RFC 193 - CDI Integration

Draft

64 Pages

Abstract

While OSGi services are very powerful, some still find it challenging to use them effectively. This RFC looks at how CDI can be used to interact with the OSGi service layer. The intent is to bring the popular CDI programming model to OSGi as a way to interact with OSGi services. It will provide the convenience of CDI and allows developers familiar with the CDI technology to reuse their skills in an OSGi context.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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

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

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

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

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

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

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

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

0.2 Trademarks

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

0.3 Feedback

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

0.4 Table of Contents

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 10.1.

`Source code is shown in this typeface.`

0.6 Revision History

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

Revision	Date	Comments
Initial	12-10-16	Initial version of the draf. David Bosschaert <david@redhat.com>
2nd draft	12-10-26	First draft of chapter 5 Paul Bakker <paul.bakker@luminis.eu>
3rd draft	12-10-30	Life-cycle callbacks Paul Bakker <paul.bakker@luminis.eu>
4th draft	13-01-27	Details on service dynamics Paul Bakker <paul.bakker@luminis.eu>
5th draft	13-02-27	Comments after EEG call 2/20/2013
7th draft	09.04.13	New sections Entities, CDI container lifecycle, Requirements and Capabilities. A number of comments on existing sections. Harald Wellmann/ Rebaze <harald.wellmann@rebaze.com>
8th draft	October, 2013	David Bosschaert, some cleanup and additions: • Removed OSGi prefix from classnames • Replaced API described in the text with actual Javadoc generated from classes in OSGi git • Tentatively added sections about support for scopes and support for 'bare' @Inject injection points • Tentatively added section about support for declaring @Component via manifest headers • Added section on the CdiContainer service • Added section on osgi.capability portable Java contracts
9th draft	13-11-07	David Bosschaert, process input from call 6th November 2013.
10th draft	13-11-14	David Bosschaert, process input from the virtual F2F November 13, 2013.
11th draft	14-11-05	Emily Jiang, update to CDI 1.2
12th draft	14-11-20	Emily Jiang more update based on the comments from F2F in Los Angeles
13th draft	15-01-17	Emily Jiang. Add lifecycle and Event
14th draft	15-02-15	Emily Jiang, update based on the call 11th March 2015
15th draft	15-09-09	Emily Jiang, updated based on the call in June F2F
16th draft	15-12-18	Emily Jiang, updated based on F2F in November
17th draft	16-11-17	Raymond Augé, based on implementation experience
18th draft	17-03-24	Raymond Augé, based on feedback from Montpellier F2F
19th draft	17-05-10	Raymond Augé, based on feedback from EEG call

1 Introduction

While OSGi services are very powerful, consuming them has been a challenge for many OSGi users. There have been a number of solutions to this problem both in OSGi specifications as well as in non-standardized technologies. OSGi Declarative Services and Blueprint are popular specifications in this area, however they provide new programming models that users need to learn. As of JavaEE 6, CDI (JSR 299) is included as a standard injection technology for JavaEE components. CDI becomes a core aspect of JavaEE platform. The JavaEE7 specification has further integrated CDI with many JavaEE components such as EJB, JSF, Bean Validation, JAX-RS etc. It is enabled by default. The CDI programming model seems suitable for interaction with the OSGi service layer as well and has the benefit that developers who are familiar with CDI don't need to learn a new technology in order to interact with the OSGi service registry.

This document proposes that OSGi will support CDI with the goal of creating a specification that describes how the CDI programming model can be used to interact with OSGi services.

2 Application Domain

Software developers often need to build loosely coupled applications. The need for this stems from a number of factors:

- Developing reusable services for consumption outside of the team
- Allowing those services to be easily consumed
- Unit testing of applications and services
- Allowing larger teams to work effectively together by isolating areas of development

Software developers also wish to using a standardized programming model. This promotes:

- Transferability of skill sets
- Ease of sourcing new developers and low initial overhead
- Clear understanding of correct behavior when unexpected behavior is encountered
- Consistency of programming model across the technological strata to provide a uniformity of approach to aid understanding

Finally, software developers require an environment in which the focus can be on solving business issues rather than technological issues. This allows a more responsive development process.

2.1 CDI

CDI, Contexts and Dependency Injection is first specified by JSR 299 and then CDI 1.1 by JSR346. CDI 1.2 is the maintenance release of JSR346. It defines a clean, mostly annotations-based injection model which has recently become very popular. CDI is part of JavaEE 6/7 but can also be used standalone in a JavaSE context.

Weld (http://weld.cdi-spec.org/ is the Reference Implementation of JSR 299 and JSR 346.

2.1.1 Example

Although many advanced features are available, the most basic annotation used in CDI is `javax.inject.Inject` which declares the injection points for CDI.

For example the following Servlet class uses CDI injection to obtain an implementation of the `WeatherBean` interface.

```
public class CDIServlet extends HttpServlet {
    @Inject
    WeatherBean weatherBean;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        PrintWriter writer = resp.getWriter();
        writer.print("The Weather in Amsterdam: " +
            weatherBean.getDescription("Amsterdam"));

        writer.flush();
        writer.close();
    }
}
```

While for the most basic use a CDI provider does not need to be annotated, CDI will attempt to find an implementor class and instantiate it using a no argument constructor. Other mechanisms to publish a bean into CDI can be defined by using the `javax.enterprise.inject.Produces` annotation. Additionally, a number of scopes are defined that can be used to the declare the lifecycle of a CDI bean.

For example, the WeatherBean above can be scoped to the application lifecycle by adding the `javax.enterprise.context.ApplicationScoped` annotation, as in this example:

```
@ApplicationScoped
public class WeatherBeanProducer {
    @Produces @ApplicationScoped
    public WeatherBean newWeatherBean() {
        WeatherBean wb = new WeatherBeanImpl();
        wb.initialize();
        return wb;
    }

    public void disposeWeatherBean(@Disposes WeatherBean wb) {
        wb.cleanup();
    }
}
```

For more information see the CDI specification at JSR 299 [3].Error: Reference source not found and JSR 346 [4].

2.2 Weld-OSGi

The Weld-OSGi project (http://mathieuancelin.github.com/weld-osgi/) has created an integration between CDI and OSGi. It allows CDI beans to be exposed as OSGi services and CDI injections to be satisfied by OSGi services. Weld-OSGi takes additional OSGi features into account such as service registration properties and the dynamic aspects of the Service Registry.

Furthermore, Weld-OSGi provides annotation based injection for the Bundle, BundleContext, Bundle Headers and the private bundle storage facility.

Additionally Weld-OSGi provides annotations-based integration with Service and Bundle events.

2.2.1 Weld-OSGi example

Many examples can be found in the weld-osgi documentation [5].

Weld-OSGi typically uses additional annotations to interact with the OSGi service Registry. For example, the `org.osgi.cdi.api.extension.annotation.Publish` annotation publishes the CDI bean in the OSGi Service Registry:

```
@Publish
@ApplicationScoped
public class MyServiceImpl implements MyService {
    @Override
    public void doSomething() { ... }
}
```

To have a CDI injection come from the OSGi Service Registry, use the `OSGiService` annotation:

```
    @Inject @OSGiService MyService service;
```

OSGi Services can also be selected by using LDAP filters:

```
    @Inject @OSGiService @Filter("&(lang=EN)(country=US)") MyService service;
```

For more examples, see the weld-osgi documentation.

2.3 Declarative Services, Blueprint and CDI

In JavaEE, the EJB and CDI containers are able to collaborate such that EJB manages an EJB component's life-cycle, whilst CDI manages its runtime dependencies. For example, when a new EJB is created it can be handed over to the CDI container for it to process the injections (@Inject) before finally being made available for use. This relationship helps ensure a complementary positioning between the different component models and reduces runtime duplication (EJB is not required to handle @Inject processing itself).

OSGi has two existing component models in the form of Declarative Services and Blueprint. Each has its own mechanism for injection of services and Blueprint also supports bean injection within a bundle. Neither has standards support for runtime annotations for injection, although there is some Blueprint prototype work in Apache Aries. In addressing any requirements for runtime annotations support, serious consideration should be given to the use of existing annotations, such as @Inject. It also makes sense to consider creating similar complementary relationship between their containers and the CDI container for runtime injection processing, thus reducing duplication between various component model containers.

2.4 Terminology + Abbreviations

CDI – Context and Dependency Injection for JavaEE. Specified in JSR 299 and JSR 346.

3 Problem Description

CDI provides a standardized, type-safe, loosely coupled programming model for JavaEE 6 and above. Furthermore, it introduces powerful extensibility into the JavaEE programming model, and promotes an ecosystem of “portable extensions”.

CDI is declarative, with metadata provided via annotations. This allows developers to locate all logic and metadata in a single location, allowing easier comprehension of the application.

CDI does not specify any modularity or inter-application communication, relying instead on the JavaEE platform to provide this.

OSGi provides the de-facto standard within Java for modular, service orientated programming.

Use of CDI in the context of OSGi provides a very compelling programming model. However, today there is no standard way to achieve this. A standard for leveraging CDI in OSGi will provide a migration path between JavaEE and OSGi where developers familiar with CDI can reuse their skills in both contexts without being locked in to a particular implementation.

4 Requirements

4.1 Functional Requirements

CDI001 – The specification MUST make it possible to use the CDI annotations and XML descriptor in an OSGi bundle to expose and consume CDI beans.

CDI003 – The specification MUST make it possible to consume OSGi services in CDI @Inject injection points in an OSGi bundle.

CDI004 – The specification MUST make it possible to select OSGi services used in CDI beans based on OSGi filters.

CDI005 – The specification MUST make it possible to consider CDI qualifiers when looking up CDI beans in the OSGi Service Registry.

CDI014 – The specification MUST provide a mechanism to specify additional OSGi service registration properties for CDI beans when published as an OSGi service.

CDI006 – The specification MUST make it possible to write a portable CDI jar that runs both in JavaEE as well as in OSGi.

CDI007 – The specification MUST consider the thread-safety issues that can arise when migrating CDI beans from JavaEE to OSGi.

CDI008 – The specification MUST consider the issues that can arise in relation to the dynamic bundle lifecycle in OSGi.

CDI015 – The specification MUST consider the issues that can arise with OSGi service dynamism when these services are dependencies of the CDI container.

CDI009 – The specification MUST make it possible to take advantage of the dynamic service capabilities of OSGi.

CDI016 – The specification MUST extend the life-cycle of the CDI container to support the dynamic life-cycle provided by OSGi services. For example, it MUST NOT be fatal to deploy a CDI container that does not have all its service dependencies initially satisfied and it MUST be possible to change service dependencies without requiring the CDI bundle to be redeployed or restarted.

CDI018 – The specification MUST provide a mechanism to consume multiple matching OSGi services of a given type in an injection point. For example via the @Inject Instance<T> mechanism. Service optionality should result from an empty number of matching services.

CDI019 – The specification MUST support CDI events as defined by the CDI specification.

CDI021 – The specification MAY provide a deep integration between CDI events and OSGi events or other OSGi mechanism.

CDI020 – The specification MUST support CDI extensions as defined by the CDI specification but the extension should not be required to contain the file of META-INF/services/`javax.enterprise.inject.spi.Extension`. It should discover the extension via the service registry service of `javax.enterprise.inject.spi.Extension`.

CDI022 – the specification MAY provide a deep integration between CDI extensions and OSGi services or other OSGi mechanism.

CDI010 – The specification MAY introduce additional annotations.

CDI011 – The specification MUST define the behavior in case of incorrect CDI metadata.

CDI012 – The specification MUST NOT prevent the use of @Inject (and other common Java annotations) in other component models/technologies present in the OSGi Framework.

