RFC 220 Bundle Annotations

Draft

31 Pages

Abstract

In the light of the enRoute project bnd pioneered the use of build time annotations to generate manifest headers. The use of annotations makes it easier to manage manifest headers per component, significantly minimizes errors, and leverages the Java type system to provide content assist in IDEs.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

0.2 Trademarks

0.3 Feedback

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

0.4 Table of Contents

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

`Source code is shown in this typeface.`

0.6 Revision History

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

Revision	Date	Comments
Initial	2016-01-08	Initial draft. BJ Hargrave
2nd draft	2016-01-12	Comments from Madrid CPEG mtg BJ Hargrave
3rd draft	2016-01-21	Clean up after CPEG call BJ Hargrave

1 Introduction

This RFC is in response to RFP 167 Manifest Annotations. The OSGi enRoute project used annotations to define bundle metadata in source code. Bnd then processes the annotations into manifest headers.

This RFC will define OSGi standard annotations in support of those ideas.

2 Application Domain

Bnd/Bndtools provides a tool chain to build bundles and/or OSGi manifests used in most OSGi builds. It is supported in Maven, Gradle, Ant, SBT, etc. Bnd uses information in the class files to create a number of headers in the manifest.

Though the class files contain enough information to find the code dependencies, there are many dependencies that are indirect. For example, extenders are often a requirement to make a bundle function correctly but often have no code dependency between their clients whatsoever. Declarative Services (DS) went out of its way to allow components to be Plain Old Java Objects. The result is that the resolving a closure would not drag in the Declarative Services resulting in a satisfied but rather useless closure.

One of the goals of the enRoute project is to rely on Java and not escape to strings. The Java language is a rather steep cost from the point of view of coding but the type engine makes this cost-effective especially since it enables the IDE to assist the developers. However, when information then gets encoded in strings the advantages are voided and what is left is the cost.

Over the past few years a number of annotations were developed to control the creation of bundle resources. The most popular are the Declarative Services annotations and Metatype annotations. Over time these annotations have made it into the OSGi specifications. The OSGi enRoute project decided to develop a number of *manifest annotations*, annotations whose sole purpose is to generate the manifest headers.

Since the manifest headers for requirements and capabilities are error prone to use, the OSGi enRoute project added manifest annotations for the Require-Capability and Provide-Capability headers to bnd:

- `@RequireCapability(ns,effective,filter,resolution)`
- `@ProvideCapability(ns,name,effective,version,uses,mandatory)`

These annotations can be applied to a type or any other annotation. If applied to a type then the annotation's requirement or capability will be added to the manifest. If applied to an annotation then this annotation is a *customized annotation*. Nothing happens until the annotated annotation is used. When this annotation is applied somewhere, bnd will automatically add the requirement or capability to the manifest.

For example:

```
@RequireCapability(
  ns        = “osgi.extender”,
  filter    = "(&(osgi.extender=osgi.enroute.configurer)${frange;1.2.3})",
  effective = "active")
@Retention(RetentionPolicy.CLASS)
public @interface RequireConfigurer {}
```

By itself the previous snippet is only a declaration, no actual requirement is added to the manifest. However, if the `@RequireConfigurer` annotation is used to annotate a type that is included in the bundle then the requirement is actually added to the manifest.

The annotations generate proper headers with any duplicates removed. All text fields of the annotations are also run through the macro preprocessor. One of the macros that was added specifically for this purpose was the `${frange;<version>}`, this macro created a version range in OSGi filter syntax. However, all bnd macros were available which then also provides access to the local package-info.

3 Problem Description

Entering manifest headers is error prone since these headers are complex and singletons. Because they are singletons, there is only one place where they can be entered. This is in contrast with the promoted component model. Components should be cheap and easy to rename or move between bundles. If a component is moved from one bundle to another bundle, it is easy to also move its corresponding headers from the manifest. This can cause orphaned headers or missing headers either in the old bundle or the new bundle. The old bundle can miss the header because multiple components were depending on that header but it was mistakingly removed.

The other problem is that these headers are notoriously hard to write, it often takes several trials to get all the parts of the requirements correct.

