RFC 12 − PackageAdmin Service for the Core Framework

Members Only, Final

10 Pages

Abstract

The core framework for OSGi manages the shared package dependencies of the installed bundles. In general it is not necessary for bundle programmers to worry about the internal workings of shared package resolution in the framework, as long as the framework resolves packages correctly. Admin type bundles may be interested in more details of the package importing and exporting as well as greater control over the process. This service provides these bundles with the ability to find out the chosen exporter of a package as well as force the removal or replacement of packages whose exporting bundles have been uninstalled or updated, respectively, and re−resolve any dependent bundles in the framework.

0 Document Information

0.1 Table of Contents

0.2 Status

This document specifies the PackageAdmin interface and related interfaces for the core platform of the Open Services Gateway Initiative, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

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

`Source code is shown in this typeface.`

0.4 Revision History

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

Revision	Date	Comments
Initial	3/15/01	Benjamin Reed, Core Platform Expert Group, OSGi.
Second Draft	3/30/01	Added the PackageAdmin.getExportedPackage() method that was accidentally deleted in the initial draft.
Third Draft	4/3/01	Changed the package name, minor edits, and wording to reflect lazy exporting of packages from the system classpath.
Final 1.0	4/10/01	Very minor copyright and version updates.

1 Introduction

Package management is done relatively transparent to the bundle programmer. There are two tags in the MANIFEST, Import−Package and Export−Package, that define the packages the bundle needs and the packages that the bundle can provide. The framework chooses exactly one bundle from among the possibly many bundles offering a given package for export according to the rules described in the OSGi Framework Specification. There can only be one instance of a shared class in the framework.

In order to make the platform less impacted by updates to or removal of any exporter bundles, a lazy update policy is used (*"lazy update and uninstall"*). This means that the class files of an exported package remain available even if the exporting bundle has been updated or removed from the framework. While this ensures that the update or removal of an exporter does not require any importer bundles to be re−resolved (with the potential of failure), it also means that new behavior of the updated package will not be available until the framework is restarted. This also means that if an exporter bundle gets uninstalled, other bundles in the framework will not be given an opportunity to replace that bundle's exported packages until the framework restarts.

The PackageAdmin service provides admin type bundles more insight into the package relationship between bundles, by listing all exported packages and exposing which bundle has exported which packages. It also allows the exported packages of individual bundles that have been updated or uninstalled to be replaced or removed, respectively, and all dependent bundles to be re−resolved (*"eager update and uninstall"*).

Note that a bundle programmer already can determine the imports and exports of a bundle by examining the bundle's MANIFEST headers using the Bundle.getHeaders method, but this information does not include the specific exporter of an exported package. Also, this information would be accurate only as long as the bundle is not updated. For example, the bundle's exported packages may depend on certain imported packages (for which a corresponding entry in the bundle's Import−Package manifest header exists), but the updated bundle may offer different packages for export that no longer depend on the same packages for import. Therefore, even though the manifest of the updated bundle no longer reflects the bundle's dependency on the imported packages, the packages exported by the previous bundle version (and therefore the bundle itself) will still be importing these packages. Without the PackageAdmin, this information would not be available.

2 Technical Discussion

The framework MAY provide the PackageAdmin service. If present, there MUST be only one instance of the PackageAdmin.

The PackageAdmin service consists of the PackageAdmin interface and the ExportedPackage interface. The PackageAdmin is used to query the framework about its exported packages and for eager bundle updates and uninstalls. The ExportedPackage interface describes an exported package and is instantiated by the query methods of the PackageAdmin.

In this section, any reference to an exported package refers to a package that is actively exported, not just available for export. For example, bundles A and B might both export package com.a.b, but only one will be chosen for export, and there will be only on ExportedPackage corresponding to com.a.b.

2.1 Querying the PackageAdmin

There are two methods to find out information about an exported package: PackageAdmin.getExportedPackages(Bundle) and PackageAdmin.getExportedPackage(String name). The first method returns an array of ExportedPackages and the last returns an ExportedPackage.

PackageAdmin.getExportedPackages(Bundle bundle), returns an array of packages exported by a specific bundle. Passing null as the Bundle argument returns all the exported packages in the framework. If the bundle has been updated, the list of packages will include any packages exported by older versions of the bundle, unless PackageAdmin.refreshPackages has been called.

It should be noted that the framework itself MAY export packages. These packages will be exported by the system bundle, whose bundle id is always zero. The "OSGi Service Gateway: Framework Specification" states that java.* packages are implicitly exported, but other packages such as javax.comm may be on the system class path and need to be explicitly exported by the framework.

PackageAdmin.getExportedPackage(String name) allows the bundle programmer to get information on a specific exported package. If no package with the specified name is among the exported packages in the framework, null is returned.