CDI013 – The specification MUST define an opt-in mechanism. Bundles not opting in MUST not be considered by the CDI - OSGi integration layer.

CDI023 – All the inter-bundle interaction between CDI beans MUST go through the OSGi Service Registry.

CDI024 – The specification MUST make it possible to access the BundleContext from inside a CDI bean in an OSGi Framework.

CDI025 – The specification MUST provide support for @PostContruct and @PreDestroy activation and deactivation callbacks.

CDI026 – The specification SHOULD consider defining behavior for relevant CDI scopes.

CDI027 – The solution MAY define new scopes for use with CDI inside an OSGi Framework.

CDI028 – The specification MUST define an opt-in mechanism for CDI extensions.

CDI029 – The specification MUST consider the issues that arise from dynamically adding CDI extensions to the system.

CDI030 – The specification MUST support the inclusion of CDI beans and descriptors in a Web Application Bundle in the same way they can be included in a WAR -.

CDI032 – The specification MUST support the OSGi Service Permission security model when publishing OSGi services from CDI beans and injecting services into CDI beans. It needs to take into account that the CDI extender acts on behalf of other bundles and uses the permissions associated with those.

CDI033 – The specification implementation MUST support CDI 1,2 but may support CDI 1,0.

CDI034a – The specification MUST comply with CDI specification bean defining annotation for honoring a CDI bean not requiring the existence of beans.xml.

CDI034b – The specification SHOULD choose a service when multiple services are available for a particular instead of throwing `AmbiguousResolutionException` specified by CDI specification

CDI035 – The specification MUST exclude the classes from considering to be CDI beans if the classes are listed in the beans.xml under excludes elements.

CDI036 – The specification SHOULD define a means of interacting with configuration admin.

CDI037 – The solution must perform bean discovery at tool time. Processing of beans is done at runtime.

4.2 Non-functional Requirements

CDI050 – The specification MUST NOT prevent an implementation from injecting OSGi services into CDI beans which are not deployed as OSGi bundles.

CDI051 – The specification SHOULD adhere to the current CDI programming model as much as possible.

4.3 Requirements from RFP 98 (OSGi/JavaEE umbrella RFP)

JEE001 – A Java EE/OSGi system SHOULD enable the standard Java EE application artifacts (e.g. web application) to remain installed when a supporting Java EE runtime element (e.g. web container) is dynamically replaced.

JEE002 – RFCs that refer to one or more Java EE technologies MUST NOT impede the ability of an OSGI-compliant implementation to also be compliant with the Java EE specification.

JEE003 – RFCs that refer to one or more Java EE technologies MAY define the additional aspects of the technology that are required for the technology to be properly integrated in an OSGi framework but MUST NOT make any syntactic changes to the Java interfaces defined by those Java EE specifications.

JEE004 – RFCs whose primary purpose is integration with Java EE technologies MUST NOT require an OSGi Execution Environment greater than that which satisfies only the signatures of those Java EE technologies.

5 Technical Solution

5.1 Entities

- CDI – Contexts and Dependency Injection 1.0 (JSR-299), Contexts and Dependency Injection 1.1/1.2 (JSR346).
- CDI Provider – An implementation of the CDI 1.2 specification.
- CDI OSGi adapter – Adapts a given CDI Provider to the OSGi environment. This entity is implementation dependent and may or may not be separate from the CDI provider.

- CDI Bundle – An OSGi bundle containing CDI metadata and beans.
- CDI Container – A container for managed beans in a CDI Bundle. Each CDI Bundle has its own CDI container.
- CDI Extender – An application of the extender pattern to discover CDI Bundles and to manage the CDI container life-cycle on behalf of CDI Bundles. This entity is implementation dependent and may or may not be separate from the CDI OSGi adapter.
- CDI Portable Extension – A portable extension as defined in CDI 1.2.
- CDI Portable Extension Bundle – A bundle providing one or more CDI portable extensions. An extension bundle may or may not be a CDI bundle at the same time.

5.2 Extender Capability and Opt-In

Bean deployment archives according to CDI 1.0 are required to opt in to OSGi enrichment by the CDI Extender. However, opting in may have no effect at all if a would-be CDI Bundle is installed and started in a system where no CDI Extender is available.

This kind of dependency can be made explicit using capabilities. T~~For these reasons, t~~his specification defines an `osgi.extender` capability called `osgi.cdi`.

A CDI Bundle MUST require the OSGi CDI e~~E~~xtender capability, e.g.

```
Require-Capability:\
  osgi.extender;\
    filter:="(&(osgi.extender=osgi.cdi)(version>=1.0)(!(version>=2.0)))"
```

A CDI Bundle must declare the location of any CDI bean descriptors to read from the bundle classpath using the attribute `beans` on the requirement, which is a `List<String>` of file paths, e.g.

```
Require-Capability:\
  osgi.extender;\
    filter:=”(&(osgi.extender=osgi.cdi)(version>=1.0)(!(version>=2.0)))”;\
    beans:List<String>=”META-INF/beans.xml,WEB-INF/beans.xml”
```

If there are no beans files to declare the `beans` attribute is not required.

~~It~~The requirement may also declare an attribute `osgi.beans` ~~on the `osgi.cdi` extender requirement~~ who’s type is `List<String>` of file paths and who’s values~~–~~ are XML documents that contains OSGi Bean descriptor information, e.g.

```
Require-Capability:\
  osgi.extender;\
    filter:=”(&(osgi.extender=osgi.cdi)(version>=1.0)(!(version>=2.0)))”;\
    osgi.beans:List<String>=”OSGI-INF/cdi/osgi-beans.xml”
```

The last component of each path in the `osgi.beans` attribute may use wildcards so that `Bundle.findEntries` can be used to locate the XML document within the bundle and its fragments.

For example:

```
Require-Capability:\
  osgi.extender;\
    filter:=”(osgi.extender=osgi.cdi)”;\
    osgi.beans:List<String>=”OSGI-INF/cdi/*.xml”
```

If the attribute is not specified, the default path `OSGI-INF/cdi/*.xml` will be used.

The CDI Extender must process each XML document specified in this attribute. If an XML document specified by

the attribute cannot be located in the bundle and its attached fragments, the CDI Extender must log an error message with the Log Service, if present, and continue.

The elements in the osgi beans documents are defined in `cdi.xsd` in this RFC folder.

The bean classes may be filtered based on the declared CDI beans descriptors.

If `foo.A` is excluded by `beans.xml`, `foo.A` will not be considered. If there are no CDI bean descriptors, all the beans defined in the OSGi Beans Descriptors will be considered, e.g.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:cdi="http://www.osgi.org/xmlns/cdi/v1.0.0">
  <cdi:bean class="foo.FooImpl" />
  <cdi:bean class="foo.FumImpl" />
  <cdi:bean class="baz.Baz" />
</beans>
```

Note that the `beans.xml`'s `bean-discovery-mode` mode is ignored in favor of populating the OSGi Beans Descriptor with the complete list of bean class names to be considered at runtime. This could be implemented in build tooling.

A CDI Extender implementation MUST provide the `osgi.extender` capability named `osgi.cdi` with version `1.0`, e.g.

```
Provide-Capability: osgi.extender; osgi.extender=osgi.cdi; version:Version=1.0
```

Future versions of this specification will require higher version numbers.

5.3 CDI Portable Extensions

A CDI Pportable eExtension is a mechanism by which a third party may integrate with the CDI container by:

- Providing its own beans, interceptors and decorators to the container
- Injecting dependencies into its own objects using the dependency injection service
- Providing a context implementation for a custom scope
- Augmenting or overriding the annotation-based metadata with metadata from some other source

Use of the API associated with an extension may not be sufficient to express the requirement for the extension's implementation and perhapsit is entirely possible an extension does not provide an API at all. Capabilities are useful to express such loose dependencies. Therefore a capability namespace `osgi.cdi.extension` is defined for use by CDI Portable Extensions and CDI Bundles. This covers the following use case:

CDI Bundle A works with a CDI extension with an annotation API defined in bundle B and an extension implementation in bundle C. A has a package dependency on B, but not on C. There is only an implicit runtime dependency on C. C needs to be resolved when the CDI container for A is constructed, so that the extension can modify the set of manged beans, but C may not be available at all.

CDI Portable Extension Bundles MUST therefore provide an `osgi.cdi.extension` capability with an attribute `osgi.cdi.extension` of type `String` holding a distinctive name for the extension. It must also define a `version` attribute of type `Version`, e.g.

```
Provide-Capability:\
  osgi.cdi.extension; osgi.cdi.extension=frobnicator; version:Version=1.0
```

The extension name `osgi.cdi` is reserved by this specification in case a CDI OSGi Aadapter chooses to implement it's functionality using the extension model.

Due to the dynamic nature of the OSGi runtime, it's essential that a CDI Extender be able to react to the coming and going of CDI Portable Extension Bbundles. The CDI Extender must alter the state of CDI Containers who depend on these extensions. The most reliable way to address this is via the service registry.

For this reason CDI Portable Extensions Bundles must be publishedd extensions into the OSGi service registry as factory services under the interface `javax.enterprise.inject.spi.Extension`. As well, the extension service must include the property `osgi.cdi.extension` containing the name of the extension. This name should be the same as that used in the `osgi.cdi.extension` capability.

~~Extension implementations have been typically instantiated using `Class.forName()` per application classloader. As such, it's likely that Extensions would not support the multi-bundle nature of OSGi. In this case the extension should be published as a `ServiceFactory`. The simplest way to achieve this is using the DS `@Component` annotation with `@Component.scope = ServiceScope.BUNDLE`, e.g.~~

```
~~@Component(~~
~~    property = {CdiExtenderConstants.CDI_EXTENSION + "=frobnicator"},~~
~~    scope = ServiceScope.BUNDLE~~
~~)~~
~~public class Frobnicator implements Extension {}~~
```

A CDI Bundle may depend on CDI ~~p~~Portable ~~e~~Extensions. To express this requirement the CDI Bundle MUST declare the requirement using the `osgi.cdi.extension` namespace with a filter matching the CDI extension's Capability attributes, e.g.

```
Require-Capability:\
  osgi.cdi.extension;\
    filter:="(&(osgi.cdi.extension=frobnicator)(version>=1.0)(!(version>=2.0)))"
```

CDI Portable Extension Bundles should also ~~support~~employ a mechanism for resolving their dependency on the CDI Extender. ~~In order to address this~~Therefore ~~and since the CDI Extender is consuming, but not extending, "extensions" as services;~~ the CDI Extender must provide an `osgi.implementation` capability also using the namespace `osgi.cdi`.

