Using Guava in your build

How to configure your build

For any code snippet below, please substitute the version given with the version of Guava you wish to use.

Maven

Add the following snippet to the `<dependencies />` section:

```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>23.5-jre</version> <!-- or 23.5-android for the Android flavor -->
</dependency>
```

Gradle

Ensure the maven standard repositories are available like so:

```
repositories {
  mavenCentral()
}
```

Then add the Guava dependency in the dependencies section like so:

```
dependencies {
  compile group: 'com.google.guava', name: 'guava', version: '23.5-jre' # or 23.5-android fo
}
```

Ivy

Add the following line to the dependencies section:

```
<dependency org="com.google.guava" name="guava" rev="23.5-jre" /> <!-- or rev="23.5-android
```

and make sure that the “public” resolver is used.

Buildr

Include this:

```
compile.with 'com.google.guava:guava:jar:23.5-jre' # or '...:23.5-android' for the Android f
```

Manual Dependencies

You can also just manually download JARs for the classes, sources and javadocs. See the appropriate release.

What about GWT?

In all the examples above, leave "com.google.guava" as it is but replace the other occurrence of "guava" with "guava-gwt". (There is no GWT artifact for the Android flavor.)

What about Guava's own dependencies?

Guava has one dependency that is needed for linkage at runtime: `com.google.guava:failureaccess:1.0.1`. We created it as part of making `ListenableFuture` available as a separate artifact from the rest of Guava.

Guava also depends on some artifacts that contain only annotations. We use those annotations primarily to help certain compilers and static analyzers, so the annotations usually need not be present on your runtime classpath unless you want your app to access them through Java's reflection APIs. (However, note an app that omits the dependency may fail during reflective operations, even if it is not looking specifically for *those* annotations. For example, see the discussion of `NoClassDefFoundError` in JDK-8247797.) In many cases, the annotations need not be present even on your *compile-time* classpath. Yet we make them available by default on both your compile-time classpath and your runtime classpath in order to avoid errors in unusual cases. (For examples, see #2652 / #2837 / auto#1032 (annotation processors – which can also fail silently), #2721 / #2824 (`-Xlint:all -Werror`), and #1095 / #1127 (Scala).) If you wish to avoid the dependencies, you can usually safely exclude them.

(Users sometimes request that we make the annotations available at compile-time but not runtime. This is not currently possible in Maven's model, but it recently became possible in Gradle's model. We are considering doing this for at least the j2objc-annotations artifact (whose annotations don't have runtime retention, anyway), but we will probably not do so for the others, given the uncertain dangers of omitting the annotations even only at runtime.)

How do I avoid conflicts between multiple versions of Guava?

In the simple case, just upgrade your project to the newest version of Guava. Or, if you don't use Guava directly, use dependency management to select the newest version of Guava for all your dependencies. However, this can fail in some cases:

- One of your dependencies uses `@Beta` APIs against our warnings.
- One of your dependencies uses a version of Guava from the days when we were still removing non-`@Beta` APIs.

In these cases, you may be able to find an older version of Guava that satisfies all your dependencies. However, the only general solution is for all projects to update to at least Guava 21 and for those used as dependencies to avoid `@Beta` APIs.

What if I want to use `@Beta` APIs from a library that people use as a dependency?

First, please let us know. We may be able to remove `@Beta` from the APIs you need. Even if we can't do it immediately, we'd like to know about your need so that we can prioritize better.

Second, if you don't need to expose *Guava types* to users in your API (e.g., expose a `public` method that returns `ImmutableList`), you can configure your build to *shade/relocate* Guava: You write your code against Guava as usual, and your build creates a private copy of Guava, rewrites your compiled code to use it, and includes it in your artifact. All you need is some Maven configuration like that shown below. Just substitute in a package of your own for `my.organization` in the `shadedPattern`:

```
<plugin>
  <artifactId>maven-shade-plugin</artifactId>
  <version>3.1.0</version>
  <configuration>
    <artifactSet>
      <includes>
        <include>com.google.guava:*</include>
      </includes>
    </artifactSet>
    <relocations>
      <relocation>
        <pattern>com.google</pattern>
        <shadedPattern>my.organization.repackaged.com.google</shadedPattern>
      </relocation>
    </relocations>
  </configuration>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Gradle users can do the same with the Shadow Plugin.