	Bundle Revision.	*25*
BundleWiring	A wiring for a bundle.	*27*
BundleWirings	The `in use` bundle wirings for a bundle.	*32*
Capability	A capability that has been declared from a `bundle revision`.	*33*
FrameworkWiring	Query and modify wiring information for the framework.	*36*
WiredCapability	A wired capability that has been provided from a `bundle wiring`.	*39*

Package org.osgi.framework.wiring Description

Framework Wiring Package Version 1.0.

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

```
 Import-Package: org.osgi.framework.wiring; version="[1.0,2.0)"
```

Interface BundleRevision

org.osgi.framework.wiring

All Superinterfaces:
org.osgi.framework.BundleReference

```
public interface BundleRevision
extends org.osgi.framework.BundleReference
```

Bundle Revision. Since a bundle update can change the entries in a bundle, different bundle wirings for the same bundle can be associated with different bundle revisions.

The current bundle revision for a bundle can be obtained by calling `bundle.adapt`(BundleRevision.class).

Version:
$Id: 1a15861159e7071a1eb504afbc3f75f8af1aff01 $
ThreadSafe

Field Summary		**Page**
`int`	**`TYPE_FRAGMENT`** Bundle revision type indicating the bundle revision is a fragment.	25

Method Summary		**Page**
`List<Capability>`	**`getDeclaredCapabilities`**`(String namespace)` Returns the capabilities declared by this bundle revision.	26
`String`	**`getSymbolicName`**`()` Returns the symbolic name for this bundle revision.	25
`int`	**`getTypes`**`()` Returns the special types of this bundle revision.	26
`org.osgi.framework.Version`	**`getVersion`**`()` Returns the version for this bundle revision.	26

Methods inherited from interface org.osgi.framework.BundleReference
`getBundle`

Field Detail

TYPE_FRAGMENT

```
public static final int TYPE_FRAGMENT = 1
```

Bundle revision type indicating the bundle revision is a fragment.

See Also:
`getTypes()`

Method Detail

getSymbolicName

```
String getSymbolicName()
```

Returns the symbolic name for this bundle revision.

Returns:
The symbolic name for this bundle revision.
See Also:
`org.osgi.framework.Bundle.getSymbolicName()`

---

getVersion

`org.osgi.framework.Version` **`getVersion`**`()`

Returns the version for this bundle revision.

Returns:
The version for this bundle revision, or `org.osgi.framework.Version.emptyVersion` if this bundle revision has no version information.
See Also:
`org.osgi.framework.Bundle.getVersion()`

---

getDeclaredCapabilities

`List<Capability>` **`getDeclaredCapabilities`**`(String namespace)`

Returns the capabilities declared by this bundle revision.

Parameters:
`namespace` - The name space of the declared capabilities to return or `null` to return the provided capabilities from all name spaces.
Returns:
A list containing a snapshot of the declared `Capability`s, or an empty list if this bundle revision declares no capabilities in the specified name space. The list contains the provided capabilities in the order they are specified in the manifest.

---

getTypes

`int` **`getTypes`**`()`

Returns the special types of this bundle revision. The bundle revision type values are:

- `TYPE_FRAGMENT`

A bundle revision may be more than one type at a time. A type code is used to identify the bundle revision type for future extendability.

If this bundle revision is not one or more of the defined types then 0 is returned.

Returns:
The special types of this bundle revision. The type values are ORed together.

Interface BundleWiring

org.osgi.framework.wiring

All Superinterfaces:
org.osgi.framework.BundleReference

```
public interface BundleWiring
extends org.osgi.framework.BundleReference
```

A wiring for a bundle. Each time a bundle is resolved, a new bundle wiring for the bundle is created. A bundle wiring consists of a bundle and it attached fragments and represents the dependencies with other bundle wirings.

The bundle wiring for a bundle is the `current` bundle wiring if the bundle is resolved and the bundle wiring is the most recent bundle wiring. All bundles with non-current, in use bundle wirings are considered removal pending. A bundle wiring is `in use` if it is the current wiring or if some other in use bundle wiring is dependent upon it. For example, wired to a package exported by the bundle wiring or requires the bundle wiring. An in use bundle wiring has a class loader. Once a bundle wiring is no longer in use, it is considered stale and is discarded by the framework.