```
Provide-Capability:\
  osgi.implementation; osgi.implementation=osgi.cdi; version:Version=1.0
```

CDI Portable Extension Bundles MUST~~must~~ require this capability:

```
Require-Capability:\
  osgi.implementation;\
    filter:="(&(osgi.implementation=osgi.cdi)(version>=1.0)(!(version>=2.0)))"
```

5.4 CDI Container Life-Cycle

The CDI Extender tracks all bundles becoming ACTIVE. When a tracked bundle is identified as a CDI Bundle, the CDI Extender creates a CDI Container for this bundle. ~~-~~When a ~~tracked~~ CDI Bundle is stopped, the CDI Extender ~~stops~~destroyes the CDI Container for the given bundle.

5.4.1 Initialization of a CDI Container

A CDI Eextender manages the life cycle of a CDI Bundle based on:

- The CDI Bundle state
- The CDI Eextender's bundle state

All activities on behalf of the CDI Bbundle must use the BundleContext of the CDI Bbundle. All dynamic class loads must use the CDI Bbundle's `Bundle.loadClass` method.

5.4.1.1 Initialization steps

The following is a description of the initialization steps. The CDI Container will update its state. State changes are broadcast as events [5.13.1]

If any failures occurs during initialization, or the CDI Bbundle or CDI Eextender bundle is stopped, the CDI Ccontainer must be destroyed.

The initialization process of a CDI Container is defined in the following steps:

1. Wait until a CDI Bbundle is ready. A CDI Bbundle is ready when it is in the ACTIVE state, and for CDI Bbundles that have a lazy activation policy, also in the STARTING state.
2. The CDI ediContainer service is published with the service property `osgi.cdi.container.state = CREATING` and the `Creating` CDI event is fired
3. If the CDI bBundle depends on any extensions, the CDI Ccontainer's `osgi.cdi.container.state` property is updated to `WAITING_FOR_EXTENSIONS`, the `WaitingForExtensions` CDI event is fired and the CDI Ccontainer waits for extensions to be resolved (this MUST not be a busy, blocking wait).
4. The CDI Bbundle's beans are processed and during processing `@org.osgi.service.cdi.annotations.Configuration`, `@org.osgi.service.cdi.annotations.Reference` and `@org.osgi.service.cdi.annotations.Service` annotations are collected.
5. If `@org.osgi.service.cdi.annotations.Configuration` annotations were found, or if the OSGi Beans Description contains any **`configuration`** elements the CDI Ccontainer's `osgi.cdi.container.state` property is updated to `WAITING_FOR_CONFIGURATIONS`, the `WaitingForConfigurations` CDI event is fired and the CDI Ccontainer waits for the referenced configurations to be resolved (this MUST not be a busy, blocking wait).
6. If `@org.osgi.service.cdi.annotations.Reference` annotations were found, or if the OSGi Beans Description contains any reference elements the CDI container's `osgi.cdi.container.state` property is updated to `WAITING_FOR_SERVICES`, the `WaitingForServices` CDI event is fired and the CDI container waits for referenced services to be resolved (this MUST not be a busy, blocking wait).
7. The CDI container's `osgi.cdi.container.state` property is updated to `SATISFIED` and the `Satisfied` CDI event is fired.
8. If `@org.osgi.service.cdi.annotations.Service` annotations were found the beans having the annotations are published into the OSGi service registry using whatever service properties accompanied the annotation and under the correct OSGi service scope depending on their CDI scope.
9. The CDI container's `javax.enterprise.inject.spi.BeanManager` service is published into the OSGi service registry. The CDI container's `osgi.cdi.container.state` property is updated to `CREATED` and the `Created` CDI event is fired.
10. The CDI beans are now active and perform their function until the CDI bBundle or the CDI Eextender bundle are stopped.
11. When the CDI Bbundle is stopped OR the CDI Eextender bundle is stopped, the CDI Ccontainer's `osgi.cdi.container.state` property is updated to `DESTROYING` and the `Destroying` CDI event is fired.
12. The CDI Ccontainer's published bean services are unpublished.
13. The CDI Ccontainer's service dependencies are released.
14. The CDI eContainer's extensions are released.
15. The CDI eContainer's `osgi.cdi.container.state` property is updated to `DESTROYED` and the `Destroyed` CDI event is fired.
16. The CDI Ccontainer is unpublished from the service registry.

17. At any time between the `CREATED` and `DESTROYING` state if any dependent service goes away, the ~~e~~CDI Container must return to the `WAITING_FOR_SERVICES` state making sure to unpublish it's services and destroy it's beans.
18. The moment service dependencies are again satisfied the CDI C~~c~~ontainer may proceed through initialization as described above.
19. At any time between the `CREATED` and `DESTROYING` state if any dependent extension goes away, the ~~e~~CDI Container must return to the `WAITING_FOR_EXTENSIONS` state making sure to unpublish it's services and destroy it's beans.
20. The moment extension dependencies are again satisfied the CDI C~~c~~ontainer may proceed through initialization as described above.
21. At any time between the `CREATED` and `DESTROYING` state if any dependent configuration goes away, the CDI Container must return to the `WAITING_FOR_CONFIGURATIONS` state making sure to unpublish it's services and destroy it's beans.
22. The moment configuration dependencies are again satisfied the CDI Container may proceed through initialization as described above.

5.4.1.2 Failure

If at any time there is a failure, the CDI Container must perform the following steps:

1. The CDI C~~c~~ontainer's `osgi.cdi.container.state` property is updated to `FAILURE` and the `Failure` CDI event is fired.
2. Unregister the CDI C~~c~~ontainer service
3. Destroy the CDI C~~c~~ontainer
4. Wait for the CDI B~~b~~undle to be stopped

5.4.1.3 C*diC~~DI c~~ontainer service*

The CDI Container must be registered ~~as~~ a service under the interface `org.osgi.service.cdi.CdiContainer` with the following service property:

`osgi.cdi.container.state` – (`CdiEvent.Type` enum value) The state of the CDI container

5.4.1.4 Destroy the CDI Container

The CDI Container must be destroyed when any of the following condition becomes true:

- The CDI B~~b~~undle is stopped.
- The CDI ~~e~~Extender is stopped
- One of the initialization phases failed.

Destroying the CDI Container means:

- The CDI C~~c~~ontainer's published bean services are unpublished.
- The CDI ~~e~~Container's service dependencies are released.
- The CDI C~~c~~ontainer's extensions are released.
- Destroying all beans
- Unregistering the CDI Container service

A CDI Container must continue to follow the destruction even when exceptions or other problems occur. These errors should be logged.

If the CDI Eextender is stopped, then all its active CDI Ceontainers must be destroyed in an orderly fashion, synchronously with the stopping of the CDI Eextender bundle. The shutdown of many CDI Ceontainers, which the CDI extender should handle, MUST be safe to perform serially and without deadlocks as the cascade effect of having interdependent services should cause CDI Ceontainers not yet processed to safely degrade into one of the `WAITING_FOR_*` states making it safe to destroy when their turn arrives to be processed.

5.5 Publishing CDI Beans as OSGi Services

A bean can be published to the OSGi service registry using the `@org.osgi.service.cdi.annotations.Service` annotation, e.g.

```
@Service
public class ExampleComponent {}
```

Note that classes that are already CDI beans, are not automatically published to the service registry; the `@org.osgi.service.cdi.annotations.Service` annotation or metadata as described in section 5.5.3 is always required for this. The requirement to explicitly declare a bean to be an OSGi component is necessary for the following reasons:

1. In CDI 1.2 every class is a potential CDI bean, no annotations are required for this. The container uses the classes declared in the OSGi Beans Description as beans. A CDI container in OSGi is scoped to a single bundle, and will only know about injection points in that bundle. OSGi services however can be used by other bundles as well. Although it is technically possible to find injection points in all bundles, this would be very hard to reason about for end users.
2. Not each CDI bean should be registered to the service registry. CDI is often used to inject tightly coupled classes into each other. This is fine for internal bundle usage, but should not be reflected in the service registry.
3. Not all CDI beans might be exported. It's useless to publish services of types that other bundles can't have access to.

This behavior is different then the behavior described in the EJB integration specification, where all EJBs are published as OSGi services by default. Although technically EJBs and CDI beans are very similar, their usage in practice is often very different. EJBs tend to be used in a similar granularity as OSGi services, while CDI beans are not.

OSGi services are published specifying the type under which they are available. In order to support this model the `@org.osgi.service.cdi.annotations.Service` annotation provides a `type` property of type `Class<?>[]`, e.g.

```
@Service(type=A.class)
public class MyBean implements A,B {}
```

The `type` property is optional. If not specified the component is registered in the OSGi Service Registry under all the interfaces it directly implements. If the component does not directly implement any interfaces it will be registered under its implementation class.

5.5.1 Service Properties

Service properties can be defined on the `@org.osgi.service.cdi.annotations.Service` annotation using the `property` element, e.g.

```
@Service(
  property = {
```

```
    "key=bar",
    "timestamp:Long=1480706423"
  }
)
public class MyBean {}
```

The `property` element takes an array of Strings. Each property string is specified as `name=value`. The type of the property value can be specified in the name as `name:type=value`. The type must be one of the property types supported by the `value-type` attribute of the `property` element of the OSGi Bean descriptor along with Collections of type `List` and `Set`.

5.5.2 CDI Qualifiers as Service Properties

All qualifiers will automatically be included as service properties. The mapping will follow the default behavior of the converter spec in converting Annotations to `Map<String, Object>`. Collisions will be handled as last wins where the ordering of the annotations will be as discovered through reflection. Finally, properties set on the @Service will have the highest precedence (be processed last).

Note: Marker annotations (having no members) convert to an empty map and so those are effectively ignored.

5.5.3 Publishing plain CDI beans in the Service Registry

To support integration of existing CDI beans with OSGi, an alternative to the `@org.osgi.service.cdi.annotations.Service` annotation is available via the OSGi Beans Descriptor. This allows beans that act as providers in plain CDI environments to be exposed as services without having to change and recompile them. A bean can be declared as a service in the descriptor by specifying the ~~`service`~~ child element `service` of the **`bean`** element, e.g.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:cdi="http://www.osgi.org/xmlns/cdi/v1.0.0">
  <cdi:bean class="foo.FooImpl">
    <service>
      <provide interface="foo.Foo" />
    </service>
  </cdi:bean>
</beans>
```

It's also possible to provide properties this way, e.g.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:cdi="http://www.osgi.org/xmlns/cdi/v1.0.0">
  <cdi:bean class="foo.FooImpl">
    <service>
      <property key="somekey" value="somevalue" type="String" />
      <property key="someotherkey" type="Integer">
         <array>
           <value>32767</value>
           <value>2345</value>
         </array>
      </property>
      <provide interface="foo.Foo" />
    </service>
  </cdi:bean>
</beans>
```

CDI qualifiers on the bean will automatically be added as service properties as described in 5.5.2.

5.6 Scopes for beans annotated as @Service

The following table outlines the CDI scopes and their support by this specification:

`@ApplicationScoped`	service.scope = bundle (ServiceFactory)
`@Singleton`	service.scope = singleton
`all others`	service.scope = prototype (PrototypeServiceFactory)

5.7 Injecting OSGi Services into beans

CDI beans using the standard `@Inject` annotation may declare that the intended injection target is an OSGi service. By default, a CDI bean has one of a variety of different scopes which means it's lifecycle is bound by the Context in which that scope is defined. In order to ensure that the OSGi services are adequately handled to suite the scope of the bean having the service reference the OSGi `ServiceObjects` API should be used.

Another difference is that there may be multiple OSGi services publishing the same interface, a client may or may not choose a specific instance using service properties and ordering. Natural ordering of OSGi ServiceReferences will be used if multiple candidates are found that match the reference.

Beans must always use `@org.osgi.service.cdi.annotations.Reference` annotation to instruct the CDI container ~~(and the developer) that we are dealing with a specific type of bean, where different rules may apply. When the `@org.osgi.service.cdi.annotations.Reference` annotation is used, the CDI container will~~to lookup the OSGi service matching the injection point bean type in the service registry and inject it's instance into the ~~CDI bean~~injection point, e.g..

```
class Fum {
  @Inject
  @Reference
  Foo foo;
}
```

Any OSGi service registered in the service registry can be injected using CDI, even if the service was not registered using CDI ~~but with the low level service API or DS for example~~.

It's also possible using the `@Reference.scope` property to specify the scope of the service to be injected, e.g.

```
class Fum {
  @Inject
  @Reference(scope = ReferenceScope.SINGLETON)
  Foo foo;
}
```

When a **`ServiceReference`** or service properties are required, the `@Reference.service` can declare the type of service when it can't be discovered by the type of Injection point, e.g.

