RFC 188 - Native Namespace

Final

21 Pages

Abstract

The generic capabilities and requirements model in OSGi provides a common way to express most dependencies specified by the OSGi Core specification. Namespaces have been defined for most all of the standard OSGi dependencies. For example, osgi.wiring.package, osgi.wiring.bundle, and osgi.wiring.host. For the OSGi Core R5 specification we also introduced the osgi.ee namespace to handle the Bundle-RequiredExecutionEnvironment dependency. The only standard dependency from the Core specification that is missing a generic namespace is for native code as defined by the Bundle-NativeCode header.

This RFC provides a new standard namespace for defining bundle native code dependencies.

0   Document Information

0.1   License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the "Distribution") in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback ("Feedback") on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2   Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3   Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4   Table of Contents

0.5   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 10.1.

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

0.6   Revision History

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

Revision	Date	Comments
Initial	Sep 4 2012	Thomas Watson, IBM, tjwatson@us.ibm.com initial version
1.1	Sep 27 2012	Thomas Watson, IBM, tjwatson@us.ibm.com Updates based on Basil F2F meeting. • osgi.native.environment is a payload namespace • Use ~= approximate equals filter operations for matching. No longer requires lowercasing • Describe more about native code selection and sorting algorithm for the framework.

Revision	Date	Comments
1.2	Oct 19 2012	Thomas Watson, IBM, tjwatson@us.ibm.com Updates based on CPEG call on Oct 11 2012 • Fixed usages of ~= • Made use of native.paths optional for selection of native code paths. • Added security considerations. A new implied permission is needed for the osgi.native.environment REQUIRE. • Used the clarified term "launching properties" from the R6 core specification. • Added javadoc for the NativeEnvironmentNamespace class
1.3	Oct 29 2012	Thomas Watson, IBM, tjwatson@us.ibm.com Updates based on CPEG call on Oct 25, 2012 • Allow osgi.native.environment requirements to be defined with the Require-Capability header • Disallow osgi.native.envornment capabilities to be defined with the Provide-Capability header • Remove JMX and initial spec chapter sections • Added DTO section • Moved all mentions of native.paths attribute and native path selection to a non-normative section at the end of section 5.
1.4	Nov 10 2012	Thomas Watson, IBM, tjwatson@us.ibm.com Renamed namespace to osgi.native
1.5	Feb 1 2013	Thomas Watson, IBM, tjwatson@us.ibm.com • Update to add text for bug https://www.osgi.org/members/bugzilla/show_bug.cgi?id=2456 • Update to add text for bug https://www.osgi.org/members/bugzilla/show_bug.cgi?id=2457

1   Introduction

The generic capabilities and requirements model in OSGi provides a common way to express most dependencies specified by the OSGi Core specification. Namespaces have been defined for most all of the standard OSGi dependencies. For example, osgi.wiring.package, osgi.wiring.bundle, and osgi.wiring.host. For the OSGi Core R5 specification we also introduced the osgi.ee namespace to handle the Bundle-RequiredExecutionEnvironment dependency. The only standard dependency from the Core specification that is missing a generic namespace is for native code as defined by the Bundle-NativeCode header.

2   Application Domain

A company provides deployment software for managing OSGi bundles across a set of platforms (Windows, Linux, Mac etc.). The deployment software uses OSGi generic requirements and capabilities in order to determine a valid set of bundles to install into a particular platform. Some bundles that need to be managed contain native code. As such the bundles with native code must only be provisioned to the platforms which matches the native code they provide.

3   Problem Description

The introduction of the generic capabilities and requirements model in OSGi Core R4.3 and R5 have allowed for bundle dependencies to be reasoned about in a generic way. This generic model can be used by the core framework implementations as well as deployment implementations for modeling bundle dependencies. This can be very powerful because it allows for more predictable and stable deployment of bundles because the deployment software can reason about the bundles it is deploying and easily determine what if any dependencies are missing.

A new bundle native code namespace would help in the deployment of bundles with native code because it would allow for deployment software to easily determine if a particular bundle can actually function on the running environment/platform. Currently native code dependencies are not expressed in the generic capabilities and requirements runs model. But if native code requirements are ignored by deployment software then the operator runs

the risk of installing bundles that are not applicable to the running environment/platform, resulting in a broken deployment.

