RFC 0072 - Native Code Changes

Confidential, Final Review Draft

15 Pages

Abstract

This RFC describes enhancements to the native code loading model.

0 Document Information

0.1 Table of Contents

-

All Page Within This Box

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 [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	21 Nov 2003	First Draft Jennifer Fogell, IBM jfogell@us.ibm.com BJ Hargrave, IBM, hargrave@us.ibm.com
2nd draft	14 Dec 2004	Updated based upon comments from the 3-4 Dec 2003 CPEG meeting. • Removed Bundle-OptionalNativeCode Header • Added an optional clause to Bundle-NativeCode • Removed the windowing system property from the Bundle-NativeCode clause • Add an optional filter property to Bundle-NativeCode clause Jennifer Fogell, IBM jfogell@us.ibm.com BJ Hargrave, IBM, hargrave@us.ibm.com
3rd draft	4 Feb 2004	Updated based upon comments from the 28-29 Jan 2004 CPEG meeting. • Added win32 alias • All path names are relative to the root of the bundle/fragment • Made example have consistent use of leading slashes in nativepath elements Jennifer Fogell, IBM jfogell@us.ibm.com BJ Hargrave, IBM, hargrave@us.ibm.com

All Page Within This Box

4th draft	24 Feb 2004	Updated based upon comments from the 11th Feb 2004 CPEG meeting • Changed text regarding searching fragments for nativepath elements Updated to address cpeg issue #251 • Added native code considerations section Jennifer Fogell, IBM jfogell@us.ibm.com BJ Hargrave, IBM, hargrave@us.ibm.com
Final Review Draft	14 April 2004	Peter Kriens added table a beginning of 5.2. Moved to Final Draft Review.
Final Review 2nd Draft	21 April 2004	Updated to address invalid filter syntax in native code clauses Jennifer Fogell, IBM, jfogell@us.ibm.com
Final Review 3rd Draft	5 May 2004	Change osversion to use new version range syntax from RFC 70[7]. Updated Bundle-NativeCode syntax to use jar-path. selection-filter constant removed because it is specified in RFC 71. BJ Hargrave, IBM, hargrave@us.ibm.com

1 Introduction

The OSGi Framework was originally designed for embedded use and its design has been remarkably robust and stable. However, additional usage scenarios for the framework are now appearing that require the framework be further enhanced.

One area that requires further enhancement is in the bundle native code model. This RFC describes the additional requirements, enhancements and design of the bundle native code model.

All Page Within This Box

2 Application Domain

The original context for the framework was the embedded world (e.g. vehicles, gateways, etc.). We are now seeing the need to deploy the framework and its service oriented architecture in different contexts. While the basic operation of the framework is correct. Additional features are needed to enable the framework to be fully useful on new contexts such as desktops and servers which impose additional requirements in the applications that are to be run within the framework. These changes are also important and useful in the traditional contexts in which the OSGi framework is currently successful.

3 Problem Description

Further enhancements to the framework are necessary to support the new usage models the framework is being asked to fulfill. RFC specifies proposed solutions to enhance the bundle native code model.

4 Requirements

4.1 Native Code Matching Enhancements

Currently the framework selects native code based upon operating system, processor and language. An optional filter must be added as a selection criteria as well. This is necessary to select native code based on a filter's match against system properties.

4.1.1 Use Cases

1. A bundle contains the native libraries for several windowing systems. The correct library should be selected for the current windowing system. The current windowing system is set in a system property. The appropriate filter can be specified in the corresponding native code clause. The filter will be matched against the system properties when the native code algorithm is run.

All Page Within This Box

4.2 Relax Constraints on Presence of Native Libraries

Currently a bundle install/update must fail if the bundle has a Bundle-NativeCode header and there is no matching clause. It should be possible for a bundle to be optionally installed if there is no matching native library. The bundle can take necessary action if there is no native library available for the platform.

4.2.1 Use Cases

1. A bundle provides a crypto API to other bundles. This bundle contains an all Java implementation of the crypto API so that it operates on all platforms. The bundle also contains native implementations of the crypto library for some platform as a performance enhancement. The bundle's manifest has a Bundle-NativeCode header which lists the native libraries for the specific platforms. If this bundle is installed on a platform for which there is no matching clause specified in the Bundle-NativeCode header, the installation will fail even though the bundle is able to function by using the all Java implementation. This requirement would allow the installation of the bundle to succeed.

5 Technical Solution

To put the proposed changes in context, this RFC has copied sections from the OSGi R3 spec and modified them to reflect the proposed specification changes. The respective sections of the R3 spec are referenced in the titles of the sections below.