4 Requirements

- G0010 – It must be possible to provide the content of manifest headers through manifest annotations.
- G0020 – The following headers must be supported by dedicated annotations:
  - `Provide-Capability`
  - `Require-Capability`
- G0030 – It must be possible to enter a clause for a specific header through an annotation
- G0040 – It must be possible to create customized annotations for specific requirements. This can be done via a meta-annotation which can be used to define other custom annotations.

- G0060 – Generated headers must be valid OSGi or generate an error
- G0070 – The annotations must use enums and other Java constructs to enable type safety and IDE support
- G0080 – Duplicate clauses must be removed in the manifest
- G0100 – The solution must simplify the writing of a version import filter in a requirement.
- G0110 – DS and Metatype annotations must be retrofitted to use the meta-annotation from G0040.
- G0120 – The solution must include a package annotation for marking a package to be exported. The annotation must support specifying directives (e.g. mandatory, include, exclude) and attributes. The package version must come from the existing Version annotation.

5 Technical Solution

A new package, `org.osgi.annotation.bundle`, is defined containing new annotations for defining OSGi bundle metadata. These annotations are CLASS retention annotations to be processed by bundle assembly tools like Bnd.

5.1 General

When annotations support being applied to other annotations, so called meta-annotations, bundle assembly tools must look for the use of annotations which are meta annotated. This allows, for example, the DS Component annotation to be meta-annotated with a Requirement annotation for the osgi.component extender. So the use of the Component annotation by code in a bundle must then result in the bundle having a requirement on the osgi.component extender.

When annotations support Repeatable, bundle assembly tools must support the annotation container annotation holding the repeated annotation.

Bundle assembly tools must detect duplicate header information and remove duplicates. This is especially important given the support for meta-annotations. For example, there can be many DS components all annotated with Component which itself is meta-annotated with a Requirement for the osgi.component extender.

Since manifest headers only appear once in the manifest, bundle assembly tools must collect up all clauses for a specific manifest header separating the values with commas. For example, there can be multiple packages annotation with Export.

For OSGi specified manifest headers, the bundle assembly tool must check that the resulting manifest header is valid.

5.2 Export

An Export annotation is defined which can be applied to a package. The version of the package is taken from the Version annotation on the package. Other information regarding the package can be specified in the Export annotation including how the exported package should be substitutably imported.

5.3 Capability

A Capability annotation is defined to express a capability for a bundle. This annotation can be used as a meta-annotation and also applied to a package or class. This annotation supports Repeatable using the Capabilities container annotation.

A Capability annotation must specify the namespace of the capability. Other information regarding the capability can be specified including version, attributes and the uses and effective directive values.

5.4 Requirement

A Requirement annotation is defined to express a requirement for a bundle. This annotation can be used as a meta-annotation and also applied to a package or class. This annotation supports Repeatable using the Requirements container annotation.

A Requirement annotation must specify the namespace of the requirement. Other information regarding the requirement can be specified including filter and the effective, cardinality and resolution directive values.

5.5 Header

A Header annotation is defined to express a header for a bundle. This annotation can be used as a meta-annotation and also applied to a package or class. This annotation supports Repeatable using the Headers container annotation.

A Header annotation must specify the header name and value.

5.6 Directive and Attribute

Directive and Attribute annotations are defined which can be used when using Requirement and Capability as meta annotations. These annotations mark the annotated element as a directive or attribute of the generated requirement or capability.

For example:

```
@Requirement(namespace = "foo")
public @interface MyRequirement {
  @Attribute("foo")
  String value();
}

@MyRequirement("bar")
public class MyClass {}
```

This will result in the attribute `foo=bar` on the generated requirement.

6 Javadoc

OSGi Javadoc

1/12/16 12:34 PM

Package Summary		Page
org.osgi.annotation.bundle	OSGi Bundle Annotations Package Version 1.0.	11

Package org.osgi.annotation.bundle

`@org.osgi.annotation.versioning.Version(value="1.0")`

OSGi Bundle Annotations Package Version 1.0.