```
class Fum {
  @Inject
  @Reference(service = Foo.class)
  ServiceReference<?> sr;
}
```

This makes it possible to bind `ServiceReference` as well as service properties as `Map<String, Object>`.

5.7.1 Allowing Service Injections in vanilla CDI injection points

In certain scenarios it may be desirable allow injections from the OSGi service registry into plain `@Inject` injection points. This is to allow migration paths from pure JavaEE to configurations where unmodified CDI beans are injected with services from the OSGi service registry. Clearly such situations introduce risks and analysis of the system is necessary beforehand to ensure that no fatal situations can occur by the introduced dynamics.

Support for this ability is provided through the OSGi Beans Description using the **`reference`** element. References declare a target filter which allows matching against OSGi services. These references also define which specific class under which to make the bean available. The service should be cast-able to the specified type. The service will then be instantiated as a bean. These “reference” beans can then match any injection point as the CDI Ceontainer sees fit, e.g.

```
class Fum {
  @Inject
  foo.Foo foo;
}
```

The following reference would define a reference bean that would match the injection point above:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:cdi="http://www.osgi.org/xmlns/cdi/v1.0.0">
  <cdi:reference beanClass=”foo.Foo” target=”(objectClass=foo.Foo)” />
</beans>
```

5.7.2 Service Filters

A sService filtering can be specifieddone using the `@Reference.target` parameter usingfollowing the standard OSGi service filter (LDAP like) syntax:

```
@Reference(target = “(some.key=somevalue)”)
```

As a convenience CDI Qualifiers will be converted to filters:

```
@Qualifier
public @interface SomeKey {
  String value();
}

@Inject
@Reference
@SomeKey(“somevalue”)
MyService service;
```

Conversion of Qualifiers to filters will map to the behavior specified by the converter specification in converting Annotations to `Map<String, String>` along with name mangling rules.

Note: Marker annotations (having no members) convert to an empty map and so those are effectively ignored.

Multiple qualifiers will be combined to create AND filters. In the following example both annotations `@SomeKey` and `@SomeOtherKey` are qualifiers, and the resulting filter is `“(&(foo>=10)(some.key=somevalue)(some.other.key=someothervalue))”`.

```
@Inject
@Reference(target = “(foo>=10)”)
@SomeKey(“somevalue”)
@SomeOtherKey(“someothervalue”)
MyService service;
```

More complex filters are not supported using qualifiers, the standard filter syntax should be used instead with the `@Reference` annotation.

5.8 Handling Instance<?> interface

CDI provides a mechanism for accessing multiple beans with a few limitations. This is known as **contextual instance by programmatic lookup**. This is implemented by means of the interface `javax.enterprise.inject.Instance`. This interface allows an injection point to be created which does not bind a bean tightly to a specific injection but rather to be able to perform dynamic lookup or to iterate over matches, e.g.

```
@Inject
Instance<Foo> foos;

Foo foo = foos.get();
```

When used with the `@Reference`, `Instance` will work. However ordering is not guaranteed, nor is cardinality. It's also possible to map the injection to `ServiceReference` or service properties as `Map<String, Object>` with the same caveats of ordering and cardinality.

This would be one way to accomplish "optional" service dependencies.

5.9 Requiring configuration

Beans can require Configuration Admin Configurations. The @BeanConfiguration annotation is used to specify the pids and policy for the configurations and instances of the configuration are provided through @Inject with the qualifier `@Configuration`. The CDI Ceontainer will only become available when matching configurations are found. This is useful when a bean does not have usable default configuration values. If the configuration object is not available, the behavior is the same as for a unavailable required service dependency. `@Configuration` is supported on all types which could be emitted by the converter specification given a `Configuration` object. The `@BeanConfiguration` annotation accepts an array of String values listing the expected PIDs. The `$` symbol is a placefolder for the default PID which is the name of the bean class by default. This is to prevent having to duplicate the name of the PID.

```
@Inject
@Configuration({"$", "other.pid", "some.other.pid"})
Map<String, Object> props;
```

Type safety is supported by using configuration types:

```
@interface MyConfig {
  int some_property() default 0;
  String some_other_property() default "foo";
}
@Inject
@Configuration({"$", "other.pid", "some.other.pid"})
MyConfig config;
```

`@Configuration` by default implies required configurations. The CDI container will remain in the `WAITING_FOR_CONFIGURATIONS` state until all configurations are available. Configurations can be made optional by specifying the annotation property `required` as `false`.

```
@Configuration(required = false, value = {"$", "other.pid", "some.other.pid"})
```

For missing optional configuration, the CDI container must proceed into the CREATED state. Once optional configuration becomes available the CDI container must step back through the initialization process until once again achieving the CREATED state.

Note: Factory configurations are ignored in this version of the specification.

5.9.1 Allowing Configuration Injections in vanilla CDI injection points

In certain scenarios it may be desirable allow injections from Configuration Admin into plain `@Inject` injection points. This is to allow migration paths from pure JavaEE to configurations where unmodified CDI beans are injected with configuration from Configuration Admin. Clearly such situations introduce risks and analysis of the system is necessary beforehand to ensure that no fatal situations can occur by the introduced dynamics.

Support for this ability is provided through the OSGi Beans Description using the configuration element. Configurations declare a `pid` attribute who's value is a whitespace separated array of values which allows matching against the `service.pid` property of Configuration Admin configurations. These configurations also define the attribute `beanClass` which specifies the class under which to make the bean available. The class should be one which the CDI bundle can load and to which the converter specification describes a conversion from the configuration Dictionary. The configuration dictionary will then be instantiated as a bean of this type. These configuration beans can then match any injection point as the CDI container sees fit, e.g.

```
class Foo {
  @Inject
  org.example.Config config;
}
```

The following configuration would define a configuration bean that would match the injection point:

```
<?xml version="1.0" encoding="UTF-8"?>
<cdi:configuration
  beanClass=”org.example.Config”
  pid=”some.pid”
  xmlns:cdi="http://www.osgi.org/xmlns/cdi/v1.0.0"
/>
```

As with the `@Configuration` annotation, it's possible to make the configuration optional by specifying the attribute `required=”false”` on the configuration element.

```
<?xml version="1.0" encoding="UTF-8"?>
<cdi:configuration
  beanClass=”org.example.Config”
  pid=”some.pid”
  required=”false”
  xmlns:cdi="http://www.osgi.org/xmlns/cdi/v1.0.0"
/>
```

5.10 BundleContext injection

It's possible to inject the BundleContext of the CDI bundle using CDI annotations.

```
@Inject BundleContext context;
```

This will always inject the BundleContext of the bundle that contains the CDI container.

5.11 `BeanManager` service

The implementation will register a service registered under the `javax.enterprise.inject.spi.BeanManager` interface. The BeanManager provides a standard portable intropspective interfaces into the CDI container.

The BeanManager service is registered under the Bundle Context of the associated CDI bundle, as each CDI bundle has its own container. As a result many CdiContainer services will be present in the system.

5.12 `osgi.contract` Capability

The OSGi Enterprise specification version 5 defines the `osgi.contract` capability namespace and RFC 180 defines mappings of JSR-defined technologies to these capabilities. Relevant technologies from RFC 180 to this specification are the `JavaCDI` and `JavaInject` API contracts.

An implementation of this specification is not required to export the associated `javax.**` packages, but if it does, it must also provide the `JavaCDI` and `JavaInject` capabilities in the `osgi.contract` namespace.

5.13 Events

The CDI Container must track all CDI Listener services and keep the listeners updated of the progress of all its managed bundles.

CDI Events must be sent to each registered CDI Listeners service. The service has the following method:

- cdiEvent(CdiEvent) – notify the listener of a new CDI Event synchronously.

5.13.1 CDI Event

The CDI Event supports the following event types:

- CREATING - The CDI extender has started creating a CDI Container for the bundle.
- CREATED - The CDI Container is ready. The application is now running.
- WAITING_FOR_CONFIGURATIONS - The CDI container is waiting for dependent configurations.
- WAITING_FOR_EXTENSIONS – The CDI container is waiting for dependent extensions.
- WAITING_FOR_SERVICES – The CDI container is waiting for dependent services.
- SATISFIED – The CDI container has all it's dependencies satisfied and is completing initialization.
- DESTROYING - The CDI Container is being destroyed because the CDI bundle or CDI extender has stopped.
- DESTROYED - The CDI Container is completely destroyed.
- FAILURE - An error occurred during the creation of the CDI Container.

The CDI Event provides the following methods:

- getBundle() - The CDI bundle
- getCause() - Any occurred exception or null
- getDependencies() - A list of filters that specify the unsatisfied dependencies being waited for.
- getExtenderBundle() - The CDI extender bundle.
- getTimestamp() - The time the event occurred
- getType() - The type of the event.

It should be noted that while CDI Events are published onto the CDI container's event bus, not all events are visible to it since some are fired before and after the CDI container's beans are instantiated. However, CDI Listeners see all events (provided they are registered early enough).

6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined an explanation should be given explaining why this is not applicable in this case.

This section is optional and could also be provided in a separate RFC.

7 Javadoc

23/08/17 9:55 AM

Package Summary		Page
org.osgi.service.cdi	CDI Integration Package Version 1.0.	26
org.osgi.service.cdi.annotations	CDI Integration Package Version 1.0.	34

Package org.osgi.service.cdi

```
@org.osgi.annotation.versioning.Version(value="1.0.0")
```

CDI Integration Package Version 1.0.

See:

Description

Interface Summary		Page
CdiContainer	A CdiContainer object is registered by the CDI extender in the OSGi registry for each managed CDI bundle.	29

Class Summary		Page
CdiConstants	Defines CDI constants.	27
CdiEvent	CdiEvents are sent by the CDI extender and received by registered CdiListener services.	30

Enum Summary		Page
CdiEvent.Type	An enum defining the states of a CDI container.	32

Package org.osgi.service.cdi Description

CDI Integration Package Version 1.0.

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.cdi; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.cdi; version="[1.0,1.1)"
```

Class CdiConstants

org.osgi.service.cdi

```
java.lang.Object
  ↳ org.osgi.service.cdi.CdiConstants
```

```
@org.osgi.annotation.versioning.ProviderType
public class CdiConstants
extends Object
```

Defines CDI constants.

Field Summary		Page
static String	CDI_CAPABILITY_NAME Capability name for CDI Integration.	27
static String	CDI_CONTAINER_STATE The property of the CdiContainer service holding it's state.	27
static String	CDI_EXTENSION_NAMESPACE Namespace name for CDI extension capabilities and requirements.	27
static String	CDI_SPECIFICATION_VERSION Compile time constant for the Specification Version of CDI Integration.	28
static String	REQUIREMENT_BEANS_ATTRIBUTE The 'beans' attribute on the CDI extender requirement.	28
static String	REQUIREMENT_OSGI_BEANS_ATTRIBUTE The 'osgi.beans' attribute on the CDI extender requirement.	28

Field Detail

CDI_CAPABILITY_NAME

```
public static final String CDI_CAPABILITY_NAME = "osgi.cdi"
```

Capability name for CDI Integration.

Used in `Provide-Capability` and `Require-Capability` manifest headers with the `osgi.extender` namespace. For example:

```
Require-Capability: osgi.extender;
 filter:="(&(osgi.extender=osgi.cdi)(version>=1.0)(!(version>=2.0)))"
```

CDI_CONTAINER_STATE

```
public static final String CDI_CONTAINER_STATE = "osgi.cdi.container.state"
```

The property of the CdiContainer service holding it's state.

The property will be updated every time the CdiContainer's state changes. The possible values are defined by `CdiEvent.Type`.

CDI_EXTENSION_NAMESPACE

```
public static final String CDI_EXTENSION_NAMESPACE = "osgi.cdi.extension"
```

Namespace name for CDI extension capabilities and requirements.

Used in `Provide-Capability` and `Require-Capability` manifest headers. For example:

```
Require-Capability: osgi.cdi.extension;
 filter:="(&(osgi.cdi.extension=foo)(version>=1.0)(!(version>=2.0)))"
```

CDI_SPECIFICATION_VERSION

```
public static final String CDI_SPECIFICATION_VERSION = "1.0.0"
```

Compile time constant for the Specification Version of CDI Integration.

Used in `Version` and `Requirement` annotations. The value of this compile time constant will change when the specification version of CDI Integration is updated.

REQUIREMENT_BEANS_ATTRIBUTE

```
public static final String REQUIREMENT_BEANS_ATTRIBUTE = "beans"
```

The 'beans' attribute on the CDI extender requirement.

The value of this attribute is a comma delimited list of bean CDI bean descriptor files to be searched on the `Bundle-ClassPath`. For example:

```
Require-Capability: osgi.extender;\
 filter:="(&(osgi.extender=osgi.cdi)(version>=1.0)(!(version>=2.0)))";\
 beans:List<String>="META-INF/beans.xml"
```

REQUIREMENT_OSGI_BEANS_ATTRIBUTE

```
public static final String REQUIREMENT_OSGI_BEANS_ATTRIBUTE = "osgi.beans"
```

The 'osgi.beans' attribute on the CDI extender requirement.

The value of this attribute is the name of the OSGi Beans Description file. The default value when unspecified is `OSGI-INF/cdi/osgi-beans.xml`. For example:

```
Require-Capability: osgi.extender;\
 filter:="(&(osgi.extender=osgi.cdi)(version>=1.0)(!(version>=2.0)))";\
 osgi.beans="OSGI-INF/cdi/osgi-beans.xml"
```

Interface CdiContainer

org.osgi.service.cdi

`public interface` **`CdiContainer`**

A CdiContainer object is registered by the CDI extender in the OSGi registry for each managed CDI bundle.

Method Summary		Page
BeanManager	**getBeanManager**() When the CdiContainer is READY the BeanManager will be available.	29

Method Detail

getBeanManager

`BeanManager` **`getBeanManager`**()

When the CdiContainer is READY the BeanManager will be available.

Returns:

BeanManager

Class CdiEvent

org.osgi.service.cdi

```
java.lang.Object
  ↳ org.osgi.service.cdi.CdiEvent
```

```
public class CdiEvent
extends Object
```

CdiEvents are sent by the CDI extender and received by registered CdiListener services.

Nested Class Summary		Page
static enum	**CdiEvent.Type** An enum defining the states of a CDI container.	32

Constructor Summary	Page
CdiEvent(CdiEvent.Type type, org.osgi.framework.Bundle bundle, org.osgi.framework.Bundle extenderBundle)	30
CdiEvent(CdiEvent.Type type, org.osgi.framework.Bundle bundle, org.osgi.framework.Bundle extenderBundle, String dependencies)	30
CdiEvent(CdiEvent.Type type, org.osgi.framework.Bundle bundle, org.osgi.framework.Bundle extenderBundle, String dependencies, Throwable cause)	31

Method Summary		Page
org.osgi.framework.Bundle	**getBundle**()	31
Throwable	**getCause**()	31
String	**getDependencies**()	31
org.osgi.framework.Bundle	**getExtenderBundle**()	31
long	**getTimestamp**()	31
CdiEvent.Type	**getType**()	31
String	**toString**()	31

Constructor Detail

CdiEvent

```
public CdiEvent(CdiEvent.Type type,
                org.osgi.framework.Bundle bundle,
                org.osgi.framework.Bundle extenderBundle)
```

CdiEvent

```
public CdiEvent(CdiEvent.Type type,
                org.osgi.framework.Bundle bundle,
                org.osgi.framework.Bundle extenderBundle,
                String dependencies)
```

CdiEvent

```
public CdiEvent(CdiEvent.Type type,
                org.osgi.framework.Bundle bundle,
                org.osgi.framework.Bundle extenderBundle,
                String dependencies,
                Throwable cause)
```

Method Detail

getBundle

```
public org.osgi.framework.Bundle getBundle()
```

Returns:
the bundle who's CDI container triggered this event

---

getCause

```
public Throwable getCause()
```

Returns:
the cause of the event if there was one

---

getExtenderBundle

```
public org.osgi.framework.Bundle getExtenderBundle()
```

Returns:
the bundle of the CDI extender

---

getDependencies

```
public String getDependencies()
```

Returns:
the payload associated with this event

---

getTimestamp

```
public long getTimestamp()
```

Returns:
the timestamp of the event

---

getType

```
public CdiEvent.Type getType()
```

Returns:
the state of this event

---

toString

```
public String toString()
```

Overrides:
`toString` in class `Object`

Enum CdiEvent.Type

org.osgi.service.cdi

```
java.lang.Object
  ↳ java.lang.Enum<CdiEvent.Type>
      ↳ org.osgi.service.cdi.CdiEvent.Type
```

All Implemented Interfaces:

Comparable<CdiEvent.Type>, Serializable

Enclosing class:

CdiEvent

```
public static enum CdiEvent.Type
extends Enum<CdiEvent.Type>
```

An enum defining the states of a CDI container.

Enum Constant Summary	Page
CREATED The CDI container is created and should be fully usable.	33
CREATING The CDI container has started being created.	32
DESTROYED The CDI container is completely destroyed.	33
DESTROYING The CDI container has started being destroyed.	33
FAILURE The CDI container has suffered a failure and will be destroyed.	33
SATISFIED The CDI container is satisfied and resuming construction.	33
WAITING_FOR_CONFIGURATIONS The CDI container is waiting for dependent configurations.	33
WAITING_FOR_EXTENSIONS The CDI container is waiting for dependent extensions.	33
WAITING_FOR_SERVICES The CDI container is waiting for dependent services.	33

Method Summary		Page
static CdiEvent.Type	**valueOf**(String name)	33
static CdiEvent.Type[]	**values**()	33

Enum Constant Detail

CREATING

```
public static final CdiEvent.Type CREATING
```

The CDI container has started being created.

CREATED
public static final CdiEvent.Type **CREATED**

The CDI container is created and should be fully usable.

DESTROYING
public static final CdiEvent.Type **DESTROYING**

The CDI container has started being destroyed.

DESTROYED
public static final CdiEvent.Type **DESTROYED**

The CDI container is completely destroyed.

WAITING_FOR_CONFIGURATIONS
public static final CdiEvent.Type **WAITING_FOR_CONFIGURATIONS**

The CDI container is waiting for dependent configurations.

WAITING_FOR_EXTENSIONS
public static final CdiEvent.Type **WAITING_FOR_EXTENSIONS**

The CDI container is waiting for dependent extensions.

WAITING_FOR_SERVICES
public static final CdiEvent.Type **WAITING_FOR_SERVICES**

The CDI container is waiting for dependent services.

SATISFIED
public static final CdiEvent.Type **SATISFIED**

The CDI container is satisfied and resuming construction.

FAILURE
public static final CdiEvent.Type **FAILURE**

The CDI container has suffered a failure and will be destroyed.

Method Detail

values
public static CdiEvent.Type[] **values**()

valueOf
public static CdiEvent.Type **valueOf**(String name)

Package org.osgi.service.cdi.annotations

```
@org.osgi.annotation.versioning.Version(value="1.0.0")
```

CDI Integration Package Version 1.0.

See:

Description

Interface Summary		Page
ServiceEvent	This interface is used in CDI Observer methods to watch OSGi service events.	55
ServiceEvent.Consumer	A functional interface who's method is `accept(Object)` describing an operation to be executed during a service event.	57

Enum Summary		Page
ConfigurationPolicy	Configuration Policy for the `Component` annotation.	41
ReferenceCardinality	Cardinality for the `Reference` annotation.	47
ReferencePolicy	Policy for the `Reference` annotation.	49
ReferencePolicyOption	Policy option for the `Reference` annotation.	51
ReferenceScope	Reference scope for the `Reference` annotation.	53
ServicePolicy	Service policy for the `Reference` annotation.	58

Annotation Types Summary		Page
BundleScoped	Annotation used to indicate a CDI bean marked with has an OSGi `service.scope = bundle`.	36
Component	Identify the annotated CDI bean class as a Service Component.	37
Configuration	Annotation used with `Inject` in order to have `Component` configuration properties injected into CDI beans.	39
FactoryScoped	CDI scope annotation used to indicate a component who's scope is managed by a `ManagedServiceFactory`.	43
Reference	Annotation used to annotate a CDI injection point informing the CDI container that the injection should apply a service obtained from the OSGi registry.	44

Package org.osgi.service.cdi.annotations Description

CDI Integration Package Version 1.0.

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.cdi.annotations; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.cdi.annotations; version="[1.0,1.1)"
```

Annotation Type BundleScoped

org.osgi.service.cdi.annotations

```
@Documented
@org.osgi.annotation.bundle.Requirement(namespace="osgi.extender",
            name="osgi.cdi",
            version="1.0.0")
@Retention(value=RetentionPolicy.RUNTIME)
@Scope
@Target(value=ElementType.TYPE)
public @interface BundleScoped
```

Annotation used to indicate a CDI bean marked with has an OSGi `service.scope = bundle`.

By default CDI beans annotated with only support:

- or - which translates to `service.scope = singleton`
- - which translates to `service.scope = bundle`
- - which translates to `service.scope = prototype`

Any other scope will result in a definition error.

By default CDI beans without a scope have scope which means developers should take care that their services have `prototype` scope by default.

Annotation Type Component

org.osgi.service.cdi.annotations

```
@Retention(value=RetentionPolicy.RUNTIME)
@Target(value=ElementType.TYPE)
@org.osgi.annotation.bundle.Requirement(namespace="osgi.extender",
            name="osgi.cdi",
            version="1.0.0")
public @interface Component
```

Identify the annotated CDI bean class as a Service Component.

Required Element Summary		Page
String	**name** The name of this Component.	37
String[]	**property** Properties for this Component.	38
Class<?>[]	**service** The types under which to register this Component as a service.	37
ServicePolicy	**servicePolicy** The service policy used for the Component.	38

Element Detail

name

```
public abstract String name
```

The name of this Component.

If not specified, the name of this Component is the fully qualified type name of the class being annotated.

Default:

""

service

```
public abstract Class<?>[] service
```

The types under which to register this Component as a service.

The result of this value depends on the value of the service policy.

1. If service policy is ServicePolicy.DERIVED and service is NOT specified: the component is published using all directly implemented interfaces. If there are no directly implemented interfaces, the class of the Component is used.
2. If service policy is ServicePolicy.DERIVED and service is specified: the component is published using the specified types.
3. If service policy is ServicePolicy.NONE: the value is ignored and the component is not published as a service.

Default:

{}

servicePolicy

```
public abstract ServicePolicy servicePolicy
```

The service policy used for the Component.

If not specified the Component will not be published as a service.

Default:

ServicePolicy.DEFAULT

---

property

```
public abstract String[] property
```

Properties for this Component.

Each property string is specified as `"name=value"`. The type of the property value can be specified in the name as `name:type=value`. The type must be one of the property types supported by the `type` attribute of the `property` element of a Component description.

To specify a property with multiple values, use multiple name, value pairs. For example, `{"foo=bar", "foo=baz"}`.

Default:

{}

Annotation Type Configuration

org.osgi.service.cdi.annotations

```
@Qualifier
@Target(value={
  ElementType.FIELD,
  ElementType.METHOD,
  ElementType.PARAMETER,
  ElementType.TYPE
})
@Retention(value=RetentionPolicy.RUNTIME)
@Documented
@org.osgi.annotation.bundle.Requirement(namespace="osgi.extender",
            name="osgi.cdi",
            version="1.0.0")
public @interface Configuration
```

Annotation used with `Inject` in order to have `Component` configuration properties injected into CDI beans. Properties are a combination of the Map defined by `Component.property()` overlaid by properties provided through Configuration Admin in association with the configuration PIDs defined by `Component.value()`.

