RFP-167 Manifest Annotations

Draft

8 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 Table of Contents

0.2 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.`

0.3 Revision History

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

Revision	Date	Comments
Initial	*27-11-14*	*Initial* *Peter.Kriens@aQute.biz*

1 Introduction

This RFP originates from the OSGi enRoute work. In this project, a number of services were identified, designed and implemented based on their needs for web based applications. This document analyzes the application domain and defines the problem that needs to be solved.

2 Application Domain

2.1.1 bnd

bnd(tools) 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.

Implementing these annotations made it clear there were more headers in the manifest that could benefit from annotations. Therefore manifest annotations were added for the following headers:

- `@BundleLicense(name,description,link)`
- `@BundleCopyright(value)`
- `@BundleCategory (Category[] value, String[] custom)`
- `@BundleContributors(value (email), name, roles, organization, organizationUrl, timezone)`
- `@BundleDevelopers(value (email), name, roles, organization, organizationUrl, timezone)`
- `@BundleDocUrl(value (url))`

Since there are a limited number of popular open source licenses, enRoute also added annotations for each of those with a proper Bundle-License header applied to it.

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.

2.2 Terminology + Abbreviations

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. Also headers like Bundle-License are non trivial, especially if multiple licenses need to be entered. It is often hard to get the names right, especially if headers are not used very often.

This RFP therefore seeks a solution to simplify the manifest header generation for the common OSGi headers.

4 Use Cases

4.1.1 Requiring an Extender

A DS component to create an alarm if the network interfaces change is build by Al Bundle. Al uses the R7 version of the DS extender. In R7, the enRoute/bnd annotations were used to annotate the Component annotation.

```
@RequireCapability(
  ns        = "osgi.extender",
  filter    = "(&(osgi.extender=osgi.component)
                 (version:Version>=1.3)(!(version:Version>=2.0)))"
)
@Retention(RetentionPolicy.CLASS)
@Target(ElementType.TYPE)
public @interface Component { … }
```

Since the Component annotation now creates the proper requirement, al Al has to do is annotate his component.

```
@Component
public class Peggy { … }
```

Using maven with the latest bundle plugin, Al generates his bundle. When looking in the manifest he sees:

```
Require-Capability:
     osgi.extender;osgi.extender=osgi.component;filter:='(&(...))',
     …
```

4.1.2 Governance

Al Bundle works in Apache. It is common in Apache to provide the Bundle-License header in each bundle. Al therefore always adds the @ASL_2_0 to its packages:

```
@ASL_2_0
@Component
public class Peggy { … }
```

This now automatically generates the Bundle-License header.

5 Requirements

5.1.1 General

- G0010 – It must be possible to provide the content of manifest headers through manifest annotations.
- G0020 – The following headers should be supported by dedicated annotations:
  - `Provide-Capability`
  - `Require-Capability`
  - `Bundle-License`
  - `Bundle-Copyright`
  - `Bundle-Category`
  - `Bundle-DocUrl`
  - `Bundle-Vendor`
- 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.
- G0050 – The following customized annotations must be provided:

- Popular licenses like MIT, Apache, etc.
- All current Bundle Categories

- 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
- G0090 – It must be possible to refer to the current package in strings of the manifest annotations
- G0100 – The solution must simplify the writing of a version import filter in a requirement.
- G0110 – Existing specifications must be adapted to use these manifest annotations and define custom annotations when possible, among which
  - The DS Component annotation must create a requirement to the extender
  - Customized requirement annotations for all defined extenders

6 Document Support

6.1 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

6.2 Author's Address

Name	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drezery
Voice	+33 467542167
e-mail	Peter.kriens@aQute.biz

6.3 End of Document