Dr. Strangeversion, or, How I learned to stop worrying and love Qualifiers

Neil Bartlett – Paremus Ltd
Everything you always wanted to know about versions (*but were afraid to ask)
OMG I’m doing a talk on versions(!) and desperately need to sex up the title
What are Versions FOR Anyway??
Uniquely identifying releases

- Great, just use a SHA.
- Done!
- Next speaker please...
Wait, which is “Latest”?

- 21aa486ed420da46ec2c25ea3ac674eb602367ce
- 07e3d4c74bd61e231710461a8b911e4073de8594
- 8090b637657d09e1dacea12219ad5fb6d1353f96
- 62529587489ba2fd8a190d2c383738f8291ca9b0
- 92825c17eae8c35341ad0dc05478b7189fd95f4f
- 00cfb1b293fe48ac166171fdad9ee7e804d1894f
- dc415b82d06d3165e00ec552657b0e387aa368de
- f61302d1cb5c1593d65e005104ffc001fddbae7d
- ...

Use Integers!

- 1
- 2
- 3
- 4

- Man that was easy. Next?
Compatibility

• Can I run code compiled for version N on version N+x?
• Naïve answer: yes for all N, x.
• Example: Google Guava
Until Things Change

- Hmm we need to make a breaking change.
- How do we signal this?
Version Segments

- 1.0.0
- 2.10.15-SNAPSHOT
- java full version "1.6.0_26-b03-383"
What Does it MEAN?

• Versions are COMMUNICATION
• A promise to other developers about compatibility and future changes.
Scope of Changes?

1.6.0_26-b03-383

1.7.0_05-b06

1.7.0_06-b24
OSGi Versions
Legal Versions

0.0.0
1.0.0
2.0.9
3.4.12
4.0.4.beta3
5.1.2.RELEASE
Omitting segments

1.1 == 1.1.0
1 == 1.0 == 1.0.0
<empty> == 0 == 0.0 == 0.0.0
Qualifiers

1.0.0 == 1.0.0.<empty>

1.1.0.beta > 1.1.0.alpha
1.1.0.beta3 > 1.1.0.beta2
GOTCHAS

1.1.0.beta > 1.1.0
1.1.0.beta2 > 1.1.0.beta10
Semantic Versions
Which Segment?

- I made a change to 1.2.14
- Is my new version 1.2.15?
- Or 1.3.0?
- Or 2.0.0?
Importing

• I compiled against 1.2.14
• Can I use 1.2.15?
• Can I use 1.3.0?
• How about 2.0.0?
Semantics

- Need consistent rules for which segment to bump
- Based on degree of change in the artifact
Bigger than Java

- Tom Preston-Warner (GitHub co-founder)
- http://semver.org/
OSGi Semantics

- R5 Core Spec section 3.7.3
• Major: breaking change

• Minor: new feature

• Micro: non-visible change

• Qualifier: no semantic meaning
Is This Breaking?

public interface Foo {
    void foo();
}

downarrow

gives

public interface Foo {
    void bar();
}
Is This Breaking?

```java
public interface Foo {
    void foo();
}

public interface Foo {
    void foo(int x);
}
```
public interface Foo {
    void foo();
}

public interface Foo {
    /** Fooify all the Bars */
    * @author Bob */
    void foo();
}
Is This Breaking?

```java
public interface Foo {
    void foo();
}
```

```java
public interface Foo {
    void foo();
    void bar();
}
```
Consumer vs Provider

- Consumers can ignore new features
- Providers CANNOT
Consumer Range

- Export: 1.0.0
- Import: [1.0.0, 2.0.0)
Provider Range

- Export: 1.0.0
- Import: [1.0.0, 1.1.0)
public interface Callback {
    void thingChanged(Thing t);
}
public interface Callback {
    void thingChanged(Thing t);
    void otherThingChanged();
}

Bang!
Putting it into Practice
Best Practices: USELESS

• “Always version your exported packages”

• Great. I have 1000s of packages!

• (Maybe you *should* have far fewer exported packages)
Possible Solution

• “Release Train”: synchronise *everything* into one massive release once a year

• It’s The Eclipse Way™

• Also the Google way & many others
Use The Tools, Luke

• Don’t make developers reason about import ranges.

• Don’t make developers reason about export versions.

• WE WILL GET IT WRONG

• We’re only human!
bnd

- Import-Package used to be such a hassle
- bnd made that problem go away!
- It can make the version problem go away too.
Feature Summary

• Import range generated
• Consumer/provider relation detected
• Export version calculated on release
• Wrong version => broken build
Help the Tool
to Help You
• Version analysis of executable code is impossible... in practice and in theory

• Another benefit of service-oriented design

• Focus on the interface

• Minimise exports
• Consumers should NOT track latest API

• Use the LOWEST version they can still be compatible with

• Providers should track: more tightly coupled.
Qualifiers
• There will be many versions 1.0.0 before it is “released”

• Qualifier can be used to signal “phase”

• E.g. ALPHA, BETA, RC1, SNAPSHOT, RELEASE
Problem

1.0.0.RC5

↓

1.0.0.RELEASE
• RC5 is thoroughly tested by QA, it passes.
• Replace RC5 with RELEASE and ship it!
• But this change can affect resolution, and other changes can creep in.

• What You Test Is What You Release.
Smart Repositories

- Developer Repo
  - Freely replace 1.0.0 with new 1.0.0

- Release Repo
  - Attempting to replace 1.0.0 with another 1.0.0 is an error that breaks the build

- Multi-level Repos: dev, group, division, organisation... global?
Conclusion
• Versions are communication

• They describe a fundamental aspect of your code

• They can be derived, directly and automatically, from your code
Marketing Department: Keep Your Filthy Hands Off My Versions!!!
Tip

- Give the marketing dept their own version:

  Bundle-ManifestVersion: 2.0
  Bundle-SymbolicName: org.example
  Bundle-Version: 2.1.14.20121024_1400
  Bundle-Activator: org.example...
  Initech-Version: 12.0 Super Enhanced
  XP Ultimate Edition (Manatee)
  Export-Package: org.example;version=1
Finally: Like Tools?

- 20:00 Tonight: OSGi Tooling BoF, Seminarraume 1-3
- Friday all day: OSGi Tooling Workshop, organised by User Forum Germany
- [http://germany.osgiusers.org/Main/OSGiToolingWorkshop](http://germany.osgiusers.org/Main/OSGiToolingWorkshop)