RFC 55 Dynamic Import Package

Confidential, Draft

8 Pages

Abstract

This RFC describes a framework enhancement to allow a bundle to dynamically import packages at class load time.

0 Document Information

0.1 Table of Contents

0.2 Status

This document specifies proposed enhancement for the Open Services Gateway Initiative, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

0.3 Acknowledgement

The author would like to acknowledge Pam Tobias and Roy Paterson of IBM for their assistance in working out some of the final details of this proposal.

0.4 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 [1].

```
Source code is shown in this typeface.
```

0.5 Revision History

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

Revision	Date	Comments
Initial	10 May 2002	Initial draft. BJ Hargrave, hargrave@us.ibm.com
2nd draft	24 May 2002	Second draft incorporating comments from the Cologne EG meeting. Manifest header supports wildcards; java.* is always dynamically imported; PackagePermission("java.*", "import") is always given to bundles as part of their implied permissions. BJ Hargrave, hargrave@us.ibm.com

1 Introduction

It is a common Java programming idiom for one party to pass a String naming one of his classes to another party. The other party will then use Class.forName to load the named class. When both parties are loaded by the same classloader as happens in a traditional Java application, this works fine. In the OSGi environment, if each party is a separate bundle, this idiom will only work if the first party exports the package containing the named class and the second party imports the same package.

This, however, is impractical in many situations since the second party may need to load classes from packages unknown to the developer at the time the bundle was created. This is the case when the bundle will contain "legacy code" that was designed without knowledge of OSGi. In order to support such legacy code being packaged as an OSGi bundle, we need a mechanism for the bundle to dynamically import packages as necessary.

All Page within this Box

All Page Within This Box

2 Application Domain

This problem addressed by this RFC has been known for some time and has been tracked by Issue #44[3] in the OSGi issue database. Quoting from the issue:

"Legacy code (like JMF) uses Class.forName() extensively. Classes used for this are often implementation classes so the bundle that holds the legacy code cannot a priori import the packages of those classes."

3 Problem Description

In order for bundle B to load a class contained in bundle A, bundle A must export the package containing the class and bundle B must import the package. This situation works fine in the case where both bundles have a priori knowledge of the classes (and thus packages) that they will share. The Import-Package and Export-Package manifest header are designed to satisfy this purpose and the bundle resolve action ensures that the appropriate packages are available before permitting the bundle to be resolved.

However, this does not work for bundles that do not have a priori knowledge of the classes they will load. This is the case in the Class.forName idiom where the class to loaded is specified by an argument pass from a calling bundle. A solution is necessary that allows a bundle to load a class (and thus import the package containing the class) without a priori knowledge.

4 Requirements

The solution must work within the existing package sharing model of the OSGi framework. Once a bundle dynamically imports a package, the bundle must be returned by PackageAdmin.getExportedPackage(packageName).getImportingBundles(). Furthermore, the bundle must be refreshed if the bundle exporting the package is refreshed.

The solution must not require any code changes to the bundle that desires to dynamically import a package. This is necessary to allow legacy code to be packaged as a bundle without requiring any code changes. Sometime the source to the legacy code is not available. Even if it were, derivative works of the legacy code may not be possible due to licensing issues.

5 Technical Solution

This RFC proposes adding dynamic import package support to the framework. This support will allow a bundle with a newly defined manifest header to dynamically import a package at class load time.

5.1 New Bundle Manifest Header

The framework must now support the DynamicImport-Package bundle manifest header. The String constant for this new header must be defined in org.osgi.framework.Constants as:

```
public static final String DYNAMICIMPORT_PACKAGE = "DynamicImport-Package";
```

The DynamicImport-Package manifest header must conform to the following syntax:

```
DynamicImport-Package = package-name *( "," package-name )

package-name = "*" | <fully qualified package name> | partial-package-name

partial-package-name = *(<package name component> ".") "*"
```

Bundles can specify packages they wish to dynamically import. The use of trailing (".*") wildcards is supported. The full wildcard of "*" is also supported indicating all package can be dynamically imported.

