# RFC 201 - Subsystems Update

Draft

45 Pages

## Abstract

*This RFC proposes various updates to the Subsystem Service Specification.*

# 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 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 | *9th April 2013* | *Created*<br>*Graham Charters, IBM (charters@uk.ibm.com)* |
| 0.1 | *7th June 2013* | *Updates to domain, problem desc, service dep calc, etc, based on last telecon review.*<br>*Graham Charters, IBM (charters@uk.ibm.com)* |
| 0.2 | *7th June 2013* | *Added sections 5.5 – 5.7.*<br>*John Ross, IBM (jwross@us.ibm.com)* |
| 0.3 | *7th June 2013* | *Added Configuration section*<br>*Tom Watson, IBM (tjwatson@us.ibm.com)* |
| 0.4 | *17th June 2013* | *Accepted all changes as of the Jun 13 OSGi F2F.*<br>*Updated sections 3.2 and 5.5 through 5.7 based on F2F feedback.*<br>*John Ross, IBM (jwross@us.ibm.com)* |
| 0.5 | *27th June 2013* | *Accepted all changes reviewed on the 26th June 2013 EEG call.*<br>*Added the text of the comment in 5.5 regarding outstanding issues with woven dynamic imports becoming part of the sharing policy into the document. Deleted the comment.*<br>*Added the text of the comment in 5.5 regarding why DynamicImport-Package was not addressed to the Considered Alternatives chapter. Deleted the comment.*<br>*Added the text of the comment in 5.7 regarding SubsystemDTO to the Considered Alternatives section. Deleted the comment.*<br>*Added section 3.10 based on bug 2537.*<br>*John Ross, IBM (jwross@us.ibm.com)* |
| 0.6 | *29th October 2013* | *Minor edits.*<br>*Created javadoc and linked to RFC.*<br>*Updated table of contents.*<br>*John Ross, IBM (jwross@us.ibm.com)* |
| 0.7 | *30th October 2013* | *Add abstract.*<br>*Add descriptions to information subsystem headers.*<br>*Update javadoc with informational subsystem headers.*<br>*John Ross, IBM (jwross@us.ibm.com)* |

| Revision | Date | Comments |
|---|---|---|
| 0.8 | *5th November 2013* | *Minor edits.*<br>*Delete all information associated with handling the DynamicImport-Package header (requirement SSU025). Move to Considered Alternatives.*<br>*Delete all information associated with the TCCL (requirement SSU030, sections 2.1, 3.3, and 5.3). Move to Considered Alternatives.*<br>*Delete all information associated with Subsystem Configuration (requirements SSU070 and SSU075, sections 3.7, 5.8, and 5.9). Move to Considered Alternatives.*<br>*Delete all information associated with Manifest Matching Rules (section 3.8). Move to Considered Alternatives.*<br>*Delete comments associated with the previous topics. Move to Considered Alternatives.*<br>*John Ross, IBM (jwross@us.ibm.com)* |
| 0.9 | *14th November 2013* | *Accept all changes discussed at November F2F.*<br>*Reformatting and reorganization with no content changes (untracked).*<br>*Update revision history.*<br>*Remove section 3.6.*<br>*Remove requirement SSU026.*<br>*Add description of syntax requirements for informational headers to section 5.1.4.*<br>*Add chosen solution for bug 2537 to section 5.6.*<br>*Add security considerations.*<br>*Add authors.*<br>*John Ross, IBM (jwross@us.ibm.com)* |

# 1 Introduction

The Subsystems specification defines how to describe and provision bundle collections into an OSGi framework. Subsystem types have different sharing policies enabling a variety of use cases (e.g. isolated applications running in an application server, or non-isolated features used to assemble a middleware product).

It was not possible to satisfy all requirements in the first version of the specification and as Subsystems has been used in the wild, new requirements have also come to light. This RFC addresses a prioritized set of requirements in the form of a minor update to the Subsystems specification.

# 2 Application Domain

The application domain for this RFC is the use of Subsystem 1.0. Further background can be found in the application domain of the subsystem 1.0 RFC (RFC 152). Subsystem 1.0 was first standardized in OSGi Enterprise R5. A few requirements that were known at the time were not addressed due to specification schedules, and since the release of R5, user experience has resulted in new requirements being raised. This RFC is intended to gather these requirements together into a 1.1 update to the Subsystem specification.

# 3 Problem Description

## 3.1 Bug 2251 – Translation of Subsystem Headers

Products that support Subsystem for applications or runtime features may have tools that display information about available or installed Subsystems, and so it should be possible to translate the human readable headers (e.g. Subsystem-Name, Subsystem-Description).

In addition to translating the existing headers, we should consider adding more of the informational headers that are available to bundles, such as Bundle-License.

## 3.2 Bug 2430 - Subsystems spec needs to account for Weaving Service

The Core R5 spec section 56, "Weaving Hook Service Specification" defines a whiteboard service by which weaving hooks can introduce new package dependencies into java byte code as it is loaded. Section 56.3 states, "These dynamically added dependencies are made visible through the Bundle Wiring API Specification on page 133 as new requirements."

The Enterprise R5 spec section 134 introduces the "Subsystem Service Specification." Section 134 does not explicitly discuss how new requirements introduced by weaving hooks should be handled. Woven requirements are introduced at run time, often after a deployment has been precalculated.

Overall, a design is required that handles dependencies introduced dynamically via weaving hooks. What we need out of this bug is a standard mechanism by which a Scoped Subsystem's resolver hooks can detect and permit dynamic requirements to be allowed into that subsystem.

Section 134.6, "Determining dependencies" should explicitly address how dynamically introduced dependencies should be handled.

Section 134.15.5 should state how a Scoped Subsystem's associated Region's sharing policy will permit packages matching dynamically introduced dependencies to be allowed into the Scoped Subsystem.

Both sections 134.16.2.1 and 134.16.3.2 should be updated to account for dynamic (woven) requirements.

See RFC 191 – Weaving Hook Enhancements - this was created as a pre-requisite for enabling weavers to enhance bundles inside scoped subsystems.

## 3.3 Bug 2270 - Subsystems should allow a Deployment.MF to be provided separately

As a Subsystem moves through the software life-cycle, from development, through QA testing and into production, it is often necessary to be able to lock down the deployment to ensure what goes into production is exactly what was tested. A deployment is locked down using a deployment manifest which must be included in the subsystem archive (.esa). However, a deployment is likely to be locked down after the initial subsystem archive has been created and so in some circumstances it is desirable to not have to crack open the archive file in order to put the deployment manifest in. Instead it should be possible to install a subsystem and provide its deployment manifest separately in the same installation step.

## 3.4 Bug 2211 – Subsystem API to provide access to more information

Subsystems 1.0 only provides very limited access to subsystem information through the API. At a minimum, we need to provide access to the deployment manifest as this is potentially generated by the Subsystem runtime and a management agent may need it to reason about a deployed subsystem.

## 3.5 Bug 2081 - Determining service dependencies for application subsystems

Subsystems 1.0 requires scoped subsystem resolution to prefer services provided by the contents of the subsystem. It states that the services provided and required by bundles may be derived from DS or Blueprint, but does not provide a normative approach to declaring services that is portable and component model agnostic.

Enterprise OSGi R5 defines the osgi.service namespace for declaring service capabilities and requirements. This mechanism could be standardized as a component model agnostic complement to the use of DS and Blueprint for determining service dependencies during subsystem provisioning.

needs more spec clarification. Main thing is to use the osgi.service capability with an effective attribute other than 'resolve'. How the capability/requirement is added to the bundle can be left to a tool or can be done directly by the developer. In DS/Blueprint cases this metadata can be inferred from the xml file, but how thats done is left up to a tool.

## 3.6 ~~Exposing services outside an Application Subsystem~~

~~See bug: https://www.osgi.org/members/bugzilla/show_bug.cgi?id=2506~~

~~Application Subsystems isolate all packages and services from use outside the application. Users have expressed a need to be able to selectively allows a subset of the application's services to be accessible outside the Subsystem~~.

## 3.7 Bug 2537 - Preferred-Provider not always a cure for uses constraint violation.

The issue is highlighted by the following simplified scenario.

```
- Root Subsystem

    - Bundle A - Export-Package: foo,bar;uses:=foo

    - Child Subsystem

        - Bundle B - Export-Package: foo
```

```
- Bundle C - Import-Package: foo,bar
```

Based on the current specification, a uses constraint violation will always result from this scenario, even if Bundle A is listed as a Preferred-Provider. Content resources are strictly favored (i.e. if a matching capability is found in the Content Repository, the search must stop, so the resolver is not given Bundle A as an option) so Bundle C will always get wired to Bundle B for package foo. Since the only provider of package bar is Bundle A, a uses constraint violation results.

Note that the solution, if any, could not simply be to reverse the current Content followed by Preferred Provider repository lookup. If Preferred Provider came first, you'd encounter the same issue by switching the positions of Bundle A and Bundle B. The solution would need to include the capabilities from both the Content and Preferred Provider repositories in the list of capabilities returned to the resolver.

# 4 Requirements

SSU010 – The specification MUST make it possible to provide translations for all human readable Subsystem-Headers and package these inside a Subsystem archive file.

SSU015 – The specification MUST define a standard set of informational headers similar to those available for bundles (e.g. Bundle-License).

SSU020 – The specification MUST make it possible for a weaving hook to add new package imports to a scoped subsystem's sharing policy in order to allow resolution of any new package dependencies it has woven into a subsystem content bundle.