4   Requirements

- NTV-0010: The solution MUST define a new osgi namespace which allows a requirement to be used to represent the native code clauses defined by the Bundle-NativeCode header. This is referred to as the native namespace.

- NTV-0020: The solution SHOULD allow bundles to define requirements directly in the new native namespace. The native namespace is intended to represent the Bundle-NativeCode header in the generic requirement/capability model but other usecases may be satisfied by allowing generic requirements in the native namespace.

- NTV-0040: The solution MUST allow the system bundle to provide capabilities in the native namespace for satisfying native requirements.

- NTV-0045: Except for the system bundle, the solution must prohibit other bundles from providing capabilities in the native namespace. This implies a bundle is not allowed to define a capability using the native namespace with the Provide-Capability header.

- NTV-0050: The solution MUST allow for native requirements to be optional.

- NTV-0060: The solution SHOULD not require special rules for matching or wiring by OSGi Resolver service.

- NTV-0070: The solution SHOULD not define the mechanism a framework uses to select the native paths at runtime. This solution is scoped to only defining the resolvability of a bundle with native code. The actual native path selection is left to be implementation specific.

5   Technical Solution

A new namespace is defined by the core specification named "osgi.native". An osgi.native capability is used to describe the native environment in which the framework is executing. An osgi.native requirement is used by a bundle to describe the native environment required to execute native code packaged within the bundle.

Native code included in a fragment bundle gets loaded by the class loaders of the host bundles the fragment is attached to. Since fragments add native code as payload to the hosts they are attached to, a osgi.native requirement from a fragment is considered to be a payload requirement.

5.1   Native Environment Capability

An OSGi Framework must provide a capability in the osgi.native namespace that represents the native environment in which the Framework is executing. The following capability attributes are defined:

- osgi.native.osname – a List<String> value containing the alias values of the org.osgi.framework.os.name launching property. For example, an org.osgi.framework.os.name value of "Windows95" will get a List<String> value of "Windows95,Windows 95,Win95,Win32" according to http://www.osgi.org/Specifications/Reference

- osgi.native.osversion - value is type Version as parsed from the value of org.osgi.framework.os.version property

- osgi.native.processor – a List<String> value containing the alias values of the org.osgi.framework.processor launching property. For example, an org.osgi.framework.processor value of "x86" will get a List<String> value of "x86,pentium,i386,i486,i586,i686" according to http://www.osgi.org/Specifications/Reference

- osgi.native.language - the value of org.osgi.framework.language property

Provide-Capability headers in the osgi.native namespace must not be used in the manifest, Require-Capability with this Namespace is allowed.

5.1.1   Arbitrary Matching Attributes

Frameworks must also populate the osgi.native capability attributes with the values included in the Framework launching properties (section 4.2.2 of the latest core spec) Launching property keys that start with "osgi.native." are excluded to prevent collisions with the defined capability attribute keys. [XXX – may also want to exclude any values that start with dot '.' similar to config admin].

5.1.2   Example

Running on a Windows 7 machine 64-bit, with en_US the system bundle would have the following capability plus any attributes specified as framework configuration values:

```
osgi.native;
 osgi.native.osname:List<String>="Windows7,Windows 7,Win7,Win32";
 osgi.native.osversion:Version="7.0";
 osgi.native.processor:List<String>="x86-64,amd64,em64t,x86_64";
 osgi.native.language="en";
```

5.2   Native Environment Requirement

Frameworks must convert a Bundle-NativeCode header to a requirement in the osgi.native namespace. A Bundle-NativeCode header may have multiple native code clauses. Each Bundle-NativeCode clause is composed of the following:

- A list of native code paths

- The following set of native code matching attributes:

- osname – Name of the operating system. Is matched against a canonical name as specified by the org.osgi.framework.os.name property value

- osversion – The operating system version. The value of this attribute must be a version range and is matched against the version as specified by the org.osgi.framework.os.version property value.

- language – The ISO code for a language. The value of this attribute is matched against the language specified by the org.osgi.framework.language property value.

- selection-filter – A specified filter which is matched against the framework configuration property values.

- If optional then the final native code clause is a single asterisk (*) character

5.2.1 Native Code Clause Filter Component

Each Native Code clause is converted into a filter component for the osgi.native namespace using the following architected matching attributes:

- osgi.native.osname – use the approximate equals (~=) filter type to evaluate the value specified by the osname Bundle-NativeCode attribute.