The bundle's classloader(s) must support dynamically importing packages during class load requests. At the time the classloader would normally check to see if the requested class was from an imported package, the classloader must attempt to import the requested class' package if

1. The package matches (including wildcard matches) one of the package names specified on the DynamicImport-Package manifest header (if the header is present) or it wildcard matches "java.*"
2. The requesting bundle does not already import the package
3. The package is currently exported by a bundle (including the system bundle)
4. The requesting bundle has PackagePermission[IMPORT] for the package if permissions are supported

If all the above conditions are met, then the framework must cause the package to be imported by the bundle as if it were imported during normal bundle resolving. After importing the package, the requested class can then be loaded from the exporting bundle satisfying the class load request.

After a bundle dynamically imports a package, the framework must reflect that the bundle has imported the package. Calls to PackageAdmin regarding the bundle or the package must show that the bundle is importing the package. A PackageAdmin.refreshPackage call that results in the bundle exporting the package being refreshed must also result in the bundles that have dynamically imported the package being refreshed. In summary, there is no difference between a bundle that statically imported a package (via Import-Package) and bundle that has dynamically imported a package. Both bundles are "bound" to the exported package and act similarly with respect to the package.

Support for DynamicImport-Package does alter the normal package sharing process (Import-Package and Export-Package manifest header processing) during bundle resolve. Packages named by the Import-Package manifest header must be available to successfully resolve the bundle.

The framework must only allow packages to be dynamically imported by the bundle if the package matches a package specified on the DynamicImport-Package manifest header or matches “java.*”. The bundle’s classloaders will otherwise only import packages specified by the Import-Package manifest header.

5.2 java.* Packages

All bundles must dynamically import packages which wildcard match “java.*” even if this is not specified on the DynamicImport-Package manifest header. Since bundles are not required to specify packages beginning with “java.” on the Import-Package manifest header, this allows the java.* package family to be easily provided by either the normal system classpath or via another bundle with no *a priori* knowledge of where the package is delivered.

In support of the above, the framework must give all bundles the implied permission: PackagePermission(“java.*”, “import”) to allows them to successfully dynamically import packages starting with “java.”.

6 Considered Alternatives

6.1 Method based solutions

Several alternative solutions were proposed[3]. These solutions required some bundle to have knowledge of the package name to be dynamically imported. These solutions were rejected in favor of the proposed solution that did not require any new methods nor require bundle code having to possess sufficient knowledge to call such methods.

6.1.1 BundleContext addImport

```
boolean BundleContext.addImport( String package );
```

This method must dynamically add the package to the list of imports as if it was added to the manifest. Security restrictions apply. If the package cannot be resolved, a false is returned and the package is not imported.

6.1.2 Resolver interface

```
public interface Resolver {

   Class findClass( String name );

   InputStream findResource( String name );
```

All Page within this Box

All Page Within This Box

```
}


BundleContext.setResolver( Resolver r );

BundleContext.removeResolver( Resolver r );
```

The Class loader would call the Resolver when a class cannot be found in the normal way. This is obviously a potential security issue.

6.1.3 PackageAdmin importPackages

```
public void importPackages(String[] packages) throws BundleException,
SecurityException;
```

This method will dynamically import packages if they are statically exported in the Framework. java.lang.SecurityException will be thrown if the caller does not have the AdminPermission and the Java runtime environment supports permissions.

7 Security Considerations

This solution will use the existing PackagePermission model for controlling the importing of packages. A bundle specifying a DynamicImport-Package manifest header must also possess the necessary PackagePermission[IMPORT] permission to successfully import a package. A bundle that wished to be able to dynamically import any possible class will need PackagePermission("*", "import") permission. More restrictive PackagePermissions can be assigned to the bundle to control the actual packages that may be imported.

The framework must give all bundles the implied permission: PackagePermission("java.*", "import").

8 Document Support

8.1 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

[3]. Issue #44, http://membercvs.osgi.org/rt/rt.cgi?serial_num=44&display=History

8.2 Author's Address

Name	BJ Hargrave
Company	IBM
Address	11501 Burnet Rd, Austin, TX 78758 USA
Voice	+1 512 838 9938
e-mail	hargrave@us.ibm.com

8.3 Acronyms and Abbreviations

8.4 End of Document