See:

Description

Enum Summary		Page
Export.Substitution	Substitution policy for this package.	19
Requirement.Cardinality	Cardinality for this requirement.	26
Requirement.Resolution	Resolution for this requirement.	28

Annotation Types Summary		Page
Attribute	Mark an annotation element as an attribute.	12
Capabilities	Container annotation for repeated `Capability` annotations.	13
Capability	Define a capability for a bundle.	14
Directive	Mark an annotation element as a directive.	16
Export	Mark a package to be exported from its bundle.	17
Header	Define a manifest header for a bundle.	21
Headers	Container annotation for repeated `Header` annotations.	22
Requirement	Define a requirement for a bundle.	23
Requirements	Container annotation for repeated `Requirement` annotations.	30

Package org.osgi.annotation.bundle Description

OSGi Bundle Annotations Package Version 1.0.

This package is not used at runtime.

Annotation Type Attribute

org.osgi.annotation.bundle

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.METHOD)
public @interface Attribute
```

Mark an annotation element as an attribute.

This is used when applying `Capability` or `Requirement` as a meta annotation to an annotation declaration. The value of the annotation element annotated with `Attribute` is used as the value of an attribute in the generated capability or requirement clause. For example:

```
@Capability(namespace = "my.namespace")
public @interface MyCapability {
  @Attribute("attr")
  String value();
}

@MyCapability("foo")
public MyClass {}
```

The use of the `MyCapability` annotation, which is meta annotated with the `Capability` and `Attribute` annotations, will result in a capability in the namespace `my.namespace` with the attribute `attr=foo`.

This annotation is not retained at runtime. It is for use by tools to generate bundle manifests.

Required Element Summary		Page
String	**value** The name of the attribute.	*12*

Element Detail

value

```
public abstract String value
```

The name of the attribute.

If not specified, the name of the annotated element is used as the name of the attribute.

Default:
""

Annotation Type Capabilities

org.osgi.annotation.bundle

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value={
  ElementType.TYPE,
  ElementType.PACKAGE
})
public @interface Capabilities
```

Container annotation for repeated `Capability` annotations.

Required Element Summary		*Page*
`Capability[]`	**`value`** Repeated `Capability` annotations.	*13*

Element Detail

value

```
public abstract Capability[] value
```

Repeated `Capability` annotations.

Annotation Type Capability

org.osgi.annotation.bundle

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value={
  ElementType.TYPE,
  ElementType.PACKAGE
})
@Repeatable(value=Capabilities.class)
public @interface Capability
```

Define a capability for a bundle.

For example:

```
 @Capability(namespace=ExtenderNamespace.EXTENDER_NAMESPACE,
             name="osgi.component", version="1.3.0")
```

This annotation is not retained at runtime. It is for use by tools to generate bundle manifests or otherwise process the type or package.

Required Element Summary		Page
String[]	**attribute** A list of attribute or directive names and values.	15
String	**effective** The effective time of this capability.	15
String	**name** The name of this capability within the namespace.	14
String	**namespace** The namespace of this capability.	14
Class<?>[]	**uses** A list of classes whose packages are inspected to calculate the `uses` directive for this capability.	15
String	**version** The version of this capability.	15

Element Detail

namespace

`public abstract String namespace`

The namespace of this capability.

name

`public abstract String name`

The name of this capability within the namespace.

If specified, adds an attribute with the name of the namespace and the value of the specified name to the capability clause.

Default:
""

---

version

`public abstract String` **`version`**

The version of this capability.

If specified, adds an attribute with the name and type of `version:Version` and the value of the specified version to the capability clause.

The specified version must be a valid OSGi version string.

Default:
""

---

uses

`public abstract Class<?>[]` **`uses`**

A list of classes whose packages are inspected to calculate the `uses` directive for this capability.

If not specified, the `uses` directive is omitted from the capability clause.

Default:
{}

---

effective

`public abstract String` **`effective`**

The effective time of this capability.

Specifies the time the capability is available. The OSGi framework resolver only considers capabilities without an effective directive or effective:=resolve. Capabilities with other values for the effective directive can be considered by an external agent.

If not specified, the `effective` directive is omitted from the capability clause.

Default:
"resolve"

---

attribute

`public abstract String[]` **`attribute`**

A list of attribute or directive names and values.

Each string should be specified in the form:

- `"name=value"` for attributes.
- `"name:type=value"` for typed attributes.
- `"name:=value"` for directives.

These are added, separated by semicolons, to the export package clause.

Default:
{}

Annotation Type Directive

org.osgi.annotation.bundle

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.METHOD)
public @interface Directive
```

Mark an annotation element as a directive.

