# Pepper

# Developer's Guide for PepperModules

**Florian Zipser <`saltnpepper@lists.hu-berlin.de`>**
**INRIA**
**SFB 632 Information Structure / D1 Linguistic Database**
**Humboldt-Universität zu Berlin**
**Universität Potsdam**

# Pepper: Developer's Guide for PepperModules

by Florian Zipser, , , , and

Version ${project.version}

# Table of Contents

# List of Figures

# List of Tables

# Foreword

The aim of this document is to provide a helpful guide to create own PepperModules to plug them into the pepper framework.

We are trying to make things as easy to use as possible, but we are a non profit project and we need your help. So please tell us if things are to difficult and help us improving the framework.

You are even very welcome to help us improving this documentation by reporting bugs, requests for more information or by writing sections. Please write an email to `<saltnpepper@lists.hu-berlin.de>`.

Have fun developing with SaltNPepper!

# Chapter 1. Introduction

With SaltNPepper we provide two powerful frameworks for dealing with linguistic annotated data. SaltNPepper is an Open Source project developed at the Humboldt University of Berlin (see: http://www.hu-berlin.de/) and INRIA (Institut national de recherche en informatique et automatique, see: http://www.inria.fr/) as well. In linguistic research a variety of formats exists, but no unified way of processing them. To fill that gap, we developed a meta model called Salt which abstracts over linguistic data. Based on this model, we also developed the plugable universal converter framework Pepper to convert linguistic data between various formats.

Pepper is a container controlling the workflow of a conversion process, the conversion itself is done by a set of modules called PepperModules mapping the linguistic data between a given format and Salt and vice versa. Pepper is a highly plugable framework which offers the possibility to plug in new modules in order to incorporate further formats. The architecture of Pepper is flexible and makes it possible to benefit from all already existing modules. This means that when adding a new or previously unknown format Z to Pepper, it is automatically possible to map data between Z and all already supported formats A,B, C, … . A Pepper workflow consists of three phases:

1. the import phase (mapping data from a given fromat to Salt),

2. the optional manipulation phase (manipulating or enhancing data in Salt) and the

3. export phase (mapping data from Salt to a given format).

The three phase process makes it feasible to influence and manipulate data during conversion, for example by adding additional information or linguistic annotations, or by merging data from different sources.

Since Pepper is a plugable framework, we used an underlying framework called OSGi (see: http://www.osgi.org/) providing such a functiuonality. OSGi is a mighty framework and has a lot of impact in the way of programming things in java. Because we do not want to force you to learn OSGi, when you just want to create a new module for pepper, we tried to hide the OSGi layer as good as possible. Therefore and for the lifecycle management of such projects, we used another framework named maven (see: http://maven.apache.org/). Maven is configured via an xml file called pom.xml, you will find it in all SaltNPepper projects and also in the root of the SampleModule project. Maven makes things easier for use especially in dealing with dependencies.

The SaltNPepper framework also comes with an environment for a direct start up in eclipse. for an easier developement of a PepperModule. It further contains a test bed for checking simple consistency issues of PepperModules. This test bed is based on JUnit (see: junit.org) and can easily be extended for a specific module test.

In the following, Chapter 2, *Setting up environment* explains how to set up your environment to start developing your pepper module. Chapter 3, *Adopting the SampleModules* explains how to download and adopt a template module to your own needs. This module than is the base for your own module. In Chapter 4, *Customizing behaviour of your PepperModule* we explain how to add properties to your module, so that the user can dynamically customize the behaviour of the mapping. Chapter 5, *Documenting your PepperModule* shows the usage of a documentation template written in doocbook (see http://www.docbook.org/) to make a standard documentation for the user of your module. Since testing of software often is a pain in the back, the pepper-testSuite already comes with some predefined tests. These should save you some time, not to test even the simplest things. Last but not least, this documentation contains a FAQ. SInce a FAQ lives of questions, if you have some, just help us increasing that part and send a mail to `<saltnpepper@lists.hu-berlin.de>`.

# Chapter 2. Setting up environment

If you do not belong to the hardcore 'vi' developer community, you may want to use a developement environment and an IDE for developing your pepper module. For that case we here describe how to set up your environment along the eclipse IDE (see: http://download.eclipse.org). May you rather want to use a different IDE like NetBeans (see: www.netbeans.org/). In that case, skip the eclipse specific parts, but make sure, setting up the OSGi framework in your environment simultaneously.

## Download and set up eclipse

eclipse is available in several flavours like for web developers, mobil developers etc.. We recommend the eclipse Modeling Tools, since you might want to create or use a model for the format you want to support. For this tutorial, we used eclipse juno, version 4.2 (see: http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/junosr2). Unfortunatly, in this version eclipse does not come with an integrated OSGi console anymore. It now uses the Apache Gogo Shell which is not part of the eclipse distribution. Therefore we have to download the Equinox SDK from http://download.eclipse.org/equinox/, we used version 3.8.2 which is available on http://www.eclipse.org/downloads/download.php?file=/equinox/drops/R-3.8.2-201302041200/equinox-SDK-3.8.2.zip.

- Download the eclipse IDE and unzip it to ECLIPSE_HOME (a folder of your choice).
- Download the equinox sdk and unzip it to EQUINOX_SDK (a folder of your choice).
- Copy all (not already contained) bundles (.jar) from EQUINOX_SDK/plugins folder to ECLIPSE_HOME/plugins folder.

## Making eclipse ready for SaltNPepper

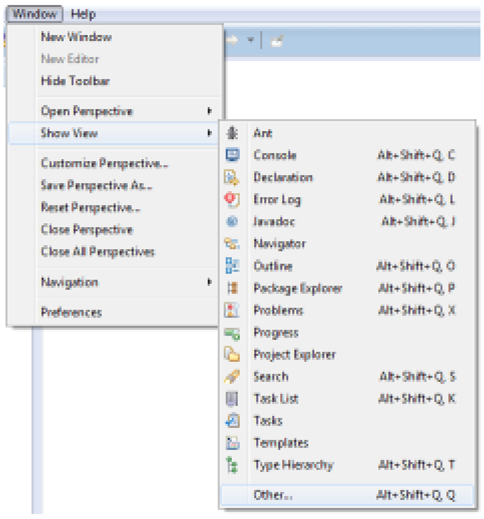
- Download the latest Pepper distribution from http://korpling.german.hu-berlin.de/saltnpepper/repository/saltNpepper_full/
- Unzip the Pepper distribution to PEPPER_HOME.
- Copy all bundles (.jar files) from PEPPER_HOME/plugins to ECLIPSE_HOME/dropins.
- Copy all folders having the same name as the bundles (.jar files) from PEPPER_HOME/plugins to a folder of your choice, lets say BUNDLE_RESOURCES/plugins.

## Starting eclipse

Now lets check if eclipse was set up correctly and is able to find all necessary bundles.

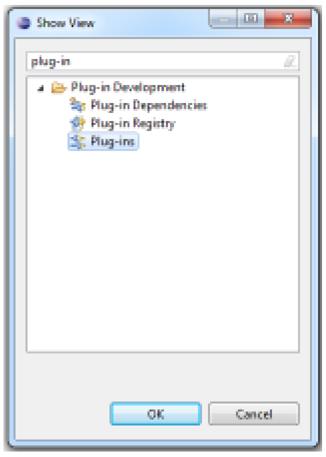
- Start eclipse.
- Open 'Plug-ins view' via 'Window --> Show View --> Others (see Figure 2.1, “”).

## Figure 2.1.



- Type in 'plug-ins' and choose Plug-ins view (see Figure 2.2, “”).

## Figure 2.2.



- In that view, you will find all OSGi bundles, which are accessable from your eclipse installation.

- Check if all bundles from SaltNPepper are there. The most important ones are
  - de.hu_berlin.german.korpling.saltnpepper.pepper-exceptions.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-framework.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-logReader.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-modules.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-moduleTests.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-testEnvironment.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-workflow.jar
  - de.hu_berlin.german.korpling.saltnpepper.salt-graph.jar
  - de.hu_berlin.german.korpling.saltnpepper.salt-saltCommon.jar

- de.hu_berlin.german.korpling.saltnpepper.salt-saltCore.jar

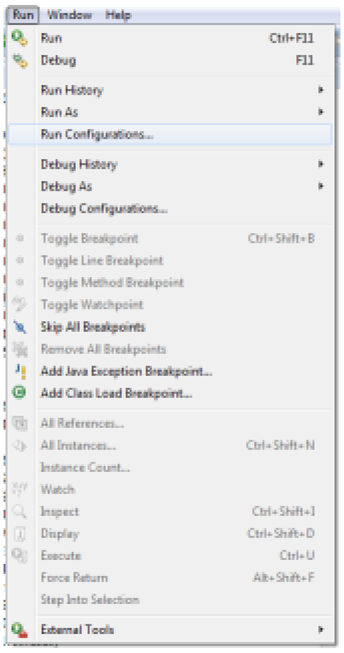
Congratulations, now you are done setting up your IDE.

## Let's run

Now we want to let the ghost out of the bottle and run the pepper framework.