5.1 Installing Native Code Libraries[4]

When a bundle makes a request to load a native code library, the findLibrary method of the caller's classloader must be called to return the file path name in which the Framework has made the requested native library available. The bundle must have the required RuntimePermission[loadLibrary.< library name>] in order to load native code in the OSGi Service Platform. The Bundle-NativeCode manifest header must conform to the following syntax:

```
Bundle-NativeCode ::= nativecode-clause ( ',' nativecode-clause )* ( ','
optional-clause )?

nativecode-clause ::= nativepath ( ';' nativepath )* ( ';' parameter )*

nativepaths ::= jar-path | ('"'jar-path'"')

parameter ::= ( processordef | osnamedef | osversiondef | languagedef |
filterdef )

processordef ::= 'processor' '=' value

osnamedef ::= 'osname' '=' value
```

All Page Within This Box

```
osversiondef ::= 'osversion' '=' version-range

languagedef ::= 'language' '=' value

filterdef ::= 'selection-filter' '=' quoted-string

value ::= token | quoted-string

optional-clause ::= '*'
```

When locating a `jar-path` entry in a bundle the Framework must attempt to locate the `jar-path` entry relative to the root of the bundle that contains the corresponding NativeCode clause in its manifest header.

The following is a typical example of a native code declaration in a bundle's manifest:

```
Bundle-NativeCode: lib/http.DLL ;
          lib/zlib.dll ;
          osname = Windows95 ;
          osname = Windows98 ;
          osname = WindowsNT ;
          processor = x86 ;
          selection-filter = "(org.osgi.framework.windowing.system =
          win32)";
          language = en ;
          language = se ,
          lib/solaris/libhttp.so ;
          osname = Solaris ;
          osname = SunOS ;
          processor = sparc,
          lib/linux/libhttp.so ;
          osname = Linux ;
          processor = mips;
          selection-filter = "(org.osgi.framework.windowing.system =
          gtk)"
```

All Page Within This Box

If a Bundle-NativeCode clause contains duplicate parameter entries, the corresponding values must be OR'ed together. This feature must be carefully used because the result is not always obvious. This is highlighted by the following example:

```
// The effect of this header

// is probably not the intended effect!

Bundle-NativeCode: lib/http.DLL ;

          osname = Windows95 ;

          osversion = 3.1 ;

          osname = WindowsXP ;

          osversion = 5.1 ;

          processor = x86
```

The previous example implies that the native library will load on Windows XP 3.1 and later, which was probably not intended. The single clause should be split up when the expected effect is desired:

```
Bundle-NativeCode: lib/http.DLL ;

          osname = Windows95 ;

          osversion = 3.1;

          processor = x86,

          lib/http.DLL ;

          osname = WindowsXP ;

          osversion = 5.1;

          processor = x86
```

If multiple native code libraries need to be installed on one platform, they must be specified in the same clause for that platform.

If the optional clause is specified at the end of the Bundle-NativeCode manifest header, a bundle installation error will not occur if the Bundle-NativeCode header has no matching clauses.

The following is a typical example of a native code declaration in a bundle's manifest with an optional clause:

```
 Bundle-NativeCode: lib/win32/winxp/optimized.dll ;

          lib/win32/native.dll ;

          osname = WindowsXP ;
```

```
processor = x86 ,

lib/win32/native.dll ;

osname = Windows95 ;

osname = Windows98 ;

osname = WindowsNT ;

osname = Windows2000;

processor = x86 ,

*
```

5.1.1 Native Code Algorithm[5]

In the description of this algorithm, [x] represents the value of the Framework property x and ~= represents the match operation. The match operation is a case insensitive comparison. The manifest header should contain the generic name for that property but the Framework should attempt to include aliases when it matches. (See Environment Properties on page X). If a property is not an alias, or has the wrong value, the Operator should set the appropriate system property to the generic name or to a valid value (System properties with this name override the Framework construction of these properties). For example, if the operating system returns version 2.4.2-kwt, the Operator should set the system property org.osgi.framework.os.version to 2.4.2. The Framework must select the native code clause selected by the following algorithm:

1. Select only the native code clauses for which the following expressions all evaluate to true.
   - osname ~= [org.osgi .framework.os.name]
   - processor ~= [org.osgi .framework.processor]
   - osversion range includes [org.osgi.framework.os.version] or osversion is not specified
   - selection-filter evaluates to true when using the values of the system properties or selection-filter is not specified
   - language ~= [org.osgi.framework.language] or language is not specified
2. If no native clauses were selected in step 1, this algoritm is terminated and a BundleException is thrown if the optional clause is not present.
3. The selected clauses are now sorted in the following priority order:
   - osversion: floor of the osversion range in descending order, osversion not specified
   - language: language specified, language not specified
   - Position in the Bundle-NativeCode manifest header: lexical left to right.
4. The first clause of the sorted clauses from step 3 must be used as the selected native code clause.

All Page Within This Box

If a native code library in a selected native code clause cannot be found within the bundle, then the bundle installation must fail with a BundleException. This is true even if the optional clause is specified.

If a selection filter is evaluated and its syntax is invalid, then the bundle installation must fail with a BundleException. If a selection filter is not evaluated (it may be in a native code clause where the osname or processor do not match) then the invalid filter must not cause the bundle installation to fail. This is also true even if the optional clause is specified.