- osgi.native.osversion – create a VersionRange using the value specified by the osversion Bundle-NativeCode attribute and then create a filter string out of the VersionRange.

- osgi.native.processor – use the approximate equals (~=) filter type to evaluate the value specified by the processor Bundle-NativeCode attribute.

- osgi.native.language – use the approximate equals (~=) filter type to evaluate the value specified by the language Bundle-NativeCode attribute

In cases where the same Bundle-NativeCode attribute is specified multiple times within the same clause then the filter components for each value for that attribute are or'ed together. For example, if osname attribute is specified as both Windows95 and Windows7 then the resulting filter will contain:

```
(|
  (osgi.native.osname~=Windows95)
  (osgi.native.osname~=Windows7)
)
```

If the selection-filter Bundle-NativeCode attribute is specified then the specified filter is included as a component of the native code clauses AND filter type. Consider the following Bundle-NativeCode header which contains a single clause:

```
Bundle-NativeCode:
  lib/http.dll; lib/zlib.dll;
    osname=Windows95;
    osname=Windows98;
    osname=WindowsNT;
    processor=x86;
    selection-filter="(com.acme.windowing=win32)";
    language=en;
    language=se
```

This clause would get translated into the following AND filter type:

```
Require-Capability:
  osgi.native;
    filter:="
      (&
        (|
          (osgi.native.osname~=Windows95)
          (osgi.native.osname~=Windows98)
          (osgi.native.osname~=WindowsNT)
        )
        (osgi.native.processor~=x86)
        (|
          (osgi.native.language~=en)
          (osgi.native.language~=se)
        )
        (com.acme.windowing=win32)
      )"
```

5.2.2 Combining Native Filter Components

The Bundle-NativeCode header may specify multiple clauses, each having their own list of native code paths and set of matching attributes. Instead of using a separate osgi.native requirement for each Bundle-NativeCode clause, the complete Bundle-NativeCode header is specified as a single osgi.native requirement. This is done by using an OR filter type using all of the individual Bundle-NativeCode clause filter components (as specified above) as components of a single filter directive. Consider the following Bundle-NativeCode header which contains three clauses:

```
Bundle-NativeCode:
  lib/http.dll; lib/zlib.dll;
    osname=Windows95;
    osname=Windows98;
    osname=WindowsNT;
    processor=x86;
    selection-filter = "(com.acme.windowing=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="(com.acme.windowing=gtk)"
```

This Bundle-NativeCode header would get translated into the following osgi.native filter directive:

```
(|
  (&
    (|
      (osgi.native.osname~=Windows95)
      (osgi.native.osname~=Windows98)
      (osgi.native.osname~=WindowsNT)
    )
    (osgi.native.processor~=x86)
```

```
   (|
     (osgi.native.language~=en)
     (osgi.native.language~=se)
   )
   (com.acme.windowing=win32)
 )
 (&
   (|
     (osgi.native.osname~=Solaris)
     (osgi.native.osname~=SunOs)
   )
   (osgi.native.processor~=sparc)
 )
 (&
   (osgi.native.osname~=Linux)
   (osgi.native.processor~=mips)
   (com.acme.windowing=gtk)
 )
)
```

5.2.3

5.2.4 Optionality

If the optional '*' is specified at the end of the Bundle-NativeCode manifest header, then the native code for the bundle is considered to be optional. When the framework converts a Bundle-NativeCode header into an osgi.native requirement which is designated as optional then the requirement resolution directive must be set to 'optional'.

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

```
Bundle-NativeCode:
  lib/http.dll; lib/zlib.dll;
    osname=Windows95;
    osname=Windows98;
    osname=WindowsNT;
    processor=x86;
    selection-filter="(com.acme.windowing=win32)";
    language=en;
    language=se,
    *
```

This Bundle-NativeCode header would be converted to the following osgi.native requirement with the proper resolution directive:

```
Require-Capability:
  osgi.native;
    filter:="
      (&
        (|
          (osgi.native.osname~=Windows95)
          (osgi.native.osname~=Windows98)
          (osgi.native.osname~=WindowsNT)
        )
        (osgi.native.processor~=x86)
```

```
            (|
            (osgi.native.language~=en)
            (osgi.native.language~=se)
            )
            (com.acme.windowing=win32)
        )";
    resolution:="optional"
```

5.3   Specification Update for Invalid Selection Filters

Section 3.10.1 of the R5 Core specification has the following paragraph:

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

This paragraph will be removed in the R6 core framework specification. The following text will be inserted in section 3.10 after defining the architected attributes for Bundle-NativeCode header (osname, selection-filter etc.):

> If a selection-filter attribute contains an invalid filter syntax, then the bundle must fail to install with a BundleException of type NATIVECODE_ERROR.