~~SSU026 – The specification MUST clarify the behaviour in the presence of optional package import headers defined on content bundles inside scoped Subsystems.~~

SSU040 – The specification MUST make it possible to install a subsystem through a subsystem archive and provide its deployment manifest separately, but as part of the same installation step.

SSU050 – The specification MUST make it possible to retrieve a subsystem's deployment manifest through the subsystem API. The deployment manifest MAY contain additional information over and above what was provided during installation.

SSU80 – The specification SHOULD make it possible to selectively access the services of an application subsystem from outside the application.

# 5 Technical Solution

## 5.1 Translation of Subsystem Headers

For consistency and ease of comprehension, the design for localizing subsystem manifest headers follows the approach used by bundles.

### 5.1.1 Subsystem-Localization Header

The subsystem localization header identifies the default base name of the localization properties files contained in the subsystem archive. The default value is OSGI-INF/l10n/subsystem. Translations are therefore, by default, OSGI-INF/l10n/subsystem_de.properties, OSGI-INF/l10n/subsystem_nl.properties, and so on. An example of the header that specifies the default is:

```
Subsystem-Localization:  OSGI-INF/l10n/subsystem
```

The location is relative to the root of the subsystem archive.

### 5.1.2 Localization Properties

A localization entry contains key/value entries for localized information. All headers in a subsystem's manifest can be localized. However, the subsystems implementation must always use the non-localized versions of headers that have subsystem semantics.

A localization key can be specified as the value of a subsystem's manifest header using the following syntax:

```
header-value ::= '%'text
text ::= < any value which is both a valid manifest header value and a valid
property key name >
```

For example, consider the following subsystem manifest entries:

```
Subsystem-Name: %acme subsystem
Subsystem-Description: %acme description
Subsystem-SymbolicName: acme.Subsystem
Acme-Defined-Header: %acme special header
```

User-defined headers can also be localized. Spaces in the localization keys are explicitly allowed.

The previous example manifest entries could be localized by the following entries in the manifest localization entry OSGI-INF/l10n/subsystem.properties.

```
# subsystem.properties
acme\ subsystem=The ACME Subsystem
acme\ description=The ACME Subsystem provides all of the ACME \ services
acme\ special\ header=user-defined Acme Data
```

The above manifest entries could also have French localizations in the manifest localization entry

```
OSGI-INF/l10n/subsystem_fr_FR.properties.
```

### 5.1.3 Locating localization entries

The Subsystems implementation must search for localization entries by appending suffixes to the localization base name according to a specified locale and finally appending the .properties suffix. If a translation is not found, the locale must be made more generic by first removing the variant, then the country and finally the language until

an entry is found that contains a valid translation. For example, looking up a translation for the locale en_GB_welsh will search in the following order:

```
OSGI-INF/l10n/subsystem_en_GB_welsh.properties
OSGI-INF/l10n/subsystem_en_GB.properties
OSGI-INF/l10n/subsystem_en.properties
OSGI-INF/l10n/subsystem.properties
```

### 5.1.4 Informational Subsystem Headers

All of the following headers have bundle manifest analogs and will have the same syntax restrictions as described in section 3.2.1 of the OSGi Core Specification.

#### *5.1.4.1 Subsystem-Category*

Manifest header identifying the categories of the subsystem as a comma-delimited list.

#### *5.1.4.2 Subsystem-Copyright*

Manifest header identifying the subsystem's copyright information.

#### *5.1.4.3 Subsystem-DocURL*

Manifest header identifying the subsystem's documentation URL, from which further information about the subsystem may be obtained

#### *5.1.4.4 Subsystem-License*

Manifest header identifying the subsystem's license.

#### *5.1.4.5 Subsystem-Vendor*

Manifest header identifying the subsystem's vendor.

#### *5.1.4.6 Subsystem-ContactAddress*

Manifest header identifying the contact address where problems with the subsystem may be reported; for example, an email address.

#### *5.1.4.7 Subsystem-Icon*

Manifest header identifying the icon URL for the subsystem.

## 5.2 Weaving Hooks

This section describes the technical solution for handling dynamic imports added by weaving hooks.

Dynamic package imports added by weaving hooks are observed by registering a WovenClassListener service and receiving notifications via WovenClassListener.modified(WovenClass). If the woven class is in the TRANSFORMED state [1], the bundle containing the woven class is obtained by calling WovenClass.getBundleWiring().getBundle(). The scoped subsystem, if any [2], containing the bundle as a constituent is retrieved. If necessary [3], the subsystem is resolved [4]. For each dynamic import, if necessary [5], the subsystem's sharing policy is updated [6].

[1] The sharing policy must be updated while the woven class is in the TRANSFORMED state so that it takes effect before the bundle wiring is updated during the transition to DEFINED; otherwise, the class would fail to load.

[2] A bundle might be a constituent of multiple subsystems, but never more than one scoped subsystem. The rest are features, which have no sharing policies to update. It's possible the bundle will not be a constituent of a scoped subsystem.

[3] It's possible for a classload request to occur on a bundle in an unresolved subsystem because the framework is free to resolve bundles whenever it desires. A resolved bundle can potentially receive a classload request. For example, a BundleEventListener registered with the system bundle context could receive the RESOLVED event and, for whatever reason, load a class. Also, a resolved bundle in an unresolved feature might get wired to another bundle.

[4] The subsystem must be resolved in order to guarantee the dynamic imports will not effect the resolution and, therefore, potentially create a wiring inconsistent with the deployment manifest.

[5] The sharing policy is only updated if the dynamic import cannot be completely satisfied from within the subsystem. Note that all dynamic imports with a wildcard must always be added to the sharing policy.

[6] We encounter the same issue here as in supporting the DIP header in terms of what happens when bundles are refreshed or a child subsystem is installed into an already resolved parent. It's possible that the runtime wiring will be inconsistent with the deployment manifest. We punt on specifying this under the assumptions that (1) weaving hooks are scant and (2) weaving hook providers really know what they're doing and will minimize or altogether avoid the use of wildcards in dynamic imports. We therefore consider any issues to be unlikely and rare.

## 5.3 Allow Deployment Manifest to be Provided Separately

A new method is added to the Subsystem interface with the following signature.

public Subsystem install(String location, InputStream content, InputStream deploymentManifest);

This method installs a subsystem using the provided deployment manifest instead of the one in the archive or the computed one. If the deployment manifest is null, the behavior is exactly the same as in the install(String, InputStream) method. Implementations must support input streams in the format described by section 134.2 of the Subsystem Service Specification. If the deployment manifest does not conform to the subsystem manifest (see 134.15.2), the installation fails and a SubsystemException is thrown. If the input stream throws an IOException, the installation fails and a SubsystemException is thrown with the IOException as the cause.

## 5.4 Subsystem API to provide access to more information

A new method is added to the Subsystem interface with the following signature.

public Map<String, String> getDeploymentHeaders();

The method follows the same rules as described for the getSubsystemHeaders(Locale) method except that no Locale is accepted since there are currently no translatable headers defined.

## 5.5 Application Subsystem Service Dependencies

Section 134.16.2.2 of the Subsystem 1.0 specification states the following regarding Service Imports:

“Application resolution is required to prefer services provided by content bundles over those provided outside the application. For this reason, the application Subsystem sharing policy only imports services required by the Subsystem's content bundles that are not also provided by the content bundles. There is no standard way to determine this, but a Subsystem runtime is permitted to use its own means to determine the services to import. Examples include resource metadata from a bundle repository, or analysis of bundle contents (e.g. Blueprint or Declarative Service configurations). A deployment manifest for an application Subsystem would list these service imports using the Subsystem- ImportService header.”

This specification provides a means of declaratively identifying the services a bundle provides using the Provide-Capability and Require-Capability headers with the osgi.service namespace (as described in Enterprise OSGi R5, section 135.4).

An example of a bundle providing the service and declaring it using the Provide-Capability header is as follows:

```
Provide-Capability: osgi.service;
 objectClass=com.foo.MyService;
 a.service.prop=SomePropertyValue
```

An example of a bundle requiring a service and declaring the requirement using the Require-Capability header is as follows:

```
Require-Capability: osgi.serivce;
 filter:=”(&(objectClass=com.foo.MyService)(a.service.prop=SomePropertyValue))”
```

These headers can be hand-written, for example to declare programmatic use of an OSGi service, or can be generate by a tool, such as BND, based on declarative component model configuration, e.g. Declarative Services or Blueprint. A subsystem implementation must assume that if these headers are present in a bundle, they declare all the service dependencies of that bundle. The implementation must therefore not search the bundle for additional dependencies from other sources, such as contained Blueprint or DS XMLs.

## 5.6 Bug 2517 - Preferred-Provider not always a cure for uses constraint violation.

The following sections represent possible solutions to the issue.

### 5.6.1 Nothing

Do nothing. This is working as designed.

### 5.6.2 Content + Preferred Provider Repository

During the subsystem install process, while resolving the content of the subsystem (offline), change the steps outlined in section 134.6 - "Determining Dependences" to state that the search continues from step 1 "The Content Repository" to step 2 "The Preferred Repository" even if a capability is found in the content repository. The subsystem must prefer capabilities from the content repository, but should allow capabilities from the preferred repository if they can be used to solve an inconsistent class space (uses) issue.