A list of all in use bundle wirings for a bundle can be obtained by calling `bundle.adapt(BundleWirings.class).getWirings()`. For non-fragment bundles, the first item in the returned list is the current bundle wiring.

The current bundle wiring for a non-fragment bundle can be obtained by calling `bundle.adapt`(BundleWiring.class). A fragment bundle does not itself have bundle wirings. So calling `bundle.adapt`(BundleWiring.class) on a fragment must return `null`.

Version:
$Id: 6787c55f83f520429d63e93024ccf9f22821c4c8 $
ThreadSafe

Field Summary		Page
int	**FINDENTRIES_RECURSE** The find entries operation must recurse into subdirectories.	28
int	**LISTRESOURCES_LOCAL** The list resource names operation must limit the result to the names of matching resources contained in this bundle wiring's `bundle revision` and its attached `fragment revisions`.	28
int	**LISTRESOURCES_RECURSE** The list resource names operation must recurse into subdirectories.	28

Method Summary		Page
List<URL>	**findEntries**(String path, String filePattern, int options) Returns entries in this bundle wiring's `bundle revision` and its attached `fragment revisions`.	30
BundleRevision	**getBundleRevision**() Returns the bundle revision for the bundle in this bundle wiring.	30
ClassLoader	**getClassLoader**() Returns the class loader for this bundle wiring.	30
List<BundleRevision>	**getFragmentRevisions**() Returns the bundle revisions for all attached fragments of this bundle wiring.	30
List<WiredCapability>	**getProvidedCapabilities**(String capabilityNamespace) Returns the capabilities provided by this bundle wiring.	29
List<WiredCapability>	**getRequiredCapabilities**(String capabilityNamespace) Returns the required capabilities used by this bundle wiring.	29

boolean	**isCurrent**() Returns `true` if this bundle wiring is the current bundle wiring.	*29*
boolean	**isInUse**() Returns `true` if this bundle wiring is in use.	*29*
List<String>	**listResources**(String path, String filePattern, int options) Returns the names of resources visible to this bundle wiring's class loader.	*31*

Methods inherited from interface org.osgi.framework.BundleReference
getBundle

Field Detail

FINDENTRIES_RECURSE

`public static final int` **`FINDENTRIES_RECURSE`** `= 1`

The find entries operation must recurse into subdirectories.

This bit may be set when calling `findEntries(String, String, int)` to specify the result must include the matching entries from the specified path and its subdirectories. If this bit is not set, then the result must only include matching entries from the specified path.

See Also:
`findEntries(String, String, int)`

LISTRESOURCES_RECURSE

`public static final int` **`LISTRESOURCES_RECURSE`** `= 1`

The list resource names operation must recurse into subdirectories.

This bit may be set when calling `listResources(String, String, int)` to specify the result must include the names of matching resources from the specified path and its subdirectories. If this bit is not set, then the result must only include names of matching resources from the specified path.

See Also:
`listResources(String, String, int)`

LISTRESOURCES_LOCAL

`public static final int` **`LISTRESOURCES_LOCAL`** `= 2`

The list resource names operation must limit the result to the names of matching resources contained in this bundle wiring's `bundle revision` and its attached `fragment revisions`.

This bit may be set when calling `listResources(String, String, int)` to specify the result must only include the names of matching resources contained in this bundle wiring's bundle revision and its attached fragment revisions. If this bit is not set, then the result must include the names of matching resources reachable from this bundle wiring's class loader which may include the names of matching resources contained in imported packages and required bundles.

See Also:
`listResources(String, String, int)`

Method Detail

isCurrent

`boolean **isCurrent**()`

Returns `true` if this bundle wiring is the current bundle wiring. The bundle wiring for a bundle is the current bundle wiring if the bundle is resolved and the bundle wiring is the most recent bundle wiring. All bundles with non-current, in use bundle wirings are considered `removal pending`.

Returns:
`true` if this bundle wiring is the current bundle wiring; `false` otherwise.