2.2 The ExportedPackage Interface

The ExportedPackage interface is used to describe an exported package. The attributes that are exposed by this interface are the package name, the bundle that exports the package, the bundles that import the package, the version of the package, and whether the bundle that exports the package has been deleted or updated.

If an exported package gets refreshed (that is, the bundle that is exporting it has been updated or uninstalled and is passed to PackageAdmin.refreshPackages), any instances of ExportedPackage describing the package that were instantiated before the refreshPackages operation become stale. This means that getName() and getSpecificationVersion() will continue returning their old values, isRemovalPending() will return true, and getImportingBundles() and getExportingBundle() will return null.

2.3 Re−exporting Packages

As pointed out above, because of the "lazy update and uninstall" policy, any exported packages whose bundles have been updated or uninstalled will remain exported until the framework is restarted or PackageAdmin.refreshPackages is called. When this happens, the exported packages of these bundles will be removed, and new exporters (if available) will be chosen for these packages by the framework. This requires all dependent (that is, importing) bundles to be re−resolved and their exported packages to be withdrawn, requiring the importers of those packages to be re−resolved and their exported packages to be withdrawn, and so on.

The following scenario illustrates this:

Bundle A exports com.a.b, imports com.a.c

Bundle B exports com.a.c, imports com.a.b

Bundle C imports com.a.c

Bundle D exports com.a.d, imports com.a.e

Bundle E exports com.a.e, imports com.a.d

All bundles are ACTIVE.

If A is uninstalled and passed to PackageAdmin.refreshPackages, com.a.b will be unexported, which will cause B to stop and com.a.c to be unexported, which in turn will cause C to stop. D and E will not be affected and continue to be ACTIVE. An attempt to re−resolve B and C will fail.

PackageAdmin.refreshPackages causes the bundles passed as the argument to be re−resolved. This method returns immediately, and performs its steps on a different thread.

3 Security Considerations

The framework is the only entity that can possibly provide a PackageAdmin service since its implementation is tied so closely to the implementation of the framework. Therefore, a framework should ensure that no bundle is allowed to register the PackageAdmin service.

Access to the PackageAdmin is protected by the corresponding ServicePermission. In addition, PackageAdmin.refreshPackages is protected by the AdminPermission, because it has the potential of severely disrupting the operation of the framework.

4 Document Support

4.1 References

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

4.2 Author's Address

Name	Jan Luehe
Company	Sun Microsystems, Inc.
Address	901 San Antonio Road, UCUP01−207 Palo Alto, CA 94303 USA
Voice	+1 408 863−3216
e−mail	Jan.Luehe@sun.com

Name	Benjamin Reed
Company	IBM
Address	650 Harry Road San Jose, CA 95120 USA
Voice	+1 408 927−1811
e−mail	breed@almaden.ibm.com

4.3 org.osgi.service.packageadmin Interface PackageAdmin

public interface **PackageAdmin**

Framework service which allows bundle programmers to inspect the packages exported in the framework and eagerly update or uninstall bundles.

If present, there will only be a single instance of this service registered in the framework.

The term *exported package* (and the corresponding interface `ExportedPackage`) refers to a package that has actually been exported (as opposed to one that is available for export).

Note that the information about exported packages returned by this service is valid only until the next time `refreshPackages(org.osgi.framework.Bundle[])` is called. If an ExportedPackage becomes stale (that is, the package it references has been updated or removed as a result of calling PackageAdmin.refreshPackages), its getName() and getSpecificationVersion() continue to return their old values, isRemovalPending() returns true, and getExportingBundle() and getImportingBundles() both return null.

Version:

1.0.0

Method Summary

`ExportedPackage`	**`getExportedPackage`**`(java.lang.String name)` Gets the ExportedPackage with the specified package name.
`ExportedPackage[]`	**`getExportedPackages`**`(org.osgi.framework.Bundle bundle)` Gets the packages exported by the specified bundle.
`void`	**`refreshPackages`**`(org.osgi.framework.Bundle[] bundles)` Forces the update (replacement) or removal of packages exported by the specified bundles.

Method Detail

4.3.1 getExportedPackages

`public ExportedPackage[] getExportedPackages(org.osgi.framework.Bundle bundle)`

Gets the packages exported by the specified bundle.

Parameters:

`bundle` − The bundle whose exported packages are to be returned, or `null` if all packages currently exported in the framework are to be returned. If the specified bundle is the system bundle (that is, the bundle with id 0), this method returns all packages on the system classpath whose name does not start with "java.". In an environment where the exhaustive list of packages on the system classpath is not known in advance, this method will return all currently known packages on the system classpath, that is, all packages on the system classpath that contain one or more classes that have been loaded.

Returns:

The array of packages exported by the specified bundle, or `null` if the specified bundle has not exported any packages.

