Plastic: Getting Started

Getting Started

To begin, ensure that you have the prerequisite software installed as enumerated below. You will need internet access to pull down the project dependencies from a public Maven repository.

Use git to clone the repository

Prerequisites

You will need an internet connection (at least for the first build) for the dependencies to be downloaded into your local maven repository. To build this repository, you will need the following installed on your machine:

- git 2.14+
- Java 8 JDK
- Maven 3.0+

Note that Plastic has not been ported yet past Java 8

Building From Code

Pulling The Code

For read-only access, you can do this

```
git clone https://git.opendaylight.org/gerrit/plastic
```

For a committer, you can do this

```
git clone ssh://{username}@git.opendaylight.org:29418/plastic.git
```

Building

Once you have the prerequisites and have cloned the repo, you can issue a build at the top level of your local copy of the repo

```
cd plastic
mvn clean install
```

The build should complete normally. You can look in the target directory for artifacts. There should be a plastic-*.jar and a directory called runner. If you change your current working directory to the runner directory, you can issue the following command to see things work (this uses examples from the tutorial)

```
./plastic_runner runnerroot.properties
```

You should see log output that shows a successful translation from "abcd" to "ABCD".

If you have errors building...

You might hit an error about "Failure to find com.beust:klaxon:jar". This jar comes from Spring IO at https://repo.spring.io/plugins-release/com/beust/klaxon/ and this repo is not normally mirrored in most repositories, but ODL does mirror this and you can add the following to your ~/settings.xml

```
<repository>
    <id>opendaylight-public</id>
    <name>opendaylight-public</name>
    <url>https://nexus.opendaylight.org/content/repositories/public</url>
    <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
        <enabled>false</enabled>
    </snapshots>
</repository>
```

Running the tests

Most of the testing is done using unit tests that are written using Spock (a highly recommended
alternative to JUnit). These tests are run as part of every single build and a failure of
a unit test breaks the build.

Using a Plastic distribution

Plastic is 100% independent of ODL code/dependencies and can be used stand-alone. Plastic is available several places including Maven Central and you can insert the dependency in your POM like the below example.

```
<dependency>
    <groupId>org.opendaylight.plastic</groupId>
    <artifactId>odl-plastic</artifactId>
    <version>2.1.7</version>
</dependency>
```

Once you have the depedency, create an arbitrarily named folder, lets call it my-plastic, with the following required structure. Note that the directories can be empty but they are required.

```
my-plastic/
    lib/
    morphers/
    schemas/
    classifiers/
```

To use the Plastic logic, you can create a new SearchPath("/opt/myapp/my-plastic") instance, passing the file system location to that root directory, and pass that to a new instance of CartorapherWorker. This worker's lifetime should be that of your application. Just call worker.translate(...) for each translation.

```
class MyApp {

    // throws if this path does not have the required sub-directory structure above
    SearchPath root = new SearchPath("/opt/myapp/my-plastic");

    CartographerWorker worker = new CartographerWorker(root);

    // the schemas directory should have these somewhere:
    //    my-input-schema-1.0.json
    //    my-output-schema-1.0.json

    VersionedSchema inschema = new VersionedSchema("my-input-schema", "1.0", "json")
    VersionedSchema outschema = new VersionedSchema("my-output-schema", "1.0", "json")

    void handleIncomingPayload(String payload) {
        ...
        String result = worker.translate(inschema, outschema, payload);
        ...
    }
}
```

Out-of-the-box tutorial examples

There is a set of tutorials in the target/runner directory. You can find them as *.RST files. You can install rst2pdf and convert them to PDF if you'd like.

From the target/runner directory, you can execute any of the tutorial examples using a command like

```
./plastic_runner <name>.properties
```