RFC 201 - Subsystems Update

Draft

19 Pages

Abstract

10 point Arial Centered.

Put information about the purpose of the document and the information that it contains here. This text should not extend beyond this front page. If it does, revise the abstract.

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* *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.* *Graham Charters, IBM (charters@uk.ibm.com)*
0.2	*7th June 2013*	*Added sections 5.5 – 5.7.* *John Ross, IBM (jwross@us.ibm.com)*
0.3	*7th June 2013*	*Added Configuration section* *Tom Watson, IBM (tjwatson@us.ibm.com)*
0.4	*17th June 2013*	*Accepted all changes as of the Jun 13 OSGi F2F.* *Updated sections 3.2 and 5.5 through 5.7 based on F2F feedback.* *John Ross, IBM (jwross@us.ibm.com)*
0.5	*27th June 2013*	*Accepted all changes reviewed on the 26th June 2013 EEG call.* *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.* *Added the text of the comment in 5.5 regarding why DynamicImport-Package was not addressed to the Considered Alternatives chapter. Deleted the comment.* *Added the text of the comment in 5.7 regarding SubsystemDTO to the Considered Alternatives section. Deleted the comment.* *Added section 3.10 based on bug 2537.*

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.

2.1 TCCL Reliance

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.

It is 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.

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 1916 – Define the TCCL for Scoped Subsystems

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.

3.4 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.

3.5 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.

3.6 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.

3.7 Subsystem Configuration

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.

3.8 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?

TODO – this came from meeting minutes – need more details on when matching is needed.

3.9 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.

3.10 Bug 2537 - [subsystems] 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.

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

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

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.

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).

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
```

5.2 Informational Subsystem Headers

5.2.1 Subsystem-Category

The subsystem cateogory holds a comma separated list of category names.

5.2.2 Subsystem-Copyright

5.2.3 Subsystem-DocURL

5.2.4 Subsystem-License

5.2.5 Subsystem-Vendor

5.2.6 Subsystem-ContactAddress

5.2.7 Subsystem-Icon

5.3 Application Subsystem TCCL

5.4 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 provide a means of declaratively identifying the services a bundle provides using the Provide-Capability and Require-Capability headers with the osgi.service namespace (a 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 following:

```
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 of DS XMLs.

5.5 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.

5.6 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.

5.7 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.

5.8 Subsystem Configuration

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.

5.8.1 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.

5.8.2 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.

5.8.2.1 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.

5.8.3 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?)

5.9 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.

5.10 Bug 2537 - [subsystems] Preferred-Provider not always a cure for uses constraint violation.

The following sections represent possible solutions to the issue.

5.10.1 Nothing

Do nothing. This is working as designed.

5.10.2 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.

5.10.3 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.

5.10.4 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.

5.10.5 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.

6 Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.

6.1 Weaving Hooks and DynamicImport-Package

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.

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.

8 Document Support

8.1 References

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

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

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.***

8.2 Author's Address

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

8.3 Acronyms and Abbreviations

ESA – Enterprise Subsystem Archive (or .esa)

8.4 End of Document