### 5.6.3 Content + Preferred Provider + System Repository

While option 2 gives the developer an out to work around uses constraint issues when providers are available in the system repository, it still requires developer intervention to add a Preferred-Provider header. Also, preferred providers can only be used against providers that are contained in the parent subsystem (not any higher in the hierarchy). This may not always be possible since existing providers may pre-exist higher up in the hierarchy. An extension of option 2 is to also change step 2 "The Preferred Repository" to state that the search continues from step 2 "The Preferred Repository" to step 3 "The System Repository" if and only if no matching capabilities are found in the preferred repository. The subsystem must prefer capabilities from the content repository, but should allow capabilities from the system repository if they can be used to solve an inconsistent class space (uses) issue.

### 5.6.4 Content + Preferred Provider + System + Local Repository

Option 3 with an additional step. If option 3, step 3 is executed, proceed to step 4 “The Local Repository”, even if capabilities were found in the system repository.

### 5.6.5 All Repositories

Even more radical: Always continue on to the next step even when capabilities are found for all repositories. This is most likely a bad thing to do and has the potential to pull in the world from all registered Repository services unnecessarily during a resolve operation.

Section 5.6.4 is the chosen solution. The Resolver Service Specification makes the following guarantee.

*"The Resolver will treat the order of the capabilities as preferences, the first element is more preferred than a later element. The Resolver cannot guarantee that the wiring obeys this preference since there can be other*

*constraints. However, a Resolver must use this preference order for simple cases and try to use it in more constrained situations."*

This is deemed sufficient for the purposes of subsystems. All matching capabilities from the Content, Preferred Provider, System, and Local repositories, and in that order, MUST be provided to the resolver when attempting to resolve a subsystem. An allowable implementation is to return all matching capabilities in the right order within the context of a single call to ResolveContext.findProviders. Furthermore, implementations MAY also return matching capabilities from Repository Services.

# 6 Considered Alternatives

## 6.1 Weaving Hooks and DynamicImport-Package

SSU025 – The specification MUST clarify the behaviour in the present of Dynamic-ImportPackage headers defined on content bundles inside scoped Subsystems.

The DynamicImport-Package header will not be addressed by this RFC. For the below reasons, we should wait for a compelling need to arise before attempting to tackle it.

Although it is straightforward to specify well-defined behavior when installing and starting subsystems [1], complexity rapidly ensues when considering what happens when bundles are refreshed, or when children are installed into an already resolved parent subsystem, and how the runtime wiring faithfully represents the deployment manifest [2].

Another concern is that computing the DIP header for applications is not binary compatible [2]. Some sort of opt-in mechanism would need to be devised.

[1] The DIP header would have the same effect as dynamic package imports added by weaving hooks. That is, the imports would only take affect after the subsystem was resolved. This can be accomplished by specifying that the DIP header is only applied to the sharing policy just before entering the STARTING state.

[2] For example, a requirement of a content resource might get wired to an external resource when a local one was preferred.

## 6.2 Subsystem API to provide more access to information

In addition to the new getDeploymentHeaders() method, defining a SubsystemDTO was considered. It was decided to leave this for the future since nobody was currently asking for it.

### What about provisioning decision information? If we do this, we should tie it into some kind of DTO (dumps the internal info into a DTO).

## 6.3 Bug 1916 – Define the TCCL for Scoped Subsystems

SSU030 – The specification SHOULD make it possible to define a TCCL policy for proxied interactions between bundle inside a subsystem and for calls out of a subsystem.

### Consider service loader design. This could be done just as a blueprint spec addition. Will depend on whether we went the Virgo route for context bundles. Action: move to Blueprint 1.1?

Bug 1916 summaries the requirement for TCCL support in scoped Subsystems. Essentially, certain legacy libraries, such as Hibernate, depend on using the thread context class loader (TCCL) to load application types.

Independently of the mechanism used to define the TCCL, the precise conditions which cause the TCCL to be set on a call out of the scoped application must also be specified. This may depend on whether a service is called or an exported type used directly.

### David suggested there may be some useful info in the Service Loader spec for TCCL.
### Solutions today rely on proxying by component models (e.g. Spring or Blueprint). They also vary in design.
We could standardize something that allows the existing options/approaches. We could try to standardize something lower level that allows more aggressive proxying. Or we could punt, for now...

## 6.4 TCCL Reliance (from Application Domain section)

Libraries not originally designed for use in OSGi can often rely on the use of Thread Context Class Loaders. RFC 133 sought to provide a general solution to the problem and documented the Eclipse Buddy Classloading design. Unfortunately, no worthy solution was identified and so this effort stalled. Approaches based around the concept of isolated applications (precursors to Application Subsystems) have been employed in WebSphere Application Server and Eclipse Virgo. These do not try to be general solutions, but simply seek to provide a TCCL in a deterministic fashion within the context of a scoped application. At a high level, the two approaches are:

1. Synthetic Context Bundle: When calling out of a scoped application, Virgo sets the TCCL to be the class loader of a specially created bundle which imports all the packages exported by bundles in the scope in question.

   All types which need to be available to the TCCL must therefore have their packages exported. This places a small restriction on the application: no two bundles in the scope may export the same package.

   See this blog for another description: http://underlap.blogspot.com/2011/02/thread-context-class-loading-in-virgo.html.
2. Calling Bundle: WebSphere Application Server sets the TCCL to be the class loader of the calling bundle for all calls managed by Blueprint. All types which need to be available to the TCCL must either be defined by the calling bundle or must be exported by the bundle defining the type and imported by the calling bundle.

   This approach aims to give the application developer control over the TCCL without imposing restrictions or overly compromising the application modularity.

important to standardize the TCCL approach or approaches available in Subsystems as the lack of a standard affects portability. For example, an application that relies on the visibility provided by Eclipse Virgo may not have access to all the types it requires when deployed to WebSphere Application Server.

## 6.5 Subsystem Configuration

### 6.5.1 Requirements

SSU070 – The specification SHOULD make it possible to configure the bundles of a subsystem via configuration admin (e.g. provide a standard form for the bundle locations inside a subsystem).

SSU075 – The specification SHOULD make it possible to provide configuration resources as part of a subsystem archive where those resources are intended to configure bundles of the subsystem, or any bundles within the same region. The will require some means of selecting which configuration admin to use (e.g. one inside the subsystem region).

### 6.5.2 Problem Description

Configuration of subsystems via configuration admin had to be dropped from Subsystems 1.0 due to time constraints. Time permitting, this RFC should consider two things:

1. How to identify targets of configuration within a subsystem such that configurations provided outside the subsystem can be delivered to the right target within a subsystem.
2. How to provide configurations as resources in a subsystem archive. These configuration should only apply to targets within the subsystem that contributed them or any targets that are in the same containing region.

### 6.5.3 Technical Solution

The Configuration Admin service is an important aspect of the deployment of an OSGi framework. The Subsystems 1.0 specification did not address how configuration data can be received by bundles which are deployed in a subsystem. The following are the important entities from Configuration Admin that need to be considered when dealing with configuring bundles deployed in a subsystem.

- Configuration Target - The target service that will receive the configuration information. For services, there are two types of targets: ManagedServiceFactory or ManagedService objects. Extenders may also define additional configuration targets. For example, Declarative Services specifies a way service components receive configuration data without requiring the registration of ManagedServiceFactory or ManagedService services.
  - Subsystem Configuration Target – The scoped subsystem which contains the Configuration Target.
- Configuring Bundle – A bundle that modifies the configuration information through the Configuration Admin service. This bundle is either a management bundle or the bundle for which the configuration information is intended.
  - For subsystems the configuring bundle may be deployed in Subsystem Configuration Target but this is not required. The configuring bundle may also live outside of the Subsystem Configuration Target. If configuration resources are included in a subsystem then the configuring bundle likely will be the subsystems implementation itself.
- Configuration Extenders – An Extender, such as Declarative Services, that is responsible for delivering configuration data to the entities they extend, such as service components.
  - For subsystems special consideration is needed allow an extender to deliver configuration data. For example, the Declarative Service specification mandates that Configuration objects must be obtained from the Configuration Admin service using the Bundle Context of the bundle containing the component (i.e. the Configuration Target). This implies that the ConfigurationAdmin service must be available within the context of the Subsystem Configuration Target.
- Targeted PID – Specially formatted PIDs that are interpreted by the Configuration Admin service. The target PID scopes the applicability of the PID to a limited set of target bundles.
  - For subsystems a schema is needed to allow configurations to be targeted for specific bundles deployed in scoped subsystems. This important because scoped subsystems provide isolation to a group of bundles. Configurations intended for one Subsystem Configuration Target should not affect targets contained in other Subsystem Configuration Targets.

#### *6.5.3.1 Subsystem Configuration Target*

A Configuration Target must not need to be aware of the Subsystem Configuration Target in which they are deployed. Configuration Targets must be able to receive configuration data in the same way regardless of the way they are deployed (in a subsystem or directly into a framwork). In order to allow configuring bundles to identify Subsystem Configuration Targets appropriately there needs to be a specified way of identifying which

Subsystem Configuration Target a bundle belongs to. To do this a bundle location schema is defined for bundles that are deployed by the subsystems implementation:

```
subsystem-location '/!' symbolic-name @ 'version'
```

Where subsystem-location is the subsystem location for the Subsystem Configuration Target, symbolic-name is the symbolic of the bundle and version is the version of the bundle. Except for the subsystem region context bundle, all bundles deployed by the subsystems implementation must use this bundle location schema when installing bundles which belong to a subsystem.

### 6.5.4 Delivering Configuration Data

Scoped subsystems provide isolation which allows for a bundle to be deployed multiple times within the same framework instance. This implies that the same Configuration Target may have multiple instances in the same framework instance. In order to deliver configuration data to Subsystem Configuration Targets appropriately a configuring bundle should use fully qualified targeted PIDs. This will ensure that the configuration data only gets delivered to the appropriate Configuration Target.