Field Summary		Page
String	**NAME** Special string representing the name of this Component.	39

Required Element Summary		Page
ConfigurationPolicy	**configurationPolicy** The configuration policy of this Configuration.	39
String[]	**value** The configuration PIDs for the configuration of this Component.	40

Field Detail

NAME

```
public static final String NAME = "$"
```

Special string representing the name of this Component.

This string can be used in `value()` to specify the name of the component as a configuration PID. For example:

```
@Configuration({"com.acme.system", Component.NAME})
```

Element Detail

configurationPolicy

```
public abstract ConfigurationPolicy configurationPolicy
```

The configuration policy of this Configuration.

Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service. Corresponding configurations are Configuration objects where the PIDs equal `value`, or the `name` of the component if no values are specified.

If not specified, the configuration policy is `OPTIONAL`

Default:

ConfigurationPolicy.DEFAULT

---

value

```
public abstract String[] value
```

The configuration PIDs for the configuration of this Component.

Each value specifies a configuration PID for this Component.

If no value is specified, the name of this Component is used as the configuration PID of this Component.

A special string ("$") can be used to specify the name of the component as a configuration PID. The NAME constant holds this special string. For example:

```
@Configuration({"com.acme.system", Component.NAME})
```

Default:

{ "$" }

Enum ConfigurationPolicy

org.osgi.service.cdi.annotations

```
java.lang.Object
  ↳ java.lang.Enum<ConfigurationPolicy>
      ↳ org.osgi.service.cdi.annotations.ConfigurationPolicy
```

All Implemented Interfaces:

Comparable<ConfigurationPolicy>, Serializable

```
public enum ConfigurationPolicy
extends Enum<ConfigurationPolicy>
```

Configuration Policy for the `Component` annotation.

Controls whether component configurations must be satisfied depending on the presence of a corresponding Configuration object in the OSGi Configuration Admin service. A corresponding configuration is a Configuration object where the PID is the name of the component.

Since:

1.1

Enum Constant Summary	Page
DEFAULT The value indicating that no choice was made in which case the calculated value or default behavior should take effect.	41
IGNORE Always allow the component configuration to be satisfied and do not use the corresponding Configuration object even if it is present.	42
OPTIONAL Use the corresponding Configuration object if present but allow the component to be satisfied even if the corresponding Configuration object is not present.	42
REQUIRE There must be a corresponding Configuration object for the component configuration to become satisfied.	42

Method Summary		Page
static ConfigurationPolicy	**get**(String input) Get a ConfigurationPolicy instance by value rather than by name.	42
String	**toString**()	42
static ConfigurationPolicy	**valueOf**(String name)	42
static ConfigurationPolicy[]	**values**()	42

Enum Constant Detail

DEFAULT

```
public static final ConfigurationPolicy DEFAULT
```

The value indicating that no choice was made in which case the calculated value or default behavior should take effect.

The default behavior is OPTIONAL.

OPTIONAL
```
public static final ConfigurationPolicy OPTIONAL
```

Use the corresponding Configuration object if present but allow the component to be satisfied even if the corresponding Configuration object is not present.

REQUIRE
```
public static final ConfigurationPolicy REQUIRE
```

There must be a corresponding Configuration object for the component configuration to become satisfied.

IGNORE
```
public static final ConfigurationPolicy IGNORE
```

Always allow the component configuration to be satisfied and do not use the corresponding Configuration object even if it is present.

Method Detail

values
```
public static ConfigurationPolicy[] values()
```

valueOf
```
public static ConfigurationPolicy valueOf(String name)
```

get
```
public static ConfigurationPolicy get(String input)
```

Get a ConfigurationPolicy instance by value rather than by name.

Parameters:
`input` - a non-null value
Returns:
the ConfigurationPolicy matching the value
Throws:
`NullPointerException` - if the input is null
`IllegalArgumentException` - on invalid input

toString
```
public String toString()
```

Overrides:
`toString` in class `Enum`

Annotation Type FactoryScoped

org.osgi.service.cdi.annotations

```
@Documented
@org.osgi.annotation.bundle.Requirement(namespace="osgi.extender",
            name="osgi.cdi",
            version="1.0.0")
@Retention(value=RetentionPolicy.RUNTIME)
@Scope
@Target(value=ElementType.TYPE)
public @interface FactoryScoped
```

CDI scope annotation used to indicate a component who's scope is managed by a `ManagedServiceFactory`. The value of the annotation is the factory PID to be used.

Required Element Summary		Page
String	**value** The factory PID.	43

Element Detail

value

```
public abstract String value
```

The factory PID.

Returns:
the factory PID

Annotation Type Reference

org.osgi.service.cdi.annotations

```
@Qualifier
@Target(value={
  ElementType.FIELD,
  ElementType.METHOD,
  ElementType.PARAMETER,
  ElementType.TYPE
})
@Retention(value=RetentionPolicy.RUNTIME)
@Documented
@org.osgi.annotation.bundle.Requirement(namespace="osgi.extender",
            name="osgi.cdi",
            version="1.0.0")
public @interface Reference
```

Annotation used to annotate a CDI injection point informing the CDI container that the injection should apply a service obtained from the OSGi registry.

Required Element Summary		Page
ReferenceCardinality	**cardinality** The cardinality of this reference.	45
String	**name** The name of this reference.	44
ReferencePolicy	**policy** The policy for this reference.	45
ReferencePolicyOption	**policyOption** The policy option for this reference.	46
ReferenceScope	**scope** The reference scope for this reference.	46
Class<?>	**service** The type of the service for this reference.	45
String	**target** The target property for this reference.	45

Element Detail

name

```
public abstract String name
```

The name of this reference.

If not specified, the name of this reference is based upon how this annotation is used:

- Annotated constructor parameter - The name of the reference is the simple, raw class name of the field suffixed by the positional index of the parameter.
- Annotated field - The name of the reference is the field name.
- Annotated method parameter - The name of the reference is the simple, raw class name of the field.

Default:

""

service

`public abstract Class<?> service`

The type of the service for this reference.

If not specified, the type of the service for this reference is based upon how this annotation is used:

- Annotated field - The type of the service is based upon the type of the field being annotated and the cardinality of the reference. If the cardinality is either `0..n`, or `1..n`, the type of the field must be one of `java.util.Collection`, `java.util.List`, or a subtype of `java.util.Collection` so the type of the service is the generic type of the collection. Otherwise, the type of the service is the type of the field.
- Annotated constructor or method parameter - The type of the service is based upon the type of the parameter being annotated and the cardinality of the reference. If the cardinality is either `0..n`, or `1..n`, the type of the parameter must be one of `java.util.Collection`, `java.util.List`, or a subtype of `java.util.Collection` so the type of the service is the generic type of the collection. Otherwise, the type of the service is the type of the parameter.

Default:
Object.class

---

cardinality

`public abstract ReferenceCardinality cardinality`

The cardinality of this reference.

If not specified, the cardinality of this reference is based upon how this annotation is used:

- Annotated field - The cardinality is based on the type of the field. If the type is either `java.util.Collection`, `java.util.List`, or a subtype of `java.util.Collection`, the cardinality is `0..n`. Otherwise the cardinality is `1..1`.
- Annotated constructor or method parameter - The cardinality is based on the type of the parameter. If the type is either `java.util.Collection`, `java.util.List`, or a subtype of `java.util.Collection`, the cardinality is `0..n`. Otherwise the cardinality is `1..1`.

Default:
ReferenceCardinality.DEFAULT

---

policy

`public abstract ReferencePolicy policy`

The policy for this reference.

If not specified, the policy of this reference is based upon how this annotation is used:

- Annotated method - The policy is `STATIC`.
- Annotated field - The policy is based on the modifiers of the field. If the field is declared `volatile`, the policy is `ReferencePolicy.DYNAMIC`. Otherwise the policy is `STATIC`.
- Annotated constructor parameter - The policy is `STATIC`. Constructor parameters must always assume `STATIC` policy.

Default:
ReferencePolicy.DEFAULT
See Also:
"The policy attribute of the reference element of a Component Description."

---

target

`public abstract String target`

The target property for this reference.

If not specified, no target property is set.

Default:
""

---

policyOption

```
public abstract ReferencePolicyOption policyOption
```

The policy option for this reference.

If not specified, the `RELUCTANT` reference policy option is used.

Default:
ReferencePolicyOption.DEFAULT

---

scope

```
public abstract ReferenceScope scope
```

The reference scope for this reference.

If not specified, the `ReferenceScope.BUNDLE` reference scope is used.

Default:
ReferenceScope.DEFAULT

Enum ReferenceCardinality

org.osgi.service.cdi.annotations

```
java.lang.Object
  ↳ java.lang.Enum<ReferenceCardinality>
      ↳ org.osgi.service.cdi.annotations.ReferenceCardinality
```

All Implemented Interfaces:

Comparable<ReferenceCardinality>, Serializable

```
public enum ReferenceCardinality
extends Enum<ReferenceCardinality>
```

Cardinality for the Reference annotation.

Specifies if the reference is optional and if the component implementation support a single bound service or multiple bound services.

Enum Constant Summary	Page
AT_LEAST_ONE The reference is mandatory and multiple.	48
DEFAULT The value indicating that no choice was made in which case the calculated value or default behavior should take effect.	47
MANDATORY The reference is mandatory and unary.	48
MULTIPLE The reference is optional and multiple.	48
OPTIONAL The reference is optional and unary.	48

Method Summary		Page
static ReferenceCardinality	**get**(String input) Get a ReferenceCardinality instance by value rather than by name.	48
String	**toString**()	48
static ReferenceCardinality	**valueOf**(String name)	48
static ReferenceCardinality[]	**values**()	48

Enum Constant Detail

DEFAULT

```
public static final ReferenceCardinality DEFAULT
```

The value indicating that no choice was made in which case the calculated value or default behavior should take effect.

The default behavior is MANDATORY.

OPTIONAL
```
public static final ReferenceCardinality OPTIONAL
```

The reference is optional and unary. That is, the reference has a cardinality of `0..1`.

MANDATORY
```
public static final ReferenceCardinality MANDATORY
```

The reference is mandatory and unary. That is, the reference has a cardinality of `1..1`.

MULTIPLE
```
public static final ReferenceCardinality MULTIPLE
```

The reference is optional and multiple. That is, the reference has a cardinality of `0..n`.

AT_LEAST_ONE
```
public static final ReferenceCardinality AT_LEAST_ONE
```

The reference is mandatory and multiple. That is, the reference has a cardinality of `1..n`.

Method Detail

values
```
public static ReferenceCardinality[] values()
```

valueOf
```
public static ReferenceCardinality valueOf(String name)
```

get
```
public static ReferenceCardinality get(String input)
```

Get a ReferenceCardinality instance by value rather than by name.

Parameters:
`input` - a non-null value
Returns:
the ReferenceCardinality matching the value
Throws:
`NullPointerException` - if the input is null
`IllegalArgumentException` - on invalid input

toString
```
public String toString()
```

Overrides:
`toString` in class `Enum`

Enum ReferencePolicy

org.osgi.service.cdi.annotations

```
java.lang.Object
  ↳ java.lang.Enum<ReferencePolicy>
      ↳ org.osgi.service.cdi.annotations.ReferencePolicy
```

All Implemented Interfaces:

Comparable<ReferencePolicy>, Serializable

```
public enum ReferencePolicy
extends Enum<ReferencePolicy>
```

Policy for the Reference annotation.