5.4   Specification Update for Missing Native Code Paths

Section 3.10.1 of the R5 Core specification has the following text:

> If a native code library in a selected native code clause cannot be found within the bundle then the bundle must fail to resolve. This is true even if the optional clause is specified.

This text will be removed in the R6 core framework specification. The following text will be inserted in section 3.10.1 in place of the text above:

> If a native code library in a selected native code clause cannot be found within the bundle then the bundle is still allowed to resolve. A missing native code library will result in an error being thrown at runtime when the bundle attempts to load the native code (e.g. by invoking the method System.loadLibrary).

5.5   Selecting the Native Code Paths

This section is not normative and is not intended to be included in the specification chapter. It is included in the RFC as a documentation of how the proof of concept implementation uses a purely generic requirement/capability model for doing native path selection.

When the Framework resolves a bundle with an osgi.native requirement then post processing must be done to determine the native paths that should be selected. Native path selection is done outside of the OSGi resolver. How a framework determines which native code paths should be selected is an implementation detail. The following section illustrates one way to determine the native path selection by pre-sorting the Bundle-NativeCode clauses and using the native.paths attributes:

5.5.1 Sort Native Code Clauses

The bundle native code selection algorithm specifies a priority order when multiple clauses match the native environment. To simplify the native code selection algorithm for the osgi.native namespace the Bundle-NativeCode clauses may be sorted using the priority order before converting the Bundle-NativeCode clauses to an osgi.native requirement. The sort order is specified by the section 3.10.1 Native Code Algorithm of the R5 Core specification:

The native code clauses are 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.

5.5.2 The native.paths Attribute

The Bundle-NativeCode header specifies a list of native paths for each native code clause. For example:

```
Bundle-NativeCode:
  lib/http.dll; lib/zlib.dll;
    osname=Windows95;
    osname=Windows98;
    osname=WindowsNT;
    processor=x86;
    selection-filter="(com.acme.windowing=win32)";
    language=en;
    language=se
```

This native code clause specifies the native path list "lib/http.dll, lib/zlib.dll". The native code paths may be specified in the osgi.native requirement using the attribute native.paths. The native.paths attribute must be of type List<String>. The above Bundle-NativeCode header would translate into the following osgi.native requirement:

```
Require-Capability:
  osgi.native;
    native.paths:List<String>="lib/http.dll,lib/zlib.dll";
    filter:="
      (&
        (|
          (osgi.native.osname~=windows95)
          (osgi.native.osname~=windows98)
          (osgi.native.osname~=windowsNT)
        )
        (osgi.native.processor~=x86)
        (|
          (osgi.native.language~=en)
          (osgi.native.language~=se)
        )
        (com.acme.windowing=win32)
      )"
```

The Bundle-NativeCode header may specify multiple lists of native code paths. For example:

```
Bundle-NativeCode:
  lib/http.dll; lib/zlib.dll;
    osname=Windows95;
    osname=Windows98;
    osname=WindowsNT;
    processor=x86;
    selection-filter = "(com.acme.windowing=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="(com.acme.windowing=gtk)"
```

Each native code path may be assigned a native.paths[.<index>] attribute value where <index> is the placement of the native code paths set in the Bundle-NativeCode header starting at 0. The above Bundle-NativeCode header is translated into the following osgi.native requirement:

```
Require-Capability:
  osgi.native;
    native.paths.0:List<String>="lib/http.dll,lib/zlib.dll";
    native.paths.1:List<String>="lib/solaris/libhttp.so";
    native.paths.2:List<String>="lib/linux/libhttp.sop.so";
    filter:="
      (|
        (&
          (|
            (osgi.native.osname~=Windows95)
            (osgi.native.osname~=Windows98)
            (osgi.native.osname~=WindowsNT)
          )
          (osgi.native.processor~=x86)
          (|
            (osgi.native.language~=en)
            (osgi.native.language~=se)
          )
          (com.acme.windowing=win32)
        )
        (&
          (|
            (osgi.native.osname~=Solaris)
            (osgi.native.osname~=Sunos)
          )
          (osgi.native.processor~=sparc)
        )
        (&
          (osgi.native.osname~=Linux)
          (osgi.native.processor~=mips)
          (com.acme.windowing=gtk)
```

```
    )
  )"
```

5.5.3 Selecting the Matching native.paths

After a bundle has been resolved the framework must select the native paths that apply to the native environment. The following rules may be used to make this selection.