For example, a configuration target is registered with the PID com.example.web.WebConf by a bundle with the symbolic-name com.acme.example and version 3.2.0 deployed to a scoped subsystem with the location ApplicationX. The following would be the fully qualified target pid to use:

```
com.example.web.WebConf|com.acme.example|3.2.0|ApplicationX
```

XXX - An alternative is to use the concept of Regions from the Configuration Admin service specification. This option requires the use of a security manager to provide isolation. Should we mandate that subsystems implementations grant the ConfigurationPermission[“?<subsystem-location>”, TARGET] permission to all bundles.

#### *6.5.4.1 Configuration Target Visibility*

In order for a Configuration Admin Service implementation to deliver configuration data to configuration targets it must have access to ManagedServiceFactory and ManagedService service registrations contained within a scoped subsystems. In most cases the Configuration Admin implementation will not be contained in the Subsystem Configuration Target and will not have visibility to the necessary services.

XXX – There are two ways I can think of to accomplish this:

1. Have a configuration admin implementation that is subsystems aware. All this means is that is uses the system bundle context in order to discover all MSF and MS services no matter what scope they live in.
2. Have the subsystems implementation somehow grant the configuration admin implementation bundle access to all MSF and MS service registrations.

My preference is to use option 1 since it would not require any strange special holes to be poked through the subsystem sharing policy to expose specific services to specific bundles.

### 6.5.5 Configuration Admin Service Visibility

In some scenarios it is required that the configuration target has access to the Configuration Admin Service. For example, if a bundle manages its own configuration data. An extender is also able to use the Configuration Admin Service database to retrieve and deliver configuration data to the components it is extending. Declarative Services is one such example. The SCR must use the Configuration Target bundle's context in order to retrieve configuration data from the Configuration Admin service. This is to allow for proper security checks and also to allow for accurate matching of targeted PIDs.

In order for these scenarios to work the subsystems implementation must implicitly import the ConfigurationAdmin service into each scoped subsystem. The one exception is if the scoped subsystem is a composite subsystem which exports a Configuration Admin service (XXX – not really sure on this one, perhaps we do not really need to special cases this, would it be bad to export and implicitly import the same service?)

## 6.6 Configuration Resources

XXX – Left as a brain storming session for the F2F ;-)

Need to allow for some configuration resources. Initial though is to have the resources themselves encode the configuration target in their file name:

```
<bundle-symbolic-name>@<bundle-version>@<PID>.cfg
```

The cfg file would be a simple properties file where the key is the configuration key and the value is the configuration key value. Perhaps it uses the syntax for capability attributes from the Provide-Capability header except we would have to substitute the ':' type separator with another char (maybe '@'). For example:

```
ports@List<Long>=1,2,3

server.name=Acme Server
```

The subsystems implementation would read these cfg resources and populate the configuration admin database with the proper targeted PID configurations. May also need to specify a new resource type of osgi.subsystem.configuration that can have a proper osgi identity and live in a repository and be included in the Subsystem-Content header.

## 6.7 Manifest matching rules

It's unclear how to match a Deployment.MF to a subsystem in all cases. Although the Symbolic Name and Version are required for the Deployment.MF, they are not required for Subsystem.MF. How do we match them when not specified in the Subsystem.MF? One possibility would be to try to match against defaults and if they do not match, fail the deployment?

### This could come down to a spec clarification. What's the default symbolic name.

# 7 Security Considerations

~~*Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented*~~.

*The new Subsystem.install(String location, InputStream content, InputStream deploymentManifest) method will inherit the same permission requirements from the other install methods, namely* SubsystemPermission[installed subsystem,LIFECYCLE].

*The new Subsystem.getDeploymentHeaders() method will require the* SubsystemPermission[installed subsystem,METADATA] permission.

# 8 Javadoc

# OSGi Javadoc

10/30/13 7:24 AM

| Package Summary | | Page |
|---|---|---|
| org.osgi.service.subsystem | Subsystem Service Package Version 1.1. | 21 |

# Package org.osgi.service.subsystem

```
@org.osgi.annotation.versioning.Version(value="1.1")
```

Subsystem Service Package Version 1.1.

**See:**

Description

| Interface Summary | | Page |
|---|---|---|
| ***Subsystem*** | A subsystem is a collection of resources constituting a logical, possibly isolated, unit of functionality. | *22* |

| Class Summary | | Page |
|---|---|---|
| **SubsystemConstants** | Defines the constants used by Subsystem service property, manifest header, attribute and directive keys. | *38* |

| Enum Summary | | Page |
|---|---|---|
| **Subsystem.State** | An enumeration of the possible states of a subsystem. | *34* |

# Package org.osgi.service.subsystem Description

Subsystem Service Package Version 1.1.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.subsystem; version="[1.1,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.subsystem; version="[1.1,1.2)"
```

# Interface Subsystem

**org.osgi.service.subsystem**

---

```
@org.osgi.annotation.versioning.ProviderType
public interface Subsystem
```

A subsystem is a collection of resources constituting a logical, possibly isolated, unit of functionality.

A subsystem may be *scoped* or *unscoped*. Scoped subsystems are isolated by implicit or explicit sharing policies. Unscoped subsystems are not isolated and, therefore, have no sharing policy. There are three standard `types` of subsystems.

1. `Application` - An implicitly scoped subsystem. Nothing is exported, and imports are computed based on any unsatisfied content requirements.
2. `Composite` - An explicitly scoped subsystem. The sharing policy is defined by metadata within the subsystem archive.
3. `Feature` - An unscoped subsystem.

Conceptually, a subsystem may be thought of as existing in an isolated region along with zero or more other subsystems. Each region has one and only one scoped subsystem, which dictates the sharing policy. The region may, however, have many unscoped subsystems. It is, therefore, possible to have shared constituents across multiple subsystems within a region. Associated with each region is a bundle whose context may be `retrieved` from any subsystem within that region. This context may be used to monitor activity occurring within the region.

A subsystem may have `children` and, unless it's the root subsystem, must have at least one `parent`. Subsystems become children of the subsystem in which they are installed. Unscoped subsystems have more than one parent if they are installed in more than one subsystem within the same region. The subsystem graph may be thought of as an acyclic digraph with one and only one source vertex, which is the root subsystem. The edges have the child as the head and parent as the tail.

A subsystem has several identifiers.

1. `Location` - An identifier specified by the client as part of installation. It is guaranteed to be unique within the same framework.
2. `ID` - An identifier generated by the implementation as part of installation. It is guaranteed to be unique within the same framework.
3. `Symbolic Name`/`Version` - The combination of symbolic name and version is guaranteed to be unique within the same region. Although `type` is not formally part of the identity, two subsystems with the same symbolic names and versions but different types are not considered to be equal.

A subsystem has a well-defined `life cycle`. Which stage a subsystem is in may be obtained from the subsystem's `state` and is dependent on which life cycle operation is currently active or was last invoked.

A subsystem archive is a ZIP file having an `.esa` extension and containing metadata describing the subsystem. The form of the metadata may be a subsystem or deployment manifest, as well as any content resource files. The manifests are optional and will be computed if not present. The subsystem manifest headers may be `retrieved` in raw or localized forms. There are five standard `types` of resources that may be included in a subsystem.

1. `Bundle` - A bundle that is not a fragment.
2. `Fragment` - A fragment bundle.
3. `Application Subsystem` - An application subsystem.
4. `Composite Subsystem` - A composite subsystem.
5. `Feature Subsystem` - A feature subsystem.

Resources contained by a subsystem are called `constituents`. There are several ways a resource may become a constituent of a subsystem:

- A resource is listed as part of the subsystem's content.
- A subsystem resource is a child of the subsystem.

- The subsystem has a provision policy of accept dependencies.
- A bundle resource is installed using the region bundle context.
- A bundle resource is installed using the bundle context of another resource contained by the subsystem.

In addition to invoking one of the install methods, a subsystem instance may be obtained through the service registry. Each installed subsystem has a corresponding service registration. A subsystem service has the following properties.

- `ID` - The ID of the subsystem.
- `Symbolic Name` - The symbolic name of the subsystem.
- `Version` - The version of the subsystem.
- `Type` - The type of the subsystem.
- `State` - The state of the subsystem.

Because a subsystem must be used to install other subsystems, a root subsystem is provided as a starting point. The root subsystem may only be obtained as a service and has the following characteristics.

- The ID is `0`.
- The symbolic name is `org.osgi.service.subsystem.root`.
- The version matches this specification's version.
- It has no parents.
- All existing bundles, including the system and subsystem implementation bundles, are constituents.
- The type is `osgi.subsystem.application` with no imports.
- The provision policy is `acceptDependencies`.

**ThreadSafe**

---

| Nested Class Summary | | Page |
|---|---|---|
| static enum | **Subsystem.State**<br>An enumeration of the possible states of a subsystem. | 34 |

| Method Summary | | Page |
|---|---|---|
| org.osgi.framework.BundleContext | **getBundleContext**()<br>Returns the bundle context of the region within which this subsystem resides. | 24 |
| Collection<Subsystem> | **getChildren**()<br>Returns the child subsystems of this subsystem. | 24 |
| Collection<org.osgi.resource.Resource> | **getConstituents**()<br>Returns the constituent resources of this subsystem. | 26 |
| Map<String,String> | **getDeploymentHeaders**()<br>Returns the headers for this subsystem's deployment manifest. | 26 |
| String | **getLocation**()<br>Returns the location identifier of this subsystem. | 25 |
| Collection<Subsystem> | **getParents**()<br>Returns the parent subsystems of this subsystem. | 25 |
| Subsystem.State | **getState**()<br>Returns the current state of this subsystem. | 26 |
| Map<String,String> | **getSubsystemHeaders**(Locale locale)<br>Returns the headers for this subsystem's subsystem manifest. | 25 |
| long | **getSubsystemId**()<br>Returns the identifier of this subsystem. | 27 |

| | | |
|---|---|---|
| String | **getSymbolicName**()<br>Returns the symbolic name of this subsystem. | 27 |
| String | **getType**()<br>Returns the type of this subsystem. | 27 |
| org.osgi.framework.Version | **getVersion**()<br>Returns the version of this subsystem. | 27 |
| Subsystem | **install**(String location)<br>Installs a subsystem from the specified location identifier. | 28 |
| Subsystem | **install**(String location, InputStream content)<br>Installs a subsystem from the specified content. | 28 |
| Subsystem | **install**(String location, InputStream content, InputStream deploymentManifest)<br>Installs a subsystem from the specified content according to the specified deployment manifest. | 30 |
| void | **start**()<br>Starts this subsystem. | 30 |
| void | **stop**()<br>Stops this subsystem. | 31 |
| void | **uninstall**()<br>Uninstalls this subsystem. | 32 |

# Method Detail

## getBundleContext