---

isInUse

`boolean **isInUse**()`

Returns `true` if this bundle wiring is in use. A bundle wiring is in use if it is the `current` wiring or if some other in use bundle wiring is dependent upon it. Once a bundle wiring is no longer in use, it is considered stale and is discarded by the framework.

Returns:
`true` if this bundle wiring is in use; `false` otherwise.

---

getProvidedCapabilities

`List<WiredCapability> **getProvidedCapabilities**(String capabilityNamespace)`

Returns the capabilities provided by this bundle wiring.

Parameters:
`capabilityNamespace` - The name space of the provided capabilities to return or `null` to return the provided capabilities from all name spaces.
Returns:
A list containing a snapshot of the `WiredCapability`s, or an empty list if this bundle wiring provides no capabilities in the specified name space. If this bundle wiring is not `in use`, `null` will be returned. The list contains the provided capabilities in the order they are specified in the manifest.

---

getRequiredCapabilities

`List<WiredCapability> **getRequiredCapabilities**(String capabilityNamespace)`

Returns the required capabilities used by this bundle wiring.

The result of this method can change if this bundle wiring requires additional capabilities.

Parameters:
`capabilityNamespace` - The name space of the required capabilities to return or `null` to return the required capabilities from all name spaces.
Returns:
A list containing a snapshot of the `WiredCapability`s used by this bundle wiring, or an empty list if this bundle wiring requires no capabilities in the specified name space. If this bundle wiring is not `in use`, `null` will be returned. The list contains the required capabilities in the order they are specified in the manifest.

---

getBundleRevision

`BundleRevision` **`getBundleRevision`**`()`

Returns the bundle revision for the bundle in this bundle wiring. Since a bundle update can change the entries in a bundle, different bundle wirings for the same bundle can have different bundle revisions.

The bundle object `referenced` by the returned `BundleRevision` may return different information than the returned `BundleRevision` since the returned `BundleRevision` may refer to an older revision of the bundle.

Returns:
The bundle revision for this bundle wiring.

---

getFragmentRevisions

`List<BundleRevision>` **`getFragmentRevisions`**`()`

Returns the bundle revisions for all attached fragments of this bundle wiring. Since a bundle update can change the entries in a fragment, different bundle wirings for the same bundle can have different bundle revisions.

The bundle revisions in the list are ordered in fragment attachment order such that the first revision in the list is the first attached fragment and the last revision in the list is the last attached fragment.

Returns:
A list containing a snapshot of the `BundleRevision`s for all attached fragments attached of this bundle wiring, or an empty list if this bundle wiring does not have any attached fragments. If this bundle wiring is not `in use` , `null` will be returned.

---

getClassLoader

`ClassLoader` **`getClassLoader`**`()`

Returns the class loader for this bundle wiring. Since a bundle refresh creates a new bundle wiring for a bundle, different bundle wirings for the same bundle will have different class loaders.

Returns:
The class loader for this bundle wiring. If this bundle wiring is not `in use`, `null` will be returned.
Throws:
`SecurityException` - If the caller does not have the appropriate `RuntimePermission("getClassLoader")`, and the Java Runtime Environment supports permissions.

---

findEntries

```
List<URL> findEntries(String path,
                      String filePattern,
                      int options)
```

Returns entries in this bundle wiring's `bundle revision` and its attached `fragment revisions`. This bundle wiring's class loader is not used to search for entries. Only the contents of this bundle wiring's bundle revision and its attached fragment revisions are searched for the specified entries.

This method takes into account that the "contents" of this bundle wiring can have attached fragments. This "bundle space" is not a namespace with unique members; the same entry name can be present multiple times. This method therefore returns a list of URL objects. These URLs can come from different JARs but have the same path name. This method can either return only entries in the specified path or recurse into subdirectories returning entries in the directory tree beginning at the specified path.

Note: Jar and zip files are not required to include directory entries. URLs to directory entries will not be returned if the bundle contents do not contain directory entries.