1. If the requirement attribute native.paths exists then the List<String> value is used as the native paths for the bundle and native path selection is done. This is a case where the Bundle-NativeCode header only specified a single clause.

2. Split apart osgi.native filter directive into its sub-filter components which are OR'ed together. In this case the filter directive must have an OR type filter value. Assign each sub-filter an index from left most to right most sub-filter component starting at index 0.

3. Using sub-filter index as the order; for each sub-filter determine if it matches the attributes of the osgi.native capability which the osgi.native requirement got wired to. The index of the first sub-filter that matches is used as the index for the native.paths[.<index>] attribute.

Consider the following osgi.native.requirement requirement and the corresponding matching capability:

```
Provide-Capability:
  osgi.native;
    osgi.native.osname:List<String>="Linux";
    osgi.native.osversion:Version="3.5";
    osgi.native.processor:List<String>="x86-64,amd64,em64t,x86_64";
    osgi.native.language="en";
    com.acme.windowing="gtk"

Require-Capability:
  osgi.native;
    native.paths.0:List<String>="lib/http.dll,lib/zlib.dll";
    native.paths.1:List<String>="lib/solaris/libhttp.so";
    native.paths.2:List<String>="lib/linux/libhttp.sop.so";
    filter:="
      (|
        (&
          (osgi.native.osname~=Win32)
          (osgi.native.processor~=x86-64)
          (|
            (osgi.native.language~=en)
            (osgi.native.language~=se)
          )
          (com.acme.windowing=win32)
        )
        (&
          (|
            (osgi.native.osname~=Solaris)
            (osgi.native.osname~=Sunos)
          )
          (osgi.native.processor~=sparc)
        )
        (&
```

```
            (osgi.native.osname~=Linux)
            (osgi.native.processor~=x86-64)
            (com.acme.windowing=gtk)
        )
    )"
```

In this case the filter directive can be slit into the following 3 indexed filter components:

```
Index 0:
        (&
          (osgi.native.osname~=Win32)
          (osgi.native.processor~=x86-64)
          (|
            (osgi.native.language~=en)
            (osgi.native.language~=se)
          )
          (com.acme.windowing=win32)
        )
Index 1:
        (&
          (|
            (osgi.native.osname~=Solaris)
            (osgi.native.osname~=Sunos)
          )
          (osgi.native.processor~=sparc)
        )
Index 2:
        (&
          (osgi.native.osname~=Linux)
          (osgi.native.processor~=x86-64)
          (com.acme.windowing=gtk)
        )
```

Each filter is checked to see if it matches the capability in index order. Filter at index 0 does not match and filter at index 1 does not match, but filter at index 2 does. This indicates that the native.paths.2 attribute must be used to select the native paths. So in this example the native paths `lib/linux/libhttp.sop.so` will be used.

6   DTO

No new DTOs are needed. The osgi.native.namespace will be represented using the existing wiring APIs. The DTOs used to represent bundle wiring are sufficient.

7 Considered Alternatives

Considered representing each native code clause as a separate osgi.native requirement. Ultimately decided against this since it would require awkward special handling of the osgi.native namespace by the OSGi Resolver to somehow group the requirements together. The current proposal requires no changes to the OSGi Resolver specification.

8 Security Considerations

Currently no permission is required for a bundle with a Bundle-NativeCode header to resolve. Now that the osgi.native namespace is used to model a native environment requirement, a bundles with native code will be need to have the appropriate capability permission to require the osgi.native namespace. Otherwise the bundle will not be able to resolve if security is enabled.

A new implied Capability Permission REQUIRE for the osgi.native namespace must be granted to all bundles. This is similar to the implied Capability Permission REQUIRE for the osgi.ee namespace that got added in OSGi Core R5 when the specification added a conversion of Bundle-RequiredExecutionEnvironment to an osgi.ee namespace requirement.