```
org.osgi.framework.BundleContext getBundleContext()
```

Returns the bundle context of the region within which this subsystem resides.

The bundle context offers the same perspective of any resource contained by a subsystem within the region. It may be used, for example, to monitor events internal to the region as well as external events visible to the region. All subsystems within the same region have the same bundle context. If this subsystem is in a state where the bundle context would be invalid, `null` is returned.

**Returns:**

The bundle context of the region within which this subsystem resides or `null` if this subsystem's state is in INSTALL_FAILED, UNINSTALLED.

**Throws:**

`SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission`[this,CONTEXT], and the runtime supports permissions.

---

## getChildren

```
Collection<Subsystem> getChildren()
```

Returns the child subsystems of this subsystem.

**Returns:**

The child subsystems of this subsystem. The returned collection is an unmodifiable snapshot of all subsystems that are installed in this subsystem. The collection will be empty if no subsystems are installed in this subsystem.

**Throws:**

`IllegalStateException` - If this subsystem's state is in INSTALL_FAILED, UNINSTALLED.

## getSubsystemHeaders

`Map<String,String> getSubsystemHeaders(Locale locale)`

Returns the headers for this subsystem's subsystem manifest.

Each key in the map is a header name and the value of the key is the corresponding header value. Because header names are case-insensitive, the methods of the map must treat the keys in a case-insensitive manner. If the header name is not found, `null` is returned. Both original and derived headers will be included in the map.

This method must continue to return the headers while this subsystem is in the `INSTALL_FAILED` or `UNINSTALLED` states.

**Parameters:**

`locale` - The locale for which translations are desired. The header values are translated according to the specified locale. If the specified locale is `null` or not supported, the raw values are returned. If the translation for a particular header is not found, the raw value is returned.

**Returns:**

The headers for this subsystem's subsystem manifest. The returned map is unmodifiable.

**Throws:**

`SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission`[this,METADATA], and the runtime supports permissions.

---

## getLocation

`String getLocation()`

Returns the location identifier of this subsystem.

The location identifier is the `location` that was passed to the `install` method of the `parent` subsystem. It is unique within the framework.

This method must continue to return this subsystem's headers while this subsystem is in the `INSTALL_FAILED` or `UNINSTALLED` states.

**Returns:**

The location identifier of this subsystem.

**Throws:**

`SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission`[this,METADATA], and the runtime supports permissions.

---

## getParents

`Collection<Subsystem> getParents()`

Returns the parent subsystems of this subsystem.

**Returns:**

The parent subsystems of this subsystem. The returned collection is an unmodifiable snapshot of all subsystems in which this subsystem is installed. The collection will be empty for the root subsystem; otherwise, it must contain at least one parent. Scoped subsystems always have only one parent. Unscoped subsystems may have multiple parents.

**Throws:**

`IllegalStateException` - If this subsystem's state is in `INSTALL_FAILED`, `UNINSTALLED`.

---

## getConstituents

`Collection<org.osgi.resource.Resource>` **`getConstituents`**`()`

Returns the constituent resources of this subsystem.

**Returns:**

The constituent resources of this subsystem. The returned collection is an unmodifiable snapshot of the constituent resources of this subsystem. If this subsystem has no constituents, the collection will be empty.

**Throws:**

`IllegalStateException` - If this subsystem's state is in `INSTALL_FAILED`, `UNINSTALLED`.

---

## getDeploymentHeaders

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

Returns the headers for this subsystem's deployment manifest.

Each key in the map is a header name and the value of the key is the corresponding header value. Because header names are case-insensitive, the methods of the map must treat the keys in a case-insensitive manner. If the header name is not found, `null` is returned. Both original and derived headers will be included in the map.

This method must continue to return the headers while this subsystem is in the `INSTALL_FAILED` or `UNINSTALLED` states.

**Returns:**

The headers for this subsystem's deployment manifest. The returned map is unmodifiable.

**Throws:**

`SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission`[this,METADATA], and the runtime supports permissions.

**Since:**

1.1

---

## getState

`Subsystem.State` **`getState`**`()`

Returns the current state of this subsystem.

This method must continue to return this subsystem's state while this subsystem is in the `INSTALL_FAILED` or `UNINSTALLED` states.

**Returns:**

The current state of this subsystem.

---

## getSubsystemId

`long` **`getSubsystemId`**`()`

Returns the identifier of this subsystem.

The identifier is a monotonically increasing, non-negative integer automatically generated at installation time and guaranteed to be unique within the framework. The identifier of the root subsystem is zero.

This method must continue to return this subsystem's identifier while this subsystem is in the `INSTALL_FAILED` or `UNINSTALLED` states.

**Returns:**
The identifier of this subsystem.

---

## getSymbolicName

`String` **`getSymbolicName`**`()`

Returns the symbolic name of this subsystem.

The subsystem symbolic name conforms to the same grammar rules as the bundle symbolic name and is derived from one of the following, in order.

- The value of the `Subsystem-SymbolicName` header, if specified.
- The subsystem URI if passed as the `location` along with the `content` to the `install` method.
- Optionally generated in an implementation specific way.

The combination of subsystem symbolic name and `version` is unique within a region. The symbolic name of the root subsystem is `org.osgi.service.subsystem.root`.

This method must continue to return this subsystem's symbolic name while this subsystem is in the `INSTALL_FAILED` or `UNINSTALLED` states.

**Returns:**
The symbolic name of this subsystem.

---

## getType

`String` **`getType`**`()`

Returns the `type` of this subsystem.

This method must continue to return this subsystem's type while this subsystem is in the `INSTALL_FAILED` or `UNINSTALLED` states.

**Returns:**
The type of this subsystem.

---

## getVersion

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

Returns the `version` of this subsystem.

The subsystem version conforms to the same grammar rules as the bundle version and is derived from one of the following, in order.

- The value of the Subsystem-Version header, if specified.
- The subsystem URI if passed as the location along with the content to the install method.
- Defaults to 0.0.0.

The combination of subsystem symbolic name and version is unique within a region. The version of the root subsystem matches this specification's version.

This method must continue to return this subsystem's version while this subsystem is in the INSTALL_FAILED or UNINSTALLED states.

**Returns:**
The version of this subsystem.

---

## install

```
Subsystem install(String location)
```

Installs a subsystem from the specified location identifier.

This method performs the same function as calling install(String, InputStream) with the specified location identifier and null as the content.

**Parameters:**
location - The location identifier of the subsystem to install.
**Returns:**
The installed subsystem.
**Throws:**
IllegalStateException - If this subsystem's state is in INSTALLING, INSTALL_FAILED, UNINSTALLING, UNINSTALLED.
SubsystemException - If the installation failed.
SecurityException - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission[installed subsystem,LIFECYCLE], and the runtime supports permissions.
**See Also:**
install(String, InputStream)

---

## install