Parameters:

`path` - The path name in which to look. The path is always relative to the root of this bundle wiring and may begin with "/". A path value of "/" indicates the root of this bundle wiring.
`filePattern` - The file name pattern for selecting entries in the specified path. The pattern is only matched against the last element of the entry path. If the entry is a directory then the trailing "/" is not used for pattern matching. Substring matching is supported, as specified in the Filter specification, using the wildcard character ("*"). If `null` is specified, this is equivalent to "*" and matches all files.
`options` - The options for listing resource names. See `FINDENTRIES_RECURSE`. The method must ignore unrecognized options.

Returns:

An unmodifiable list of URL objects for each matching entry, or an empty list if no matching entry could not be found or if the caller does not have the appropriate `AdminPermission[bundle,RESOURCE]` and the Java Runtime Environment supports permissions. The list is ordered such that entries from the `bundle revision` are returned first followed by the entries from `attached fragment revisions` in attachment order. If this bundle wiring is not `in use`, `null` will be returned.

See Also:

`org.osgi.framework.Bundle.findEntries(String, String, boolean)`

---

listResources

```
List<String> listResources(String path,
                           String filePattern,
                           int options)
```

Returns the names of resources visible to this bundle wiring's `class loader`. The returned names can be used to access the resources via this bundle wiring's class loader.

Parameters:

`path` - The path name in which to look. The path is always relative to the root of this bundle wiring's class loader and may begin with "/". A path value of "/" indicates the root of this bundle wiring's class loader.
`filePattern` - The file name pattern for selecting resource names in the specified path. The pattern is only matched against the last element of the resource path. If the resource is a directory then the trailing "/" is not used for pattern matching. Substring matching is supported, as specified in the Filter specification, using the wildcard character ("*"). If `null` is specified, this is equivalent to "*" and matches all files.
`options` - The options for listing resource names. See `LISTRESOURCES_LOCAL` and `LISTRESOURCES_RECURSE`. The method must ignore unrecognized options.

Returns:

An unmodifiable list of resource names for each matching resource, or an empty list if no matching resource could not be found or if the caller does not have the appropriate `AdminPermission[bundle,RESOURCE]` and the Java Runtime Environment supports permissions. The list is ordered such that resource names from this bundle are returned in the order they are visible in this bundle wiring's class loader. If this bundle wiring is not `in use`, `null` will be returned.

Interface BundleWirings

org.osgi.framework.wiring

All Superinterfaces:
org.osgi.framework.BundleReference

```
public interface BundleWirings
extends org.osgi.framework.BundleReference
```

The `in use` bundle wirings for a bundle. Each time a bundle is resolved, a new bundle wiring of the bundle is created. A bundle wiring consists of a bundle and its attached fragments and represents the dependencies with other bundle wirings.

The in use bundle wirings for a bundle can be obtained by calling `bundle.adapt(BundleWirings`.class). `getWirings()`.

Version:
$Id: ecf3e7820319a6710f4dff063577bd052f5dafe2 $
ThreadSafe

Method Summary		Page
`List<BundleWiring>`	`getWirings()` Return the `in use` wirings for the `referenced` bundle.	32

Methods inherited from interface org.osgi.framework.BundleReference
`getBundle`

Method Detail

getWirings

```
List<BundleWiring> getWirings()
```

Return the `in use` wirings for the `referenced` bundle.

If the referenced bundle is a non-fragment bundle, then the result is a list of in use bundle wirings. The list is ordered in reverse chronological order such that the first bundle wiring is the `current` bundle wiring and last wiring is the oldest in use bundle wiring.

If the referenced bundle is a fragment bundle, then the result is a list of in use bundle wirings to which the referenced fragment bundle is attached. The ordering of the list is unspecified. If the fragment bundle is not attached to any bundle wiring, then the returned list will be empty.

The list must only contain in use bundle wirings. Generally the list will have at least one bundle wiring for the bundle: the current bundle wiring. However, for an uninstalled bundle with no in use bundle wirings or a newly installed bundle which has not been resolved, the list will be empty.

Returns:
A list containing a snapshot of the `BundleWiring`s for the referenced bundle.

Interface Capability

org.osgi.framework.wiring

All Known Subinterfaces:
WiredCapability

`public interface Capability`

A capability that has been declared from a `bundle revision`.

The framework defines capabilities for `packages` and `bundles`.