This is used when applying `Capability` or `Requirement` as a meta annotation to an annotation declaration. The value of the annotation element annotated with `Directive` is used as the value of a directive in the generated capability or requirement clause. For example:

```
@Capability(namespace = "my.namespace")
public @interface MyCapability {
  @Directive("resource")
  String value();
}

@MyCapability("foo")
public MyClass {}
```

The use of the `MyCapability` annotation, which is meta annotated with the `Capability` and `Directive` annotations, will result in a capability in the namespace `my.namespace` with the directive `resource:=foo`.

This annotation is not retained at runtime. It is for use by tools to generate bundle manifests.

Required Element Summary		Pag e
`String`	**value** The name of the directive.	*16*

Element Detail

value

```
public abstract String value
```

The name of the directive.

If not specified, the name of the annotated element is used as the name of the directive.

Default:
""

Annotation Type Export

org.osgi.annotation.bundle

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.PACKAGE)
public @interface Export
```

Mark a package to be exported from its bundle.

The package must also be annotation with the `org.osgi.annotation.versioning.Version` annotation to specify the export version of the package.

This annotation is not retained at runtime. It is for use by tools to generate bundle manifests or otherwise process the package.

Nested Class Summary		Page
static enum	**Export.Substitution** Substitution policy for this package.	19

Required Element Summary		Page
String[]	**attribute** A list of attribute or directive names and values.	17
Export.Substitution	**substitution** Specify the policy for substitutably importing this package.	18
String[]	**uses** A list of package names that are used by this package.	17

Element Detail

uses

```
public abstract String[] uses
```

A list of package names that are used by this package.

If the `uses` directive must be omitted from the export package clause for this package, the empty value `{}` must be specified.

If not specified, the `uses` directive for the export package clause is calculated by inspection of the classes in this package.

Default:
{}

attribute

```
public abstract String[] attribute
```

A list of attribute or directive names and values.

Each string should be specified in the form:

! `"name=value"` for attributes.

! `"name:type=value"` for typed attributes.
! `"name:=value"` for directives.

These are added, separated by semicolons, to the export package clause.

Default:
{}

substitution

`public abstract Export.Substitution` **`substitution`**

Specify the policy for substitutably importing this package.

Bundles that collaborate require the same class loader for types used in the collaboration. If multiple bundles export packages with collaboration types then they will have to be placed in disjoint class spaces, making collaboration impossible. Collaboration is significantly improved when bundles are willing to import exported packages; these imports will allow a framework to substitute exports for imports.

If not specified, the `Export.Substitution.CALCULATED` substitution policy is used for this package.

Default:
Export.Substitution.CALCULATED

Enum Export.Substitution

org.osgi.annotation.bundle

```
java.lang.Object
  └java.lang.Enum<Export.Substitution>
      └org.osgi.annotation.bundle.Export.Substitution
```

All Implemented Interfaces:
Comparable<Export.Substitution>, Serializable

Enclosing class:
Export

```
public static enum Export.Substitution
extends Enum<Export.Substitution>
```

Substitution policy for this package.

Enum Constant Summary	Page
CALCULATED The policy value is calculated by inspection of the classes in the package.	20
CONSUMER Use a consumer type version range for the import package clause when substitutably importing a package.	19
NOIMPORT The package must not be substitutably imported.	20
PROVIDER Use a provider type version range for the import package clause when substitutably importing a package.	19

Method Summary		Page
static Export.Substitution	**valueOf**(String name)	20
static Export.Substitution[]	**values**()	20

Enum Constant Detail

CONSUMER

```
public static final Export.Substitution CONSUMER
```

Use a consumer type version range for the import package clause when substitutably importing a package.

See Also:
org.osgi.annotation.versioning.ConsumerType

PROVIDER