```
Subsystem install(String location,
                  InputStream content)
```

Installs a subsystem from the specified content.

The specified location will be used as an identifier of the subsystem. Every installed subsystem is uniquely identified by its location, which is typically in the form of a URI. If the specified location conforms to the subsystem-uri grammar, the required symbolic name and optional version information will be used as default values.

If the specified content is null, a new input stream must be created from which to read the subsystem by interpreting, in an implementation dependent manner, the specified location.

A subsystem installation must be persistent. That is, an installed subsystem must remain installed across Framework and VM restarts.

All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

The following steps are required to install a subsystem.

1. If an installed subsystem with the specified location identifier already exists, return the installed subsystem.
2. Read the specified content in order to determine the symbolic name, version, and type of the installing subsystem. If an error occurs while reading the content, an installation failure results.
3. If an installed subsystem with the same symbolic name and version already exists within this subsystem's region, complete the installation with one of the following.
    - If the installing and installed subsystems' types are not equal, an installation failure results.
    - If the installing and installed subsystems' types are equal, and the installed subsystem is already a child of this subsystem, return the installed subsystem.
    - If the installing and installed subsystems' types are equal, and the installed subsystem is not already a child of this subsystem, add the installed subsystem as a child of this subsystem, increment the installed subsystem's reference count by one, and return the installed subsystem.
4. Create a new subsystem based on the specified location and content.
5. If the subsystem is scoped, install and start a new region context bundle.
6. Change the state to `INSTALLING` and register a new subsystem service.
7. Discover the subsystem's content resources. If any mandatory resource is missing, an installation failure results.
8. Discover the dependencies required by the content resources. If any mandatory dependency is missing, an installation failure results.
9. Using a framework `ResolverHook`, disable runtime resolution for the resources.
10. For each resource, increment the reference count by one. If the reference count is one, install the resource. If an error occurs while installing a resource, an install failure results with that error as the cause.
11. If the subsystem is scoped, enable the import sharing policy.
12. Enable runtime resolution for the resources.
13. Change the state of the subsystem to `INSTALLED`.
14. Return the new subsystem.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption. An interrupted thread should result in a `org.osgi.service.subsystem.SubsystemException` with an InterruptedException as the cause and be treated as an installation failure.

All installation failure flows include the following, in order.

1. Change the state to `INSTALL_FAILED`.
2. Change the state to `UNINSTALLING`.
3. All content and dependencies which may have been installed by the installing process must be uninstalled.
4. Change the state to `UNINSTALLED`.
5. Unregister the subsystem service.
6. If the subsystem is a scoped subsystem then, uninstall the region context bundle.
7. Throw a `org.osgi.service.subsystem.SubsystemException` with the cause of the installation failure.

**Parameters:**

`location` - The location identifier of the subsystem to be installed.

`content` - The input stream from which this subsystem will be read or `null` to indicate the input stream must be created from the specified location identifier. The input stream will always be closed when this method completes, even if an exception is thrown.

**Returns:**

The installed subsystem.

**Throws:**

`IllegalStateException` - If this subsystem's state is in `INSTALLING`, `INSTALL_FAILED`, `UNINSTALLING`, `UNINSTALLED`.

SubsystemException - If the installation failed.
SecurityException - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission[installed subsystem,LIFECYCLE], and the runtime supports permissions.

---

## install

```
Subsystem install(String location,
                  InputStream content,
                  InputStream deploymentManifest)
```

Installs a subsystem from the specified content according to the specified deployment manifest.

This method installs a subsystem using the provided deployment manifest instead of the one in the archive, if any, or a computed one. If the deployment manifest is null, the behavior is exactly the same as in the install(String, InputStream) method. Implementations must support deployment manifest input streams in the format described by section 134.2 of the Subsystem Service Specification. If the deployment manifest does not conform to the subsystem manifest (see 134.15.2), the installation fails.

**Parameters:**
location - The location identifier of the subsystem to be installed.
content - The input stream from which this subsystem will be read or null to indicate the input stream must be created from the specified location identifier. The input stream will always be closed when this method completes, even if an exception is thrown.
deploymentManifest - The deployment manifest to use in lieu of the one in the archive, if any, or a computed one.

**Returns:**
The installed subsystem.

**Throws:**
IllegalStateException - If this subsystem's state is in INSTALLING, INSTALL_FAILED, UNINSTALLING, UNINSTALLED.
SubsystemException - If the installation failed.
SecurityException - If the caller does not have the appropriate org.osgi.service.subsystem.SubsystemPermission[installed subsystem,LIFECYCLE], and the runtime supports permissions.

**Since:**
1.1

---

## start

```
void start()
```

Starts this subsystem.

The following table shows which actions are associated with each state. An action of Wait means this method will block until a state transition occurs, upon which the new state will be evaluated in order to determine how to proceed. If a state transition does not occur in a reasonable time while waiting then no action is taken and a SubsystemException is thrown to indicate the subsystem was unable to be started. An action of Return means this method returns immediately without taking any other action.

| State | Action |
|---|---|
| INSTALLING | Wait |
| INSTALLED | Resolve, Start |
| INSTALL_FAILED | IllegalStateException |
| RESOLVING | Wait |
| RESOLVED | Start |
| STARTING | Wait |

| | |
|---|---|
| ACTIVE | Return |
| STOPPING | Wait |
| UNINSTALLING | IllegalStateException |
| UNINSTALLED | IllegalStateException |

All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

A subsystem must be persistently started. That is, a started subsystem must be restarted across Framework and VM restarts, even if a start failure occurs.

The following steps are required to start this subsystem.

1. Set the subsystem *autostart setting* to *started*.
2. If this subsystem is in the `RESOLVED` state, proceed to step 7.
3. Change the state to `RESOLVING`.
4. Resolve the content resources. A resolution failure results in a start failure with a state of `INSTALLED`.
5. Change the state to `RESOLVED`.
6. If this subsystem is scoped, enable the export sharing policy.
7. Change the state to `STARTING`.
8. For each eligible resource, increment the active use count by one. If the active use count is one, start the resource. All dependencies must be started before any content resource, and content resources must be started according to the specified `start order`. If an error occurs while starting a resource, a start failure results with that error as the cause.
9. Change the state to `ACTIVE`.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption. An interrupted thread should be treated as a start failure with an `InterruptedException` as the cause.

All start failure flows include the following, in order.

1. If the subsystem state is `STARTING` then change the state to `STOPPING` and stop all resources that were started as part of this operation.
2. Change the state to either `INSTALLED` or `RESOLVED`.
3. Throw a SubsystemException with the specified cause.

**Throws:**

`SubsystemException` - If this subsystem fails to start.
`IllegalStateException` - If this subsystem's state is in `INSTALL_FAILED`, `UNINSTALLING`, or `UNINSTALLED`, or if the state of at least one of this subsystem's parents is not in `STARTING`, `ACTIVE`.
`SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission`[this,EXECUTE], and the runtime supports permissions.

---

## stop

`void` **`stop`**`()`

Stops this subsystem.

The following table shows which actions are associated with each state. An action of `Wait` means this method will block until a state transition occurs, upon which the new state will be evaluated in order to determine how to proceed. If a state transition does not occur in a reasonable time while waiting then no action is taken and a SubsystemException is thrown to indicate the subsystem was unable to be stopped. An action of `Return` means this method returns immediately without taking any other action.

| State | Action |
|---|---|
| INSTALLING | Wait |
| INSTALLED | Return |
| INSTALL_FAILED | IllegalStateException |
| RESOLVING | Wait |
| RESOLVED | Return |
| STARTING | Wait |
| ACTIVE | Stop |
| STOPPING | Wait |
| UNINSTALLING | IllegalStateException |
| UNINSTALLED | IllegalStateException |

A subsystem must be persistently stopped. That is, a stopped subsystem must remain stopped across Framework and VM restarts.

All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

The following steps are required to stop this subsystem.

1. Set the subsystem *autostart setting* to *stopped*.
2. Change the state to `STOPPING`.
3. For each eligible resource, decrement the active use count by one. If the active use count is zero, stop the resource. All content resources must be stopped before any dependencies, and content resources must be stopped in reverse `start order`.
4. Change the state to `RESOLVED`.

With regard to error handling, once this subsystem has transitioned to the `STOPPING` state, every part of each step above must be attempted. Errors subsequent to the first should be logged. Once the stop process has completed, a SubsystemException must be thrown with the initial error as the specified cause.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption, in which case a SubsystemException with an InterruptedException as the cause should be thrown. If an interruption occurs while waiting, this method should terminate immediately. Once the transition to the `STOPPING` state has occurred, however, this method must not terminate due to an interruption until the stop process has completed.

**Throws:**

`SubsystemException` - If this subsystem fails to stop cleanly.
`IllegalStateException` - If this subsystem's state is in `INSTALL_FAILED`, `UNINSTALLING`, or `UNINSTALLED`.
`SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission`[this,EXECUTE], and the runtime supports permissions.

---

## uninstall

`void` **`uninstall`**`()`

Uninstalls this subsystem.

The following table shows which actions are associated with each state. An action of `Wait` means this method will block until a state transition occurs, upon which the new state will be evaluated in order to determine how to proceed. If a state transition does not occur in a reasonable time while waiting then no action is taken and a SubsystemException is thrown to indicate the subsystem was unable to be uninstalled. An action of `Return` means this method returns immediately without taking any other action.

| State | Action |
|---|---|
| INSTALLING | Wait |
| INSTALLED | Uninstall |
| INSTALL_FAILED | Wait |
| RESOLVING | Wait |
| RESOLVED | Uninstall |
| STARTING | Wait |
| ACTIVE | Stop, Uninstall |
| STOPPING | Wait |
| UNINSTALLING | Wait |
| UNINSTALLED | Return |

All references to changing the state of this subsystem include both changing the state of the subsystem object as well as the state property of the subsystem service registration.

The following steps are required to uninstall this subsystem after being stopped if necessary.

1. Change the state to `INSTALLED`.
2. Change the state to `UNINSTALLING`.
3. For each referenced resource, decrement the reference count by one. If the reference count is zero, uninstall the resource. All content resources must be uninstalled before any dependencies.
4. Change the state to `UNINSTALLED`.
5. Unregister the subsystem service.
6. If the subsystem is scoped, uninstall the region context bundle.

With regard to error handling, once this subsystem has transitioned to the `UNINSTALLING` state, every part of each step above must be attempted. Errors subsequent to the first should be logged. Once the uninstall process has completed, a `SubsystemException` must be thrown with the specified cause.

Implementations should be sensitive to the potential for long running operations and periodically check the current thread for interruption, in which case a `SubsystemException` with an `InterruptedException` as the cause should be thrown. If an interruption occurs while waiting, this method should terminate immediately. Once the transition to the `UNINSTALLING` state has occurred, however, this method must not terminate due to an interruption until the uninstall process has completed.

**Throws:**

`SubsystemException` - If this subsystem fails to uninstall cleanly.
`SecurityException` - If the caller does not have the appropriate `org.osgi.service.subsystem.SubsystemPermission`[this,LIFECYCLE], and the runtime supports permissions.

# Enum Subsystem.State

org.osgi.service.subsystem