Version:
$Id: 0ea57aa9a6c8c6ad99cbf918d42b77e156013473 $
ThreadSafe

Field Summary		Page
String	**BUNDLE_CAPABILITY** Capability name space for bundle capabilities.	34
String	**PACKAGE_CAPABILITY** Capability name space for package capabilities.	33

Method Summary		Page
Map<String,Object>	**getAttributes**() Returns the attributes of this capability.	34
Map<String,String>	**getDirectives**() Returns the directives of this capability.	34
String	**getNamespace**() Returns the name space of this capability.	34
BundleRevision	**getProviderRevision**() Returns the bundle revision declaring this capability.	35

Field Detail

PACKAGE_CAPABILITY

```
public static final String PACKAGE_CAPABILITY = "osgi.package"
```

Capability name space for package capabilities. The name of the package is stored in the capability attribute of the same name as this name space. The other directives and attributes of the package, from the `Export-Package` manifest header, can be found in the cabability's `directives` and `attributes`. The `version` capability attribute must contain the `org.osgi.framework.Version` of the package if one is specified. The `bundle-symbolic-name` capability attribute must contain the `symbolic name` of the provider if one is specified. The `bundle-version` capability attribute must contain the `version` of the provider if one is specified.

The package capabilities provided by the system bundle, that is the bundle with id zero, must include the package specified by the `org.osgi.framework.Constants.FRAMEWORK_SYSTEMPACKAGES` and `org.osgi.framework.Constants.FRAMEWORK_SYSTEMPACKAGES_EXTRA` framework properties as well as any other package exported by the framework implementation.

A bundle revision `declares` zero or more package capabilities (this is, exported packages).

A bundle wiring `provides` zero or more resolved package capabilities (that is, exported packages) and `requires` zero or more resolved package capabilities (that is, imported packages). The number of package capabilities required by a bundle wiring may change as the bundle wiring may dynamically import additional packages.

BUNDLE_CAPABILITY

`public static final String` **`BUNDLE_CAPABILITY`** `= "osgi.bundle"`

Capability name space for bundle capabilities. The bundle symbolic name of the bundle is stored in the capability attribute of the same name as this name space. The other directives and attributes of the bundle, from the `Bundle-SymbolicName` manifest header, can be found in the cabability's `directives` and `attributes`. The `bundle-version` capability attribute must contain the `org.osgi.framework.Version` of the bundle, from the `Bundle-Version` manifest header.

A bundle wiring `provides` exactly one† bundle capability (that is, the bundle can be required by another bundle) and `requires` zero or more bundle capabilities (that is, requires other bundles).

† A bundle with no bundle symbolic name (that is, a bundle with `Bundle-ManifestVersion`< 2) must not provide a bundle capability.

Method Detail

getNamespace

`String` **`getNamespace`**`()`

Returns the name space of this capability.

Returns:
The name space of this capability.

---

getDirectives

`Map<String,String>` **`getDirectives`**`()`

Returns the directives of this capability.

Returns:
An unmodifiable map of directive names to directive values for this capability, or an empty map if this capability has no directives.

---

getAttributes

`Map<String,Object>` **`getAttributes`**`()`

Returns the attributes of this capability.

Returns:
An unmodifiable map of attribute names to attribute values for this capability, or an empty map if this capability has no attributes.

---

getProviderRevision

`BundleRevision` **`getProviderRevision`**`()`

Returns the bundle revision declaring this capability.

Returns:
The bundle revision declaring this capability.

Interface FrameworkWiring

org.osgi.framework.wiring

All Superinterfaces:

org.osgi.framework.BundleReference

```
public interface FrameworkWiring
extends org.osgi.framework.BundleReference
```

Query and modify wiring information for the framework. The framework wiring object for the framework can be obtained by calling `bundle.adapt(FrameworkWiring.class)` on the system bundle. Only the system bundle can be adapted to a FrameworkWiring object.

The system bundle associated with this FrameworkWiring object can be obtained by calling `org.osgi.framework.BundleReference.getBundle()`.

