Java Tips

How do I set up the configuration of SLF4J?

You need to add a *log4j-test.properties* under the directory of java test and then add the following snippets into your *build.gradle* file.

```
test {
  systemProperty "log4j.configuration", "log4j-test.properties"
}

dependencies {
  shadow library.java.slf4j_api
  shadow library.java.slf4j_log4j12
  // or shadow library.java.slf4j_jdk14
}
```

Please note that the second dependency **shadow library.java.slf4j_log4j12** is not necessary if this dependency is already provided by other library. You can check the dependency tree by launching **./gradlew dependencies** to see if it has been included.

If you encounter the error message like this:

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation

It means that no SLF4J binding is found. So you need to add **library.java.slf4j_log4j12** or **library.java.slf4j_jdk14** in *build.gradle* file**.**

How do I automatically format code and avoid spotless errors?

You can set up a git precommit hook to always autoformat code, by putting this in `.git/hooks/pre-commit` and setting the executable bit.

```
#!/bin/sh
set -e
./gradlew spotlessApply
```

If you haven't used git hooks, the docs are here: https://git-scm.com/docs/githooks.

Using `--no-verify` will skip it and `` `chmod u-x` `` will disable it.

How do I run a single test?

```
./gradlew :examples:java:test --tests org.apache.beam.examples.subprocess.ExampleEchoPipelineTest --info
```

Running Java Dataflow Hello World pipeline with compiled Dataflow Java worker.

You can actually dump multiple definitions for gcp project name and temp folder. They are present, since different targets use different names.

Before running command, remember to configure your gcloud credentials. Add GOOGLE_APPLICATION_CREDENTIALS to your env variables.

./gradlew :runners:google-cloud-dataflow-java:examples:preCommitLegacyWorker -PdataflowProject=<GcpProjectName> -Pproject=<GcpProjectName> -PgcpProject=<GcpProjectName> -PgcsTempRoot=<Gcs location in format: gs://..., no trailing slash> -PdataflowTempRoot=<Gcs location in format: gs://...>

./gradlew :runners:google-cloud-dataflow-java:examples:preCommitFnApiWorker -PdataflowProject=<GcpProjectName> -Pproject=<GcpProjectName> -PgcpProject=<GcpProjectName> -PgcsTempRoot=<Gcs location in format: gs://..., no trailing slash> -PdataflowTempRoot=<Gcs location in format: gs://..., no trailing slash> -PdockerImageRoot=<docker image store location in format gcr.io/...>

Running a User Defined Pipeline (example is on Java Direct Runner).

If you want to run your own pipeline, and in the meanwhile change beam repo code for dev/testing purpose. Here is an example.

If it is just a simple runner like directRunner, all you need to do is to put your pipeline code under example folder, and then add following build target to the related build.gradle:

```
task execute(type:JavaExec) {
main = "org.apache.beam.examples.SideInputWordCount"
classpath = configurations."directRunnerPreCommit"
}
```

There are also alternative choices, with slight difference:

1) Create a maven project. And use the following command to publish changed code to local repository.

```
./gradlew -Ppublishing -PnoSigning publishMavenJavaPublicationToMavenLocal
```

2) Make use of Integration tests, and make your user defined pipeline part of the integration test.

Continue on error

--continue – this flag makes compileJava task to dump all found errors, not stop on first.

Intellij Proto Intellisense doesn't work.

This can happen when you start intellij, or (in my case) after modifying protos.

This is not a solved problem yet.

Currently tried approaches:

1. Clean build from console
2. Build from IntelliJ
3. Refresh Gradle Project in IntelliJ
4. Restart IntelliJ
5. Another option looked at that should help if index is not updated via 3 and 4: https://stackoverflow.com/questions/6652540/rebuild-intellij-project-indexes

Workaround that did the trick. Since many things were tried in process and no clear way to reproduce error, this might not be the correct/best steps. Update steps if you find shorter/cleaner way to do the trick.

1. Refresh tradle project in IntelliJ
2. Close intellij
3. clean build project from console (./gradlew clean cleanTest build -x testWebsite -x :rat -x test)
4. Open IntelliJ

What command should I run locally before creating a pull request?

We recommend running this command, in order to catch common style issues, potential bugs (using code analysis), and javadoc issues before creating a pull request. Running this takes 5-10 minutes.

```
./gradlew spotlessApply && ./gradlew checkstyleMain checkstyleTest javadoc spotbugsMain compileJava
compileTestJava
```

If you don't run this locally, they will be ran during presubmit, by Jenkins. However, if these fail during presubmit, you may not see the output of test failures. So doing this first is recommended to make your development process a bit smoother and iterate on your PR until it passes the presubmit.