9 Javadoc

OSGi Javadoc
11/12/12 3:22 PM

Package Summary		*Page*
org.osgi.framework.namespace	Namespace Package Version 1.1.	Error: Reference source not found

Package org.osgi.framework.namespace

Namespace Package Version 1.1.

See:
> [Description](#)

Class Summary		*Page*
[NativeNamespace](#)	Native Capability and Requirement Namespace.	Error: Reference source not found

Package org.osgi.framework.namespace Description
Namespace Package Version 1.1.

Bundles should not need to import this package at runtime since all the types in this package just contain constants for capability and requirement namespaces specified by the OSGi Alliance.

Class NativeNamespace
org.osgi.framework.namespace
```
java.lang.Object
  └org.osgi.resource.Namespace
      └org.osgi.framework.namespace.NativeNamespace
final public class NativeNamespace
extends org.osgi.resource.Namespace
```
Native Capability and Requirement Namespace.

This class defines the names for the attributes and directives for this namespace. All unspecified capability attributes are of type `String` and are used as arbitrary matching attributes for the capability. The values associated with the specified directive and attribute keys are of type `String`, unless otherwise indicated.

Version:

$Id: 641f0a04dfe88e1cdd729ce880c9c4c4f8d837de $

Immutable

Field Summary		*Page*
static String	**CAPABILITY_LANGUAGE_ATTRIBUTE** The capability attribute contains the `org.osgi.framework.language` launching property value.	rror: Reference source not found
static String	**CAPABILITY_OSNAME_ATTRIBUTE** The capability attribute contains alias values of the `org.osgi.framework.os.name` launching property value according to the [OSGi Specification References](#).	Error: Reference source not found
static String	**CAPABILITY_OSVERSION_ATTRIBUTE** The capability attribute contains a `Version` parsed from the `org.osgi.framework.os.version` launching property value.	Error: Reference source not found
static String	**CAPABILITY_PROCESSOR_ATTRIBUTE** The capability attribute contains alias values of the `org.osgi.framework.processor` launching property value according to the [OSGi Specification References](#).	Error: Reference source not found
static String	**NATIVE_NAMESPACE** Namespace name for native capabilities and requirements.	Error: Reference source not found

Fields inherited from class org.osgi.resource.Namespace
CAPABILITY_EFFECTIVE_DIRECTIVE, CAPABILITY_USES_DIRECTIVE, CARDINALITY_MULTIPLE, CARDINALITY_SINGLE, EFFECTIVE_ACTIVE, EFFECTIVE_RESOLVE, REQUIREMENT_CARDINALITY_DIRECTIVE, REQUIREMENT_EFFECTIVE_DIRECTIVE, REQUIREMENT_FILTER_DIRECTIVE, REQUIREMENT_RESOLUTION_DIRECTIVE, RESOLUTION_MANDATORY, RESOLUTION_OPTIONAL

Field Detail

NATIVE_NAMESPACE

```
public static final String NATIVE_NAMESPACE = "osgi.native"
```
Namespace name for native capabilities and requirements.

CAPABILITY_OSNAME_ATTRIBUTE

```
public static final String CAPABILITY_OSNAME_ATTRIBUTE = "osgi.native.osname"
```

The capability attribute contains alias values of the `org.osgi.framework.os.name` launching property value according to the [OSGi Specification References](). The value of this attribute must be of type `List<String>`.

CAPABILITY_OSVERSION_ATTRIBUTE

`public static final String CAPABILITY_OSVERSION_ATTRIBUTE = "osgi.native.osversion"`

The capability attribute contains a `Version` parsed from the `org.osgi.framework.os.version` launching property value. The value of this attribute must be of type `Version`.

CAPABILITY_PROCESSOR_ATTRIBUTE

`public static final String CAPABILITY_PROCESSOR_ATTRIBUTE = "osgi.native.processor"`

The capability attribute contains alias values of the `org.osgi.framework.processor` launching property value according to the [OSGi Specification References](). The value of this attribute must be of type `List<String>`.

CAPABILITY_LANGUAGE_ATTRIBUTE

`public static final String CAPABILITY_LANGUAGE_ATTRIBUTE = "osgi.native.language"`

The capability attribute contains the `org.osgi.framework.language` launching property value. The value of this attribute must be of type `String`.

---

10 Document Support

10.1 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

10.2 Author's Address

Name	
Company	
Address	
Voice	
e-mail	

10.3 Acronyms and Abbreviations

10.4 End of Document