Version:

$Id: f9f3f89b5b8d369453d645a52a482a385a2bd520 $

ThreadSafe

Method Summary		Page
`Collection<org.osgi.framework.Bundle>`	**getDependencyClosure**`(Collection<org.osgi.framework.Bundle> bundles)` Returns the dependency closure for the specified bundles.	38
`Collection<org.osgi.framework.Bundle>`	**getRemovalPendingBundles**`()` Returns the bundles that have `non-current`, `in use` bundle wirings.	38
`void`	**refreshBundles**`(Collection<org.osgi.framework.Bundle> bundles, org.osgi.framework.FrameworkListener... listeners)` Refreshes the specified bundles.	36
`boolean`	**resolveBundles**`(Collection<org.osgi.framework.Bundle> bundles)` Resolves the specified bundles.	37

Methods inherited from interface org.osgi.framework.BundleReference
`getBundle`

Method Detail

refreshBundles

```
void refreshBundles(Collection<org.osgi.framework.Bundle> bundles,
                    org.osgi.framework.FrameworkListener... listeners)
```

Refreshes the specified bundles. This forces the update (replacement) or removal of packages exported by the specified bundles.

The technique by which the framework refreshes bundles may vary among different framework implementations. A permissible implementation is to stop and restart the framework.

This method returns to the caller immediately and then performs the following steps on a separate thread:

1. Compute the `dependency closure` of the specified bundles. If no bundles are specified, compute the dependency closure of the `removal pending` bundles.
2. Each bundle in the dependency closure that is in the `ACTIVE` state will be stopped as described in the `Bundle.stop` method.

3. Each bundle in the dependency closure that is in the `RESOLVED` state is unresolved and thus moved to the `INSTALLED` state. The effect of this step is that bundles in the dependency closure are no longer `RESOLVED`.
4. Each bundle in the dependency closure that is in the `UNINSTALLED` state is removed from the dependency closure and is now completely removed from the Framework.
5. Each bundle in the dependency closure that was in the `ACTIVE` state prior to Step 2 is started as described in the `Bundle.start` method, causing all bundles required for the restart to be resolved. It is possible that, as a result of the previous steps, packages that were previously exported no longer are. Therefore, some bundles may be unresolvable until bundles satisfying the dependencies have been installed in the Framework.

For any exceptions that are thrown during any of these steps, a framework event of type `FrameworkEvent.ERROR` is fired containing the exception. The source bundle for these events should be the specific bundle to which the exception is related. If no specific bundle can be associated with the exception then the System Bundle must be used as the source bundle for the event. All framework events fired by this method are also delivered to the specified `FrameworkListener`s in the order they are specified.

When this process completes after the bundles are refreshed, the Framework will fire a Framework event of type `FrameworkEvent.PACKAGES_REFRESHED` to announce it has completed the bundle refresh. The specified `FrameworkListener`s are notified in the order specified. Each specified `FrameworkListener` will be called with a Framework event of type `FrameworkEvent.PACKAGES_REFRESHED`.

Parameters:

`bundles` - The bundles to be refreshed, or `null` to refresh the `removal pending` bundles.
`listeners` - Zero or more listeners to be notified when the bundle refresh has been completed. The specified listeners do not need to be otherwise registered with the framework. If a specified listener is already registered with the framework, it will be notified twice.

Throws:

`IllegalArgumentException` - If the specified `Bundle`s were not created by the same framework instance associated with this FrameworkWiring.
`SecurityException` - If the caller does not have `AdminPermission[System Bundle,RESOLVE]` and the Java runtime environment supports permissions.

---

resolveBundles

`boolean **resolveBundles**(Collection<org.osgi.framework.Bundle> bundles)`

Resolves the specified bundles. The Framework must attempt to resolve the specified bundles that are unresolved. Additional bundles that are not included in the specified bundles may be resolved as a result of calling this method. A permissible implementation of this method is to attempt to resolve all unresolved bundles installed in the framework.

If no bundles are specified, then the Framework will attempt to resolve all unresolved bundles. This method must not cause any bundle to be refreshed, stopped, or started. This method will not return until the operation has completed.

Parameters:

`bundles` - The bundles to resolve or `null` to resolve all unresolved bundles installed in the Framework.