```
public static final Export.Substitution PROVIDER
```

Use a provider type version range for the import package clause when substitutably importing a package.

See Also:

`org.osgi.annotation.versioning.ProviderType`

NOIMPORT

```
public static final Export.Substitution NOIMPORT
```

The package must not be substitutably imported.

CALCULATED

```
public static final Export.Substitution CALCULATED
```

The policy value is calculated by inspection of the classes in the package.

Method Detail

values

```
public static Export.Substitution[] values()
```

---

valueOf

```
public static Export.Substitution valueOf(String name)
```

Annotation Type Header

org.osgi.annotation.bundle

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value={
  ElementType.TYPE,
  ElementType.PACKAGE
})
@Repeatable(value=Headers.class)
public @interface Header
```

Define a manifest header for a bundle.

For example:

```
 @Header(name=Constants.BUNDLE_CATEGORY, value="osgi")
```

This annotation is not retained at runtime. It is for use by tools to generate bundle manifests.

Required Element Summary		Page
String	**name** The name of this header.	*21*
String	**value** The value of this header.	*21*

Element Detail

name

```
public abstract String name
```

The name of this header.

value

```
public abstract String value
```

The value of this header.

Annotation Type Headers

org.osgi.annotation.bundle

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value={
  ElementType.TYPE,
  ElementType.PACKAGE
})
public @interface Headers
```

Container annotation for repeated `Header` annotations.

Required Element Summary		***Page***
`Header[]`	**`value`** Repeated `Header` annotations.	*22*

Element Detail

value

```
public abstract Header[] value
```

Repeated `Header` annotations.

Annotation Type Requirement

org.osgi.annotation.bundle

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value={
  ElementType.TYPE,
  ElementType.PACKAGE
})
@Repeatable(value=Requirements.class)
public @interface Requirement
```

Define a requirement for a bundle.

For example:

```
 @Requirement(namespace=ExtenderNamespace.EXTENDER_NAMESPACE,
                 name="osgi.component", version="1.3.0")
```

This annotation is not retained at runtime. It is for use by tools to generate bundle manifests or otherwise process the a package.

	Nested Class Summary	Page
static enum	**Requirement.Cardinality** Cardinality for this requirement.	26
static enum	**Requirement.Resolution** Resolution for this requirement.	28

	Required Element Summary	Page
String[]	**attribute** A list of attribute or directive names and values.	24
Requirement.Cardinality	**cardinality** The cardinality of this requirement.	25
String	**effective** The effective time of this requirement.	24
String	**filter** The filter expression of this requirement, if any.	24
String	**name** The name of this requirement within the namespace.	24
String	**namespace** The namespace of this requirement.	23
Requirement.Resolution	**resolution** The resolution policy of this requirement.	25
String	**version** The floor version of the version range for this requirement.	24

Element Detail

namespace

```
public abstract String namespace
```

The namespace of this requirement.

name

`public abstract String` **`name`**

The name of this requirement within the namespace.

If specified, adds an expression, using the `&` operator with any specified `filter()`, to the requirement's filter directive to test that an attribute with the name of the namespace is equal to the value of the specified name.

Default:
""

---

version

`public abstract String` **`version`**

The floor version of the version range for this requirement.

If specified, adds a version range expression, using the `&` operator with any specified `filter()`, to the requirement's filter directive. The ceiling version of the version range is the next major version from the floor version. For example, if the specified version is `1.3`, then the version range expression is `(&(version>=1.3)(!(version>=2.0)))`.

The specified version must be a valid OSGi version string.

Default:
""

---

filter

`public abstract String` **`filter`**

The filter expression of this requirement, if any.

Default:
""

---

effective

`public abstract String` **`effective`**

The effective time of this requirement.

Specifies the time the requirement is available. The OSGi framework resolver only considers requirement without an effective directive or effective:=resolve. Requirements with other values for the effective directive can be considered by an external agent.

If not specified, the `effective` directive is omitted from the requirement clause.

Default:
"resolve"

---

attribute

`public abstract String[]` **`attribute`**

A list of attribute or directive names and values.

Each string should be specified in the form:

- `"name=value"` for attributes.
- `"name:type=value"` for typed attributes.
- `"name:=value"` for directives.