```
java.lang.Object
  └ java.lang.Enum<Subsystem.State>
      └ org.osgi.service.subsystem.Subsystem.State
```

**All Implemented Interfaces:**

Comparable<Subsystem.State>, Serializable

**Enclosing class:**

Subsystem

---

```
public static enum Subsystem.State
extends Enum<Subsystem.State>
```

An enumeration of the possible states of a subsystem.

These states are a reflection of what constituent resources are permitted to do and not an aggregation of constituent resource states.

---

| Enum Constant Summary | Page |
|---|---|
| **ACTIVE**<br>The subsystem is now running. | 36 |
| **INSTALL_FAILED**<br>The subsystem failed to install. | 35 |
| **INSTALLED**<br>The subsystem is installed but not yet resolved. | 35 |
| **INSTALLING**<br>The subsystem is in the process of installing. | 35 |
| **RESOLVED**<br>The subsystem is resolved and able to be started. | 35 |
| **RESOLVING**<br>The subsystem is in the process of resolving. | 35 |
| **STARTING**<br>The subsystem is in the process of starting. | 36 |
| **STOPPING**<br>The subsystem is in the process of stopping. | 36 |
| **UNINSTALLED**<br>The subsystem is uninstalled and may not be used. | 36 |
| **UNINSTALLING**<br>The subsystem is in the process of uninstalling. | 36 |

| Method Summary | | Page |
|---|---|---|
| static Subsystem.State | **valueOf**(String name) | 37 |
| static Subsystem.State[] | **values**() | 36 |

# Enum Constant Detail

## INSTALLING

```
public static final Subsystem.State INSTALLING
```

The subsystem is in the process of installing.

A subsystem is in the INSTALLING state when the install method of its parent is active, and attempts are being made to install its content resources. If the install method completes without exception, then the subsystem has successfully installed and must move to the INSTALLED state. Otherwise, the subsystem has failed to install and must move to the INSTALL_FAILED state.

---

## INSTALLED

```
public static final Subsystem.State INSTALLED
```

The subsystem is installed but not yet resolved.

A subsystem is in the INSTALLED state when it has been installed in a parent subsystem but is not or cannot be resolved. This state is visible if the dependencies of the subsystem's content resources cannot be resolved.

---

## INSTALL_FAILED

```
public static final Subsystem.State INSTALL_FAILED
```

The subsystem failed to install.

A subsystem is in the INSTALL_FAILED state when an unrecoverable error occurred during installation. The subsystem is in an unusable state but references to the subsystem object may still be available and used for introspection.

---

## RESOLVING

```
public static final Subsystem.State RESOLVING
```

The subsystem is in the process of resolving.

A subsystem is in the RESOLVING state when the start method is active, and attempts are being made to resolve its content resources. If the resolve method completes without exception, then the subsystem has successfully resolved and must move to the RESOLVED state. Otherwise, the subsystem has failed to resolve and must move to the INSTALLED state.

---

## RESOLVED

```
public static final Subsystem.State RESOLVED
```

The subsystem is resolved and able to be started.

A subsystem is in the RESOLVED state when all of its content resources are resolved. Note that the subsystem is not active yet.

---

### STARTING

```
public static final Subsystem.State STARTING
```

The subsystem is in the process of starting.

A subsystem is in the STARTING state when its start method is active, and attempts are being made to start its content and dependencies. If the start method completes without exception, then the subsystem has successfully started and must move to the ACTIVE state. Otherwise, the subsystem has failed to start and must move to the RESOLVED state.

---

### ACTIVE

```
public static final Subsystem.State ACTIVE
```

The subsystem is now running.

A subsystem is in the ACTIVE state when its content and dependencies have been successfully started.

---

### STOPPING

```
public static final Subsystem.State STOPPING
```

The subsystem is in the process of stopping.

A subsystem is in the STOPPING state when its stop method is active, and attempts are being made to stop its content and dependencies. When the stop method completes, the subsystem is stopped and must move to the RESOLVED state.

---

### UNINSTALLING

```
public static final Subsystem.State UNINSTALLING
```

The subsystem is in the process of uninstalling.

A subsystem is in the UNINSTALLING state when its uninstall method is active, and attempts are being made to uninstall its constituent and dependencies. When the uninstall method completes, the subsystem is uninstalled and must move to the UNINSTALLED state.

---

### UNINSTALLED

```
public static final Subsystem.State UNINSTALLED
```

The subsystem is uninstalled and may not be used.

The UNINSTALLED state is only visible after a subsystem's constituent and dependencies are uninstalled. The subsystem is in an unusable state but references to the subsystem object may still be available and used for introspection.

## Method Detail

### values

```
public static Subsystem.State[] values()
```

## valueOf

```
public static Subsystem.State valueOf(String name)
```

# Class SubsystemConstants

org.osgi.service.subsystem

```
java.lang.Object
  └ org.osgi.service.subsystem.SubsystemConstants
```

---

```
public class SubsystemConstants
extends Object
```

Defines the constants used by Subsystem service property, manifest header, attribute and directive keys.

The values associated with these keys are of type `String`, unless otherwise indicated.

**Immutable**

---

| Field Summary | | Page |
|---|---|---|
| static String | **DEPLOYED_CONTENT**<br>Manifest header identifying the resources to be deployed. | 39 |
| static String | **DEPLOYED_VERSION_ATTRIBUTE**<br>Manifest header attribute identifying the deployed version. | 40 |
| static String | **DEPLOYMENT_MANIFESTVERSION**<br>Manifest header identifying the deployment manifest version. | 40 |
| static String | **PREFERRED_PROVIDER**<br>Manifest header used to express a preference for particular resources to satisfy implicit package dependencies. | 40 |
| static String | **PROVISION_POLICY_ACCEPT_DEPENDENCIES**<br>A value for the provision-policy directive indicating the subsystem accepts dependency resources. | 40 |
| static String | **PROVISION_POLICY_DIRECTIVE**<br>Manifest header directive identifying the provision policy. | 40 |
| static String | **PROVISION_POLICY_REJECT_DEPENDENCIES**<br>A value for the provision-policy directive indicating the subsystem does not accept dependency resources. | 40 |
| static String | **PROVISION_RESOURCE**<br>Manifest header identifying the resources to be deployed to satisfy the dependencies of a subsystem. | 40 |
| static String | **ROOT_SUBSYSTEM_SYMBOLICNAME**<br>The symbolic name of the root subsystem. | 43 |
| static String | **START_ORDER_DIRECTIVE**<br>Manifest header directive identifying the start order of subsystem contents. | 41 |
| static String | **SUBSYSTEM_CATEGORY**<br>Manifest header identifying the categories of a subsystem as a comma-delimited list. | 41 |
| static String | **SUBSYSTEM_CONTACTADDRESS**<br>Manifest header identifying the contact address where problems with a subsystem may be reported; for example, an email address. | 41 |
| static String | **SUBSYSTEM_CONTENT**<br>Manifest header identifying the list of subsystem contents identified by a symbolic name and version. | 41 |
| static String | **SUBSYSTEM_COPYRIGHT**<br>Manifest header identifying a subsystem's copyright information. | 41 |

| | | |
|---|---|---|
| static String | **SUBSYSTEM_DESCRIPTION**<br>Manifest header identifying the human readable description. | 41 |
| static String | **SUBSYSTEM_DOCURL**<br>Manifest header identifying a subsystem's documentation URL, from which further information about the subsystem may be obtained. | 41 |
| static String | **SUBSYSTEM_EXPORTSERVICE**<br>Manifest header identifying services offered for export. | 42 |
| static String | **SUBSYSTEM_ICON**<br>Manifest header identifying the icon URL for the subsystem. | 42 |
| static String | **SUBSYSTEM_ID_PROPERTY**<br>The name of the service property for the subsystem ID. | 42 |
| static String | **SUBSYSTEM_IMPORTSERVICE**<br>Manifest header identifying services required for import. | 42 |
| static String | **SUBSYSTEM_LICENSE**<br>Manifest header identifying a subsystem's license. | 42 |
| static String | **SUBSYSTEM_MANIFESTVERSION**<br>Manifest header identifying the subsystem manifest version. | 42 |
| static String | **SUBSYSTEM_NAME**<br>Manifest header identifying the human readable subsystem name. | 43 |
| static String | **SUBSYSTEM_STATE_PROPERTY**<br>The name of the service property for the subsystem state. | 43 |
| static String | **SUBSYSTEM_SYMBOLICNAME**<br>Manifest header value identifying the symbolic name for the subsystem. | 43 |
| static String | **SUBSYSTEM_SYMBOLICNAME_PROPERTY**<br>The name of the service property for the subsystem symbolic name. | 43 |
| static String | **SUBSYSTEM_TYPE**<br>Manifest header identifying the subsystem type. | 43 |
| static String | **SUBSYSTEM_TYPE_APPLICATION**<br>The resource type value identifying an application subsystem. | 44 |
| static String | **SUBSYSTEM_TYPE_COMPOSITE**<br>The resource type value identifying an composite subsystem. | 44 |
| static String | **SUBSYSTEM_TYPE_FEATURE**<br>The resource type value identifying an feature subsystem. | 44 |
| static String | **SUBSYSTEM_TYPE_PROPERTY**<br>The name of the service property for the subsystem type. | 43 |
| static String | **SUBSYSTEM_VENDOR**<br>Manifest header identifying a subsystem's vendor. | 44 |
| static String | **SUBSYSTEM_VERSION**<br>Manifest header value identifying the version of the subsystem. | 44 |
| static String | **SUBSYSTEM_VERSION_PROPERTY**<br>The name of the service property for the subsystem version. | 44 |

# Field Detail

## DEPLOYED_CONTENT