5.2 Example Bundle-NativeCode Manifest Headers

Designing a bundle native code header can become complicated quickly when different operating systems, libraries and language are used. The best practice to design the header is to place all parameters in a table. Every targeted environment is then a row in than table. Each column is one of osname, osversion, processor and language,

OS Name	Version	Processor	Language	Filter	Libs
win32		x86	en		nativecodewin32.dll, nativecode1win32.dll
linux		x86	en	(org.osgi.framework.windowing.systems=gtk)	nativecodegtk.so
linux		x86	en	(org.osgi.framework.windowing.system=qt)	nativecodeqt.so

The previous table makes it easier to detect missing combinations. This table is then mapped to the Bundle-NativeCode header in the following table.

Header	Sample	Description

Bundle-NativeCode	nativecodewin32.dll;
	nativecode1win32.dll;
	processor=x86;
	osname=win32;
	language=en,
	nativecodegtk.so;
	processor=x86;
	osname=linux;
	selection-filter = “(org.osgi.framework.windowing.system = gtk)”;
	language=en,
	nativecodeqt.so;
	processor=x86;
	osname=linux;
	selection-filter =
		If the native code algorithm fails to select a native code clause from the Bundle-NativeCode header, the bundle will fail to install and a BundleException will be thrown.
		
		The selection filter is used to select the appropriate clause based upon the windowing system.

All Page Within This Box

	“(org.osgi.framework.windowing.system = qt)”; language=en	
Bundle-NativeCode	nativecode.dll; nativecode1.dll; processor=x86; osname=Windows2000, libnativecode.so; processor=x86; osname=linux, *	No matches in the Bundle-NativeCode clauses must not cause the bundle installation to fail since there is an optional clause present.

All Page Within This Box

Bundle-NativeCode	libnnicetohave.so;
	
	libnativemusthave.so;
	
	processor=x86;
	
	osname=linux;
	
	selection-filter = "(org.osgi.framework.windowing.system = gtk)",
	
	libnativemusthave.so;
	
	processor=x86;
	
	osname=linux,
	
	*
		In this case, if the os is linux, processor is x86 and the specified filter matched the system properties, the selected native code would be libnativemusthave.so and libnicetohave.so. If the os is linux , the processor is x86 and the filter did not match, then the selected native code would be libnativemusthave.so.

5.3 Loading Native Libraries

When a class located in a bundle or fragment attempts to load a native library the classloader of the class must do the following to find the native library.

- Search for the library in the bundle associated with the classloader. If the bundle does not contain the library, then search for the library in the attached fragment bundles. Fragment bundles are searched in ascending bundle id order. If the library is not found then null must be returned (this allows the parent classloader to search for the native library).

5.4 Considerations Using Native Libraries

There are some restrictions on loading native libraries due to the nature of classloaders[7]. In order to preserve namespace separation in classloaders, only one classloader can load a native library as specified by an absolute path. Loading of a native library file by multiple classloaders (from multiple bundles, for example) will result in a linkage error.

If a fragment is attached to multiple host bundles, the fragment's native code must be accessible through a unique file for each host bundle. This is to prevent a linkage error resulting from loading of the same native library file by multiple classloaders (bundles).

A native library is unloaded only when the classloader that loaded it has been garbage collected. When a bundle is uninstalled or updated, any native libraries loaded by the bundle will not be unloaded until the bundle's classloader is garbage collected. The garbage collection will not happen until all references to objects in the bundle have been garbage collected and all bundles importing packages from the updated or uninstalled bundle are refreshed. Native Libraries loaded from the system classloader are never unloaded because the system classloader is never garbage collected.

5.5 New win32 osname Alias

The “win32” alias will be added to all the non-CE versions of Windows. Windows 2003 should also be added to the list.

6 Considered Alternatives

The first draft introduced a Bundle-OptionalNativeCode header. However, the addition of a new header was not necessary as the same thing could be accomplished with an optional clause in the Bundle-NativeCode header.

The first draft also introduced an optional windowing system parameter to the native code clause. However the addition of a filter parameter allows matching based on system properties including windowing system. The filter is also far more extensible as it can be used to match against any set of system properties.

7 Security Considerations

This RFC does not pose any additional security considerations. The existing security mechanism around native code is still sufficient.

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]. The Java 2 Package Versioning Specification

http://java.sun.com/j2se/1.4/docs/guide/versioning/index.html

[4]. OSGI Service Platform Release 3 Section 4.6 Loading Native Code Libraries

[5]. OSGI Service Platform Release 3 Section 4.6.1 Native Code Algorithm

[6]. RFC 71 Framework API changes. http://membercvs.osgi.org/rfcs/rfc0071

[7]. RFC 70 Bundle Dependency and Class Loading changes. http://membercvs.osgi.org/rfcs/rfc0070

[8]. Overview of the JNI Design http://java.sun.com/docs/books/jni/html/design.html

8.2 Author's Address

Name	Jennifer Fogell
Company	IBM
Address	11501 Burnet Rd, Austin, TX 78758 USA
Voice	+1 512 838 4856
e-mail	jfogell@us.ibm.com

8.3 Acronyms and Abbreviations

8.4 End of Document

All Page Within This Box