Building

Building Camel from Source

Camel uses Maven as its build and management tool. If you don't fancy using Maven you can use your IDE directly or Download a distribution or JAR.

Prequisites

Required:

- Download and install Maven
  (Maven 3.1.1 or newer is required to build Camel 2.14 onwards)
  (Maven 3.2.5 or newer is required to build Camel 2.18 onwards)
  (Maven 3.3.3 or newer is required to build Camel 2.20 onwards)
- Get the latest Source
- Java
  (1.7 or 1.8 to build Camel 2.14 onwards)
  (1.8 to build Camel 2.18 onwards)
  (1.9 experimental support for building Camel 2.19 onwards)

Maven options

To build camel maven has to be configured to use more memory:

Unix/Linux/Mac

```
export MAVEN_OPTS="-Xms3000m -Xmx3000m -XX:MaxPermSize=512m"
```

Windows

```
set MAVEN_OPTS=-Xms3000m -Xmx3000m -XX:MaxPermSize=512m
```

To conserve memory with Java 8 you can set two additional Java options -XX:+UseG1GC to enable G1 garbage collector (default in Java 9) and -XX:+UseStringDeduplication to help decrease Maven memory usage (up to 1.5GB is required currently, but set it a higher):

Unix/Linux/Mac

```
export MAVEN_OPTS="-Xmx2G -XX:+UseG1GC -XX:+UseStringDeduplication"
```

Windows

```
set MAVEN_OPTS=-Xmx2G -XX:+UseG1GC -XX:+UseStringDeduplication
```

You can try to experiment with parallel builds by adding -T1.5C to MAVEN_OPTS, but be wary as some of the plugins are not thread safe, and the console output will be intertwined. With parallel builds and when using the install goal you might experience race conditions with the local repository, using Takari Concurrent Local Repository will help with that.

A normal build

```
mvn clean install
```

A normal build without running tests

```
mvn clean install -Pfastinstall
```

A normal build without running tests but checkstyle verification enabled

```
mvn clean install -Pfastinstall,sourcecheck
```

Doing a Quick Build

Available as of Camel 2.6

The following skips building the manual, the distro and does not execute the unit tests.

```
mvn install -Pfastinstall
```

Updating the license headers

Proper license headers are enforced using Apache RAT and Checkstyle Maven plugins. To make it less tedious and error prone you can update the license headers by using:

```
mvn -Plicense license:format
```

This can be invoked from any module, which makes it useful when working on components. You can find the various license headers that the Camel project uses in buildtools/src/main/resources/header-*.txt files. These are regenerated at build time from header.txt in the same directory.

Building source jars

If you want to build jar files with the source code, that for instance Eclipse can important so you can debug the Camel code as well. Then you can run this command from the camel root folder:

```
mvn clean source:jar install -Pfastinstall
```

Working with karaf features

If you change anything in the features.xml from `platform/karaf` you can run a validation step to ensure the generated features.xml file is correct. You can do this running the following maven goal from the `platform` directory.

```
mvn clean install -Pvalidate
```

Executing unit tests using Ekstazi

Normally, when you execute the unit tests during your development cycle for a particular component, you are executing all the tests each time. This may become inefficient, when you are changing one class and the effect of this change is limited within the component having many unit tests. Ekstazi is a regression testing tool that can keep track of the test results and the changed classes so that unaffected tests can be skipped during the subsequent testing. For more details of Ekstazi, please refer to the Ekstazi page at http://www.ekstazi.org.

To use Ekstazi, you can run the tests with the maven profile ekstazi.

```
mvn test -Pekstazi
```

See Also

- Running Unit Tests
- Source
- Examples