```
public static final String DEPLOYED_CONTENT = "Deployed-Content"
```

Manifest header identifying the resources to be deployed.

## DEPLOYED_VERSION_ATTRIBUTE

```
public static final String DEPLOYED_VERSION_ATTRIBUTE = "deployed-version"
```

Manifest header attribute identifying the deployed version.

---

## DEPLOYMENT_MANIFESTVERSION

```
public static final String DEPLOYMENT_MANIFESTVERSION = "Deployment-ManifestVersion"
```

Manifest header identifying the deployment manifest version. If not present, the default value is `1`.

---

## PREFERRED_PROVIDER

```
public static final String PREFERRED_PROVIDER = "Preferred-Provider"
```

Manifest header used to express a preference for particular resources to satisfy implicit package dependencies.

---

## PROVISION_POLICY_DIRECTIVE

```
public static final String PROVISION_POLICY_DIRECTIVE = "provision-policy"
```

Manifest header directive identifying the provision policy. The default value is `rejectDependencies`

**See Also:**
PROVISION_POLICY_ACCEPT_DEPENDENCIES, PROVISION_POLICY_REJECT_DEPENDENCIES

---

## PROVISION_POLICY_ACCEPT_DEPENDENCIES

```
public static final String PROVISION_POLICY_ACCEPT_DEPENDENCIES = "acceptDependencies"
```

A value for the `provision-policy` directive indicating the subsystem accepts dependency resources. The root subsystem has this provision policy.

---

## PROVISION_POLICY_REJECT_DEPENDENCIES

```
public static final String PROVISION_POLICY_REJECT_DEPENDENCIES = "rejectDependencies"
```

A value for the `provision-policy` directive indicating the subsystem does not accept dependency resources. This is the default value.

---

## PROVISION_RESOURCE

```
public static final String PROVISION_RESOURCE = "Provision-Resource"
```

Manifest header identifying the resources to be deployed to satisfy the dependencies of a subsystem.

---

## START_ORDER_DIRECTIVE

```
public static final String START_ORDER_DIRECTIVE = "start-order"
```

Manifest header directive identifying the start order of subsystem contents. There is no default value. Specified values are of type `String` and must represent an integer.

---

## SUBSYSTEM_CATEGORY

```
public static final String SUBSYSTEM_CATEGORY = "Subsystem-Category"
```

Manifest header identifying the categories of a subsystem as a comma-delimited list.

**Since:**
1.1

---

## SUBSYSTEM_CONTACTADDRESS

```
public static final String SUBSYSTEM_CONTACTADDRESS = "Subsystem-ContactAddress"
```

Manifest header identifying the contact address where problems with a subsystem may be reported; for example, an email address.

**Since:**
1.1

---

## SUBSYSTEM_CONTENT

```
public static final String SUBSYSTEM_CONTENT = "Subsystem-Content"
```

Manifest header identifying the list of subsystem contents identified by a symbolic name and version.

---

## SUBSYSTEM_COPYRIGHT

```
public static final String SUBSYSTEM_COPYRIGHT = "Subsystem-Copyright"
```

Manifest header identifying a subsystem's copyright information.

**Since:**
1.1

---

## SUBSYSTEM_DESCRIPTION

```
public static final String SUBSYSTEM_DESCRIPTION = "Subsystem-Description"
```

Manifest header identifying the human readable description.

---

## SUBSYSTEM_DOCURL

```
public static final String SUBSYSTEM_DOCURL = "Subsystem-DocURL"
```

Manifest header identifying a subsystem's documentation URL, from which further information about the subsystem may be obtained.

**Since:**
1.1

---

## SUBSYSTEM_EXPORTSERVICE

```
public static final String SUBSYSTEM_EXPORTSERVICE = "Subsystem-ExportService"
```

Manifest header identifying services offered for export.

---

## SUBSYSTEM_ICON

```
public static final String SUBSYSTEM_ICON = "Subsystem-Icon"
```

Manifest header identifying the icon URL for the subsystem.

**Since:**
1.1

---

## SUBSYSTEM_ID_PROPERTY

```
public static final String SUBSYSTEM_ID_PROPERTY = "subsystem.id"
```

The name of the service property for the subsystem ID. The value of this property must be of type `Long`.

---

## SUBSYSTEM_IMPORTSERVICE

```
public static final String SUBSYSTEM_IMPORTSERVICE = "Subsystem-ImportService"
```

Manifest header identifying services required for import.

---

## SUBSYSTEM_LICENSE

```
public static final String SUBSYSTEM_LICENSE = "Subsystem-License"
```

Manifest header identifying a subsystem's license.

**Since:**
1.1

---

## SUBSYSTEM_MANIFESTVERSION

```
public static final String SUBSYSTEM_MANIFESTVERSION = "Subsystem-ManifestVersion"
```

Manifest header identifying the subsystem manifest version. If not present, the default value is `1`.

---

## SUBSYSTEM_NAME

```
public static final String SUBSYSTEM_NAME = "Subsystem-Name"
```

Manifest header identifying the human readable subsystem name.

---

## SUBSYSTEM_STATE_PROPERTY

```
public static final String SUBSYSTEM_STATE_PROPERTY = "subsystem.state"
```

The name of the service property for the subsystem state. The value of this property must be of type Subsystem.State.

---

## SUBSYSTEM_SYMBOLICNAME

```
public static final String SUBSYSTEM_SYMBOLICNAME = "Subsystem-SymbolicName"
```

Manifest header value identifying the symbolic name for the subsystem. Must be present.

---

## SUBSYSTEM_SYMBOLICNAME_PROPERTY

```
public static final String SUBSYSTEM_SYMBOLICNAME_PROPERTY = "subsystem.symbolicName"
```

The name of the service property for the subsystem symbolic name.

---

## ROOT_SUBSYSTEM_SYMBOLICNAME

```
public static final String ROOT_SUBSYSTEM_SYMBOLICNAME = "org.osgi.service.subsystem.root"
```

The symbolic name of the root subsystem.

---

## SUBSYSTEM_TYPE

```
public static final String SUBSYSTEM_TYPE = "Subsystem-Type"
```

Manifest header identifying the subsystem type.

**See Also:**
SUBSYSTEM_TYPE_APPLICATION, SUBSYSTEM_TYPE_COMPOSITE, SUBSYSTEM_TYPE_FEATURE

---

## SUBSYSTEM_TYPE_PROPERTY

```
public static final String SUBSYSTEM_TYPE_PROPERTY = "subsystem.type"
```

The name of the service property for the subsystem type.

**See Also:**
SUBSYSTEM_TYPE_APPLICATION, SUBSYSTEM_TYPE_COMPOSITE, SUBSYSTEM_TYPE_FEATURE

---

## SUBSYSTEM_TYPE_APPLICATION

```
public static final String SUBSYSTEM_TYPE_APPLICATION = "osgi.subsystem.application"
```

The resource type value identifying an application subsystem.

This value is used for the `osgi.identity` capability attribute `type`, the SUBSYSTEM_TYPE manifest header and the SUBSYSTEM_TYPE_PROPERTY service property.

---

## SUBSYSTEM_TYPE_COMPOSITE

```
public static final String SUBSYSTEM_TYPE_COMPOSITE = "osgi.subsystem.composite"
```

The resource type value identifying an composite subsystem.

This value is used for the `osgi.identity` capability attribute `type`, the SUBSYSTEM_TYPE manifest header and the SUBSYSTEM_TYPE_PROPERTY service property.

---

## SUBSYSTEM_TYPE_FEATURE

```
public static final String SUBSYSTEM_TYPE_FEATURE = "osgi.subsystem.feature"
```

The resource type value identifying an feature subsystem.

This value is used for the `osgi.identity` capability attribute `type`, the SUBSYSTEM_TYPE manifest header and the SUBSYSTEM_TYPE_PROPERTY service property.

---

## SUBSYSTEM_VENDOR

```
public static final String SUBSYSTEM_VENDOR = "Subsystem-Vendor"
```

Manifest header identifying a subsystem's vendor.

**Since:**
1.1

---

## SUBSYSTEM_VERSION

```
public static final String SUBSYSTEM_VERSION = "Subsystem-Version"
```

Manifest header value identifying the version of the subsystem. If not present, the default value is `0.0.0`.

---

## SUBSYSTEM_VERSION_PROPERTY

```
public static final String SUBSYSTEM_VERSION_PROPERTY = "subsystem.version"
```

The name of the service property for the subsystem `version`. The value of this property must be of type `Version`.

# 9 Document Support

## 9.1 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

*Add references simply by adding new items. You can then cross-refer to them by chosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph.* ***STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.***

## 9.2 Author's Address


| | |
|---|---|
| Name | Graham Charters |
| Company | IBM |
| Address | |
| Voice | |
| e-mail | charters@uk.ibm.com |

| | |
|---|---|
| Name | Tom Watson |
| Company | IBM |
| Address | |
| Voice | |
| e-mail | tjwatson@us.ibm.com |

| | |
|---|---|
| Name | John Ross |
| Company | IBM |
| Address | |
| Voice | |
| e-mail | jwross@us.ibm.com |


## 9.3 Acronyms and Abbreviations

ESA – Enterprise Subsystem Archive (or .esa)

## 9.4 End of Document