These are added, separated by semicolons, to the export package clause.

Default:
{}

---

cardinality

`public abstract Requirement.Cardinality cardinality`

The cardinality of this requirement.

Indicates if this requirement can be wired a single time or multiple times.

If not specified, the `cardinality` directive is omitted from the requirement clause.

Default:
Requirement.Cardinality.SINGLE

---

resolution

`public abstract Requirement.Resolution resolution`

The resolution policy of this requirement.

A mandatory requirement forbids the bundle to resolve when this requirement is not satisfied; an optional requirement allows a bundle to resolve even if this requirement is not satisfied.

If not specified, the `resolution` directive is omitted from the requirement clause.

Default:
Requirement.Resolution.MANDATORY

Enum Requirement.Cardinality

org.osgi.annotation.bundle

```
java.lang.Object
  └java.lang.Enum<Requirement.Cardinality>
      └org.osgi.annotation.bundle.Requirement.Cardinality
```

All Implemented Interfaces:
Comparable<Requirement.Cardinality>, Serializable

Enclosing class:
Requirement

```
public static enum Requirement.Cardinality
extends Enum<Requirement.Cardinality>
```

Cardinality for this requirement.

Enum Constant Summary	Page
MULTIPLE Indicates if the requirement can be wired multiple times.	26
SINGLE Indicates if the requirement can only be wired a single time.	26

Method Summary		Page
static Requirement.Cardinality	**valueOf**(String name)	27
static Requirement.Cardinality[]	**values**()	26

Enum Constant Detail

SINGLE

```
public static final Requirement.Cardinality SINGLE
```

Indicates if the requirement can only be wired a single time.

MULTIPLE

```
public static final Requirement.Cardinality MULTIPLE
```

Indicates if the requirement can be wired multiple times.

Method Detail

values

```
public static Requirement.Cardinality[] values()
```

valueOf

```
public static Requirement.Cardinality valueOf(String name)
```

Enum Requirement.Resolution

org.osgi.annotation.bundle

```
java.lang.Object
  └java.lang.Enum<Requirement.Resolution>
      └org.osgi.annotation.bundle.Requirement.Resolution
```

All Implemented Interfaces:

Comparable<Requirement.Resolution>, Serializable

Enclosing class:

Requirement

```
public static enum Requirement.Resolution
extends Enum<Requirement.Resolution>
```

Resolution for this requirement.

Enum Constant Summary	Page
MANDATORY A mandatory requirement forbids the bundle to resolve when the requirement is not satisfied.	28
OPTIONAL An optional requirement allows a bundle to resolve even if the requirement is not satisfied.	28

Method Summary		Page
static Requirement.Resolution	**valueOf**(String name)	29
static Requirement.Resolution[]	**values**()	28

Enum Constant Detail

MANDATORY

```
public static final Requirement.Resolution MANDATORY
```

A mandatory requirement forbids the bundle to resolve when the requirement is not satisfied.

OPTIONAL

```
public static final Requirement.Resolution OPTIONAL
```

An optional requirement allows a bundle to resolve even if the requirement is not satisfied.

Method Detail

values

```
public static Requirement.Resolution[] values()
```

valueOf

```
public static Requirement.Resolution valueOf(String name)
```

Annotation Type Requirements

org.osgi.annotation.bundle

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value={
  ElementType.TYPE,
  ElementType.PACKAGE
})
public @interface Requirements
```

Container annotation for repeated `Requirement` annotations.

Required Element Summary		Page
`Requirement[]`	**`value`** Repeated `Requirement` annotations.	30

Element Detail

value

```
public abstract Requirement[] value
```

Repeated `Requirement` annotations.

7 Considered Alternatives

The following requirement is deferred for now.

- G0090 – It must be possible to refer to a package name, package version, context information, class name, bundle version and bundle symbolic name in the manifest annotations. This must be done by computing the desired information from a class constant per G0070.

8 Security Considerations

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

Since these are CLASS retention annotations, there are no runtime security considerations.

9 Document Support

9.1 References

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

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

9.2 Author's Address

Name	BJ Hargrave
Company	IBM

9.3 Acronyms and Abbreviations

9.4 End of Document