Returns:

`true` if all specified bundles are resolved; `false` otherwise.

Throws:

`IllegalArgumentException` - If the specified `Bundle`s were not created by the same framework instance associated with this FrameworkWiring.
`SecurityException` - If the caller does not have `AdminPermission[System Bundle,RESOLVE]` and the Java runtime environment supports permissions.

---

getRemovalPendingBundles

`Collection<org.osgi.framework.Bundle> **getRemovalPendingBundles**()`

Returns the bundles that have `non-current`, `in use` bundle wirings. This is typically the bundles which have been updated or uninstalled since the last call to `refreshBundles(Collection, FrameworkListener...)`.

Returns:
A collection containing a snapshot of the `Bundle`s which have non-current, in use `BundleWiring`s, or an empty collection if there are no such bundles.

---

getDependencyClosure

`Collection<org.osgi.framework.Bundle> **getDependencyClosure**(Collection<org.osgi.framework.Bundle> bundles)`

Returns the dependency closure for the specified bundles.

A graph of bundles is computed starting with the specified bundles. The graph is expanded by adding any bundle that is either wired to a package that is currently exported by a bundle in the graph or requires a bundle in the graph. The graph is fully constructed when there is no bundle outside the graph that is wired to a bundle in the graph. The graph may contain `UNINSTALLED` bundles that are `removal pending`.

Parameters:
`bundles` - The initial bundles for which to generate the dependency closure.
Returns:
A collection containing a snapshot of the dependency closure of the specified bundles, or an empty collection if there were no specified bundles.
Throws:
`IllegalArgumentException` - If the specified `Bundle`s were not created by the same framework instance associated with this FrameworkWiring.

Interface WiredCapability

org.osgi.framework.wiring

All Superinterfaces:

Capability

```
public interface WiredCapability
extends Capability
```

A wired capability that has been provided from a `bundle wiring`. This capability may or may not be required by any bundle wiring.

A wired capability represents a capability from a resolved bundle wiring.

Version:

$Id: bdf793be0ba691b543c523ca1b608e356aac9963 $

ThreadSafe

Fields inherited from interface org.osgi.framework.wiring.Capability
`BUNDLE_CAPABILITY`, `PACKAGE_CAPABILITY`

Method Summary		Page
`BundleWiring`	`getProviderWiring()` Returns the bundle wiring providing this capability.	*39*
`Collection<BundleWiring>`	`getRequirerWirings()` Returns the bundle wirings that require this capability.	*39*

Methods inherited from interface org.osgi.framework.wiring.Capability
`getAttributes`, `getDirectives`, `getNamespace`, `getProviderRevision`

Method Detail

getProviderWiring

```
BundleWiring getProviderWiring()
```

Returns the bundle wiring providing this capability.

Returns:

The bundle wiring providing this capability. If the bundle wiring providing this capability is not `in use`, `null` will be returned.

getRequirerWirings

```
Collection<BundleWiring> getRequirerWirings()
```

Returns the bundle wirings that require this capability.

The result of this method can change if this capability becomes required by additional bundle wirings.

Returns:

A collection containing a snapshot of the bundle wirings currently requiring this capability, or an empty collection if no bundle wirings require this capability. If the bundle wiring providing this capability is not `in use`, `null` will be returned.

7 Command Line API

Command line API is not considered relevant to this design. Hooks are a very low level concept which should not be accessed by the command line.

8 JMX API

JMX API is not considered relevant to this design. Hooks are a very low level concept which should not be accessed by JMX.

9 Considered Alternatives

The list of alternatives is too long and painful to list. Take a look at the document history to see the long and painful journey up to this point.

10 Security Considerations

When Java permissions are in effect, this design is secured by ServicePermissions.

The bundle registering the various hook services must have the necessary ServicePermission.REGISTER. Since there are various hook services, we have fine grained control over what specific hooks a bundle can register.

11 Document Support

11.1 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

11.2 Author's Address

Name	Thomas Watson
Company	IBM Corporation
Address	
Voice	+1 512 286 9168
e-mail	tjwatson@us.ibm.com

11.3 Acronyms and Abbreviations

11.4 End of Document