4.3.2 getExportedPackage

`public ExportedPackage getExportedPackage(java.lang.String name)`

Gets the ExportedPackage with the specified package name. All exported packages will be checked for the specified name. In an environment where the exhaustive list of packages on the system classpath is not known in advance, this method attempts to see if the named package is on the system classpath. This means that this method may discover an ExportedPackage that was not present in the array returned by `getExportedPackages`.

Parameters:

`name` − The name of the exported package to be returned.

Returns:

The exported package with the specified name, or `null` if no exported package with that name exists.

4.3.3 refreshPackages

```
public void refreshPackages(org.osgi.framework.Bundle[] bundles)
```

Forces the update (replacement) or removal of packages exported by the specified bundles.

If no bundles are specified, this method will update or remove any packages exported by any bundles that were previously updated or uninstalled. The technique by which this is accomplished may vary among different framework implementations. One permissible implementation is to stop and restart the Framework.

This method returns to the caller immediately and then performs the following steps in its own thread:

1. Compute a graph of bundles starting with the specified ones. If no bundles are specified, compute a graph of bundles starting with previously updated or uninstalled ones. Any bundle that imports a package that is currently exported by a bundle in the graph is added to the graph. The graph is fully constructed when there is no bundle outside the graph that imports a package from a bundle in the graph. The graph may contain `UNINSTALLED` bundles that are currently still exporting packages.

2. Each bundle in the graph will be stopped as described in the `Bundle.stop` method.

3. Each bundle in the graph that is in the `RESOLVED` state is moved to the `INSTALLED` state. The effect of this step is that bundles in the graph are no longer `RESOLVED`.

4. Each bundle in the graph that is in the UNINSTALLED state is removed from the graph and is now completely removed from the framework.

5. Each bundle in the graph 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 another bundle offering a compatible package for export has been installed in the framework.

For any exceptions that are thrown during any of these steps, a `FrameworkEvent` of type `ERROR` is broadcast, containing the exception.

Parameters:

`bundles` − the bundles whose exported packages are to be updated or removed, or `null` for all previously updated or uninstalled bundles.

Throws:

`java.lang.SecurityException` − if the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

4.4 org.osgi.service.packageadmin Interface ExportedPackage

public interface **ExportedPackage**

An exported package.

Instances implementing this interface are created by the `PackageAdmin` service.

Note that the information about an exported package provided by this class is valid only until the next time `PackageAdmin.refreshPackages()` is called. If an ExportedPackage becomes stale (that is, the package it references has been updated or removed as a result of calling PackageAdmin.refreshPackages()), its getName() and getSpecificationVersion() continue to return their old values, isRemovalPending() returns true, and getExportingBundle() and getImportingBundles() both return null.

Version:
1.0.0

Method Summary

`org.osgi.framework.Bundle`	**`getExportingBundle`**`()` Returns the bundle that is exporting this `ExportedPackage`.
`org.osgi.framework.Bundle[]`	**`getImportingBundles`**`()` Returns the resolved bundles that are currently importing this `ExportedPackage`.
`java.lang.String`	**`getName`**`()` Returns the name of this `ExportedPackage`.
`java.lang.String`	**`getSpecificationVersion`**`()` Returns the specification version of this `ExportedPackage`, as specified in the exporting bundle’s manifest file.
`boolean`	**`isRemovalPending`**`()` Returns `true` if this `ExportedPackage` is being exported by a bundle that has been updated or uninstalled.

Method Detail

4.4.1 getName

`public java.lang.String` **`getName`**`()`

Returns the name of this `ExportedPackage`.
Returns:
The name of this `ExportedPackage`.

4.4.2 getExportingBundle

`public org.osgi.framework.Bundle` **`getExportingBundle`**`()`

Returns the bundle that is exporting this `ExportedPackage`.

Returns:

The exporting bundle, or `null` if this `ExportedPackage` has become stale.

4.4.3 getImportingBundles

`public org.osgi.framework.Bundle[]` **`getImportingBundles`**`()`

Returns the resolved bundles that are currently importing this `ExportedPackage`.

The returned array always includes the bundle returned by `getExportingBundle()` since an exporter always implicitly imports its exported packages.

Returns:

The array of resolved bundles currently importing this `ExportedPackage`, or null if this `ExportedPackage` has become stale.

4.4.4 getSpecificationVersion

`public java.lang.String` **`getSpecificationVersion`**`()`

Returns the specification version of this `ExportedPackage`, as specified in the exporting bundle's manifest file.

Returns:

The specification version of this `ExportedPackage`, or `null` if no version information is available.

4.4.5 isRemovalPending

`public boolean` **`isRemovalPending`**`()`

Returns `true` if this `ExportedPackage` is being exported by a bundle that has been updated or uninstalled.

Returns:

`true` if this `ExportedPackage` is being exported by a bundle that has been updated or uninstalled; `false` otherwise.