Enum Constant Summary	Page
DEFAULT The value indicating that no choice was made in which case the calculated value or default behavior should take effect.	49
DYNAMIC The dynamic policy is slightly more complex since the component implementation must properly handle changes in the set of bound services.	50
STATIC The static policy is the most simple policy and is the default policy.	49

Method Summary		Page
static ReferencePolicy	**get**(String input) Get a ReferencePolicy instance by value rather than by name.	50
String	**toString**()	50
static ReferencePolicy	**valueOf**(String name)	50
static ReferencePolicy[]	**values**()	50

Enum Constant Detail

DEFAULT

```
public static final ReferencePolicy DEFAULT
```

The value indicating that no choice was made in which case the calculated value or default behavior should take effect.

The default behavior is STATIC.

STATIC

```
public static final ReferencePolicy STATIC
```

The static policy is the most simple policy and is the default policy. A CDI component instance never sees any of the dynamics. CDI Containers are deactivated before any bound service for a reference having a

static policy becomes unavailable. If a target service is available to replace the bound service which became unavailable, the CDI Container must be reactivated and bound to the replacement service.

DYNAMIC

```
public static final ReferencePolicy DYNAMIC
```

The dynamic policy is slightly more complex since the component implementation must properly handle changes in the set of bound services. With the dynamic policy, the CDI Extender can change the set of bound services without deactivating a CDI Container. If the component uses the event strategy to access services, then the component instance will be notified of changes in the set of bound services by calls to the observer methods.

Method Detail

values

```
public static ReferencePolicy[] values()
```

valueOf

```
public static ReferencePolicy valueOf(String name)
```

get

```
public static ReferencePolicy get(String input)
```

Get a `ReferencePolicy` instance by value rather than by name.

Parameters:

`input` - a non-null value

Returns:

the `ReferencePolicy` matching the value

Throws:

`NullPointerException` - if the input is null

`IllegalArgumentException` - on invalid input

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Enum ReferencePolicyOption

org.osgi.service.cdi.annotations

```
java.lang.Object
  ↳ java.lang.Enum<ReferencePolicyOption>
      ↳ org.osgi.service.cdi.annotations.ReferencePolicyOption
```

All Implemented Interfaces:

Comparable<ReferencePolicyOption>, Serializable

```
public enum ReferencePolicyOption
extends Enum<ReferencePolicyOption>
```

Policy option for the `Reference` annotation.

Enum Constant Summary	Page
DEFAULT The value indicating that no choice was made in which case the calculated value or default behavior should take effect.	51
GREEDY The greedy policy option is a valid policy option for both `static` and `dynamic` reference policies.	52
RELUCTANT The reluctant policy option is the default policy option for both `static` and `dynamic` reference policies.	51

Method Summary		Page
static ReferencePolicyOption	**get**(String input) Get a `ReferencePolicyOption` instance by value rather than by name.	52
String	**toString**()	52
static ReferencePolicyOption	**valueOf**(String name)	52
static ReferencePolicyOption[]	**values**()	52

Enum Constant Detail

DEFAULT

```
public static final ReferencePolicyOption DEFAULT
```

The value indicating that no choice was made in which case the calculated value or default behavior should take effect.

The default behavior is `RELUCTANT`.

RELUCTANT

```
public static final ReferencePolicyOption RELUCTANT
```

The reluctant policy option is the default policy option for both `static` and `dynamic` reference policies. When a new target service for a reference becomes available, references having the reluctant policy option for the static policy or the dynamic policy with a unary cardinality will ignore the new target service. References having the dynamic policy with a multiple cardinality will bind the new target service.

GREEDY

```
public static final ReferencePolicyOption GREEDY
```

The greedy policy option is a valid policy option for both `static` and `dynamic` reference policies. When a new target service for a reference becomes available, references having the greedy policy option will bind the new target service.

Method Detail

values

```
public static ReferencePolicyOption[] values()
```

valueOf

```
public static ReferencePolicyOption valueOf(String name)
```

get

```
public static ReferencePolicyOption get(String input)
```

Get a `ReferencePolicyOption` instance by value rather than by name.

Parameters:

`input` - a non-null value

Returns:

the `ReferencePolicyOption` matching the value

Throws:

`NullPointerException` - if the input is null

`IllegalArgumentException` - on invalid input

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Enum ReferenceScope

org.osgi.service.cdi.annotations

```
java.lang.Object
  ↳ java.lang.Enum<ReferenceScope>
      ↳ org.osgi.service.cdi.annotations.ReferenceScope
```

All Implemented Interfaces:

Comparable<ReferenceScope>, Serializable

```
public enum ReferenceScope
extends Enum<ReferenceScope>
```

Reference scope for the `Reference` annotation.

Enum Constant Summary	Page
BUNDLE A single service object is used for all references to the service in this bundle.	53
DEFAULT The value indicating that no choice was made in which case the calculated value or default behavior should take effect.	53
PROTOTYPE Bound services must have prototype service scope.	54
SINGLETON Bound services must have singleton service scope.	54

Method Summary		Page
static ReferenceScope	**get**(String input) Get a ReferenceScope instance by value rather than by name.	54
String	**toString**()	54
static ReferenceScope	**valueOf**(String name)	54
static ReferenceScope[]	**values**()	54

Enum Constant Detail

DEFAULT

```
public static final ReferenceScope DEFAULT
```

The value indicating that no choice was made in which case the calculated value or default behavior should take effect.

The default behavior is BUNDLE.

BUNDLE

```
public static final ReferenceScope BUNDLE
```

A single service object is used for all references to the service in this bundle.

PROTOTYPE

```
public static final ReferenceScope PROTOTYPE
```

Bound services must have prototype service scope. Each instance of the bean with this reference can receive a unique instance of the service.

SINGLETON

```
public static final ReferenceScope SINGLETON
```

Bound services must have singleton service scope. Each instance of the bean with this reference will receive the same instance of the service.

Method Detail

values

```
public static ReferenceScope[] values()
```

valueOf

```
public static ReferenceScope valueOf(String name)
```

get

```
public static ReferenceScope get(String input)
```

Get a `ReferenceScope` instance by value rather than by name.

Parameters:

`input` - a non-null value

Returns:

the `ReferenceScope` matching the value

Throws:

`NullPointerException` - if the input is null

`IllegalArgumentException` - on invalid input

toString

```
public String toString()
```

Overrides:

`toString` in class `Enum`

Interface ServiceEvent

org.osgi.service.cdi.annotations

Type Parameters:

`T` - the service argument type.

`public interface` **`ServiceEvent`**

This interface is used in CDI Observer methods to watch OSGi service events.

The type parameter is the service argument type and can be one of the following:

- service type
- `ServiceReference`
- `ServiceObjects`
- properties (`Map`)
- tuple of properties (`Map`) as key, service type as value (`Map.Entry`)

e.g.

```
void trackFoos(@Observes @Reference ServiceEvent<Foo> event) {
  event.adding(
   foo -> // perform add
  ).removed(
   foo -> // perform remove
  );
}
```

Nested Class Summary		Page
static interface	**ServiceEvent.Consumer** A functional interface who's method is ServiceEvent.Consumer.accept(Object) describing an operation to be executed during a service event.	57

Method Summary		Page
ServiceEvent<T>	**adding**(ServiceEvent.Consumer<T> consumer) An operation to be invoked upon adding a service to be tracked.	55
ServiceEvent<T>	**modified**(ServiceEvent.Consumer<T> consumer) An operation to be invoked upon modification of a tracked service's properties.	56
ServiceEvent<T>	**removed**(ServiceEvent.Consumer<T> consumer) An operation to be invoked upon the removal of a tracked service.	56

Method Detail

adding

ServiceEvent<T> **adding**(ServiceEvent.Consumer<T> consumer)

An operation to be invoked upon adding a service to be tracked.

Parameters:

`consumer` - an operation to be invoked upon adding a service to be tracked

Returns:

the service event

---

modified

ServiceEvent<T> **modified**(ServiceEvent.Consumer<T> consumer)

An operation to be invoked upon modification of a tracked service's properties.

Parameters:

`consumer` - an operation to be invoked upon modification of a tracked service's properties

Returns:

the service event

---

removed

ServiceEvent<T> **removed**(ServiceEvent.Consumer<T> consumer)

An operation to be invoked upon the removal of a tracked service.

Parameters:

`consumer` - an operation to be invoked upon the removal of a tracked service

Returns:

the service event

Interface ServiceEvent.Consumer

org.osgi.service.cdi.annotations

Type Parameters:

`T` - the service argument type

Enclosing class:

ServiceEvent

```
public static interface ServiceEvent.Consumer
```

A functional interface who's method is `accept(Object)` describing an operation to be executed during a service event.

Method Summary		Page
void	**accept**(T t) Performs this operation with the given service argument.	57

Method Detail

accept

```
void accept(T t)
```

Performs this operation with the given service argument.

Parameters:

`t` - the service argument

Enum ServicePolicy

org.osgi.service.cdi.annotations

```
java.lang.Object
  ↳ java.lang.Enum<ServicePolicy>
      ↳ org.osgi.service.cdi.annotations.ServicePolicy
```

All Implemented Interfaces:

Comparable<ServicePolicy>, Serializable

```
public enum ServicePolicy
extends Enum<ServicePolicy>
```

Service policy for the `Reference` annotation.

Enum Constant Summary	Page
DEFAULT The value indicating that no choice was made in which case the calculated value or default behavior should take effect.	58
DERIVED The derived policy means that the types under which the component will be published is derived by using the following algorithm: • The `@Component.service` is set - Use the types listed • The `@Component.service` is not set - Use the directly implemented interfaces.	58
NONE The none policy means that the component is not to be published as a service.	59

Method Summary		Page
static ServicePolicy	**get**(String input) Get a ServicePolicy instance by value rather than by name.	59
String	**toString**()	59
static ServicePolicy	**valueOf**(String name)	59
static ServicePolicy[]	**values**()	59

Enum Constant Detail

DEFAULT

```
public static final ServicePolicy DEFAULT
```

The value indicating that no choice was made in which case the calculated value or default behavior should take effect.

The default behavior is NONE.

DERIVED

```
public static final ServicePolicy DERIVED
```

The derived policy means that the types under which the component will be published is derived by using the following algorithm:

- The `@Component.service` is set - Use the types listed
- The `@Component.service` is not set - Use the directly implemented interfaces. If no interfaces, use concrete type.

NONE

```
public static final ServicePolicy NONE
```

The none policy means that the component is not to be published as a service. If `@Component.service` is set it is ignored. This is the default policy.

Method Detail

values

```
public static ServicePolicy[] values()
```

valueOf

```
public static ServicePolicy valueOf(String name)
```

get

```
public static ServicePolicy get(String input)
```

Get a `ServicePolicy` instance by value rather than by name.

Parameters:

`input` - a non-null value

Returns:

the `ServicePolicy` matching the value

Throws:

`NullPointerException` - if the input is null

`IllegalArgumentException` - on invalid input

toString

```
public String toString()
```

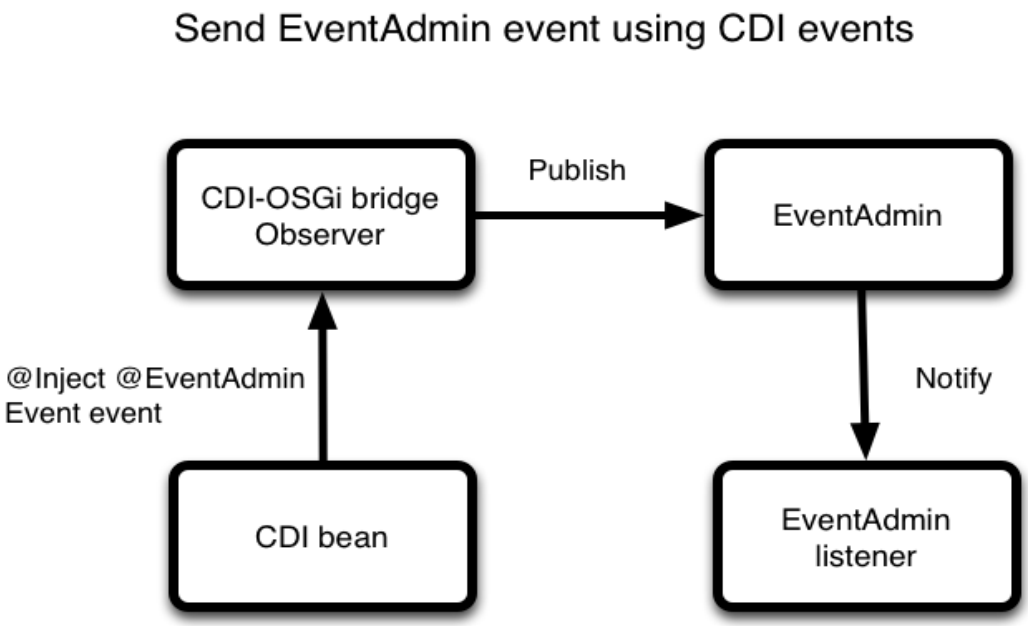
Overrides:

`toString` in class `Enum`

8 Considered Alternatives

8.1 EventAdmin integration

This section has moved into the Considered Alternatives chapter as it's postponed to a later release.



EventAdmin and CDI events are conceptually similar. The CDI programming model is much more user-friendly however. EventAdmin events can both be produced and observed using CDI annotations. Because not every event should be published to EventAdmin the developer has to use the @EventAdmin annotation. The value of this annotation should also contain the name of the *Topic,* or alternatively you can define your own qualifier that extends @EventAdmin to define the Topic.

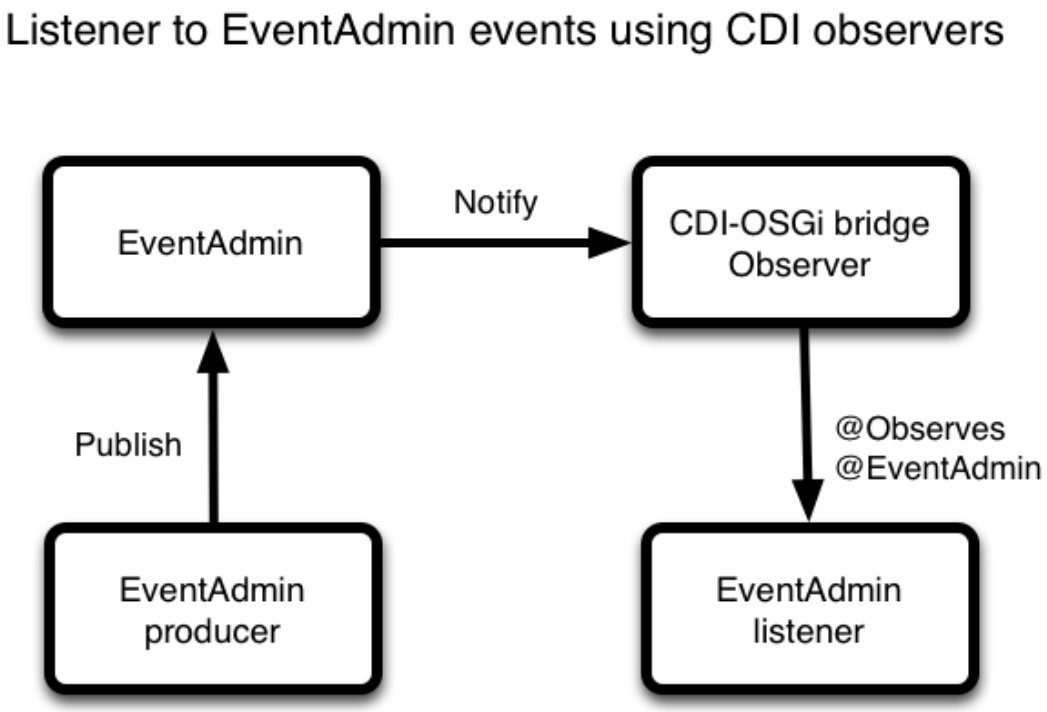
The following is an example of using the @EventAdmin qualifier.

```
@Inject @EventAdmin("MyTopic") Event<MyEvent> event;
public void send() {
    event.fire(new MyEvent("example"));                    }
}
```

The following is an example of extending the EventAdmin qualifier.

```
@EventAdmin
public @interface Demo {
   String value();
}
@Inject @Demo Event<MyEvent> event;
public void send() {
    event.fire(new MyEvent("example"));                    }
}
```

The CDI-OSGi bridge observes @EventAdmin events and republish them as EventAdmin events.



EventAdmin events can be observed using the CDI @Observes annotation. Similar to publishing EventAdmin events we need the @EventAdmin qualifier to specify the Topic name.

```
public class EventExample {
public void process(@Observes @EventAdmin("MyTopic") MyEvent event) {}
}
```

Alternatively, similar with publishing events, a qualifier can be used to define the topic name.

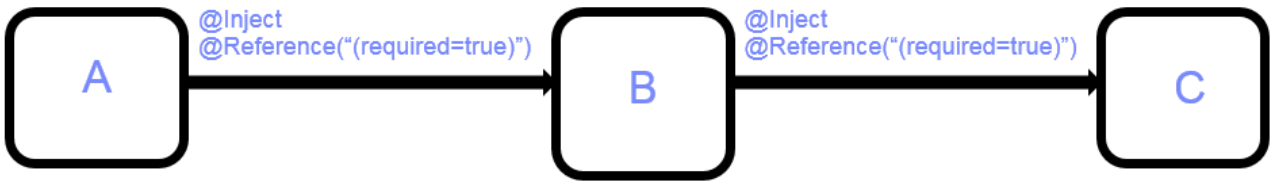
```
@EventAdmin
public @interface Demo {
  String value();
}

public class EventExample {
  public void process(@Observes @Demo MyEvent event) {}
}
```

8.2 Required dependencies (@Service only)

This was moved to considered alternatives because all References will be required by default.

Require a dependency; the component will not be registered when a required dependency is not available. This is the default behavior. When all dependencies becomes available, the component will be registered. If at some point during runtime a required dependency becomes unavailable, the component will be deregistered again. This model handles service dynamics correctly, while the code doesn't have to handle the case where dependencies are not available.

As an example we have the dependency structure:



When C becomes unavailable, B will become unavailable as well, and so will A. When C becomes available again, so will be B, and so will A.

8.3 Additional OSGi CDI Scopes

The additional OSGi CDI scopes was moved to considered alternatives because it was deamed acceptable that the OSGi service scopes of singleton, bundle, and prototype could be mapped to @Singleton, @ApplicationScoped, and @Dependent.

The following table lists OSGi-defined scopes and associated behavior. These scopes are only relevant for CDI beans registered in the OSGi Service Registry:

`@BundleScoped`	Maps to OSGi Service Factory
`@PrototypeScoped`	Maps to OSGi Prototype Service Factory
`@SingletonScoped`	

8.4 Service and bundle registration observers

This was moved to considered alternatives due to the removal of the dynamic service update requirements.

In most situations service dependencies are injected directly into a field. Sometimes some extra code needs to be executed however. This can be done using callback methods. There is a callback method for service registration and a callback for service deregistration. The parameters of the observer method should be the type of the service.

Only events fired while the bundle containing the observer methods are delivered. Events fired while the bundle was not active are ignored.

```
void serviceAdded(@Observes @ServiceAdded SomeService) ;
void serviceRemoved(@Observes @ServiceRemoved  SomeService) ;
void serviceModified(@Observes @ServiceModified  SomeService) ;
```

In some cases it's useful to also have access to the ServiceReference representing a service. For this case a special event type is introduced:

```
public class ServiceCdiEvent<T> {
  private ServiceReference<T> reference;
  private T service;
  // constructor and getters etc.
}

void serviceAdded(@Observes @ServiceAdded ServiceCdiEvent<SomeService> event)
void serviceRemoved(@Observes @ServiceRemoved ServiceCdiEvent<SomeService> event)
```

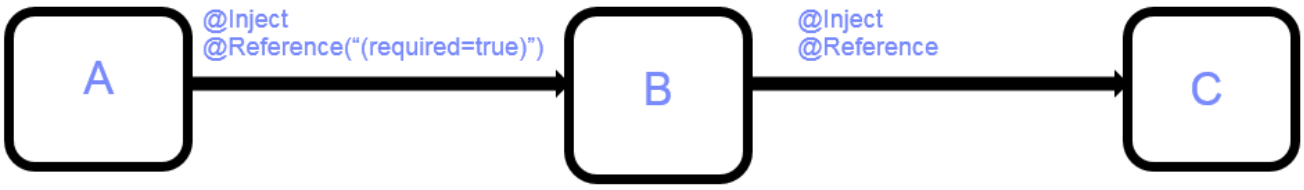
The @ServiceAdded and @ServiceRemoved annotations have the same service filtering semantics as @Reference described in the following section.

8.5 Optional Dependencies

To deal with service dynamics a proxy is injected instead of the real reference to the service. The proxy is by default a null-object, every method invocation should return null. The code that uses an optional dependency must deal with the possibility that the service is not available and handle null values properly.

Alternatively the @Reference supports a configuration parameter “proxyType” that can be used to configure the proxy to throw a org.osgi.cdi.ServiceNotAvailableException when the proxy is invoked while the service is not available. This way clients can handle null values returned by the real service differently than the situation where the service is not available.

Optional and required dependencies can be mixed in a single component and in the dependency graph of a component. Take the following example again, note that the dependency on C is now optional.



When C becomes unavailable, B should still be available, and therefore A as well. When B becomes unavailable, A should still become unavailable as well. Because the dependency on C is now optional, the code of B should be handing the fact that method invocations on C might return null or throw exceptions.

CDI does not support dynamic dependencies. All beans must be registered and all dependencies must be resolved at container startup. For normal CDI beans the container should resolve @Inject @Reference injection points immediately with proxies as described for optional dependencies. Required dependencies are not required to be implemented by the container for normal CDI beans; the container may throw an exception to inform the developer that the required attribute is not supported. A CDI container may implement required dependencies for normal CDI beans as well too however.

For components registered with the @Service annotation, required dependencies must be supported. An @Service with unavailable dependencies must not be registered to the OSGi service registry, and it's @PostConstruct method must not be invoked. If the component was active before the dependency became unavailable, the @PreDestory method must be called and the service must be de-registered from the service registry.

Because OSGi services are dynamic, a developer should make the explicit choice to inject beans using the OSGi service registry by using the @Reference qualifier. If the @Reference qualifier is not used, the container should not query the service registry. @Reference can be used on fields or method parameters.

8.5.1 Replay

The CDI Extender must remember the last CDI Event for each ready bundle that it manages. The replay event must be delivered to the CDI Listener service as the first event, before any other event is delivered , during the registration of the CDI Listener service. The CDIEvent object for a replay event must return true for the isReplay() method is this situation, and false in all other situations.

9 Security Considerations

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

10 Document Support

10.1 References

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

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

[3]. JSR 299: Contexts and Dependency Injection for Java EE flatform

[4]. JSR 346: Conexts and Dependency Injection for Java EE

[5]. Weld OSGi Reference: http://docs.jboss.org/weld/reference/1.2.0.Beta1/weld-osgi/user-guide/html/ch01.html

10.2 Author's Address

Name	Emily Jiang
Company	IBM
Address	Hursley Lab, UK, IBM
Voice	
e-mail	emijiang@uk.ibm.com

10.3 Acronyms and Abbreviations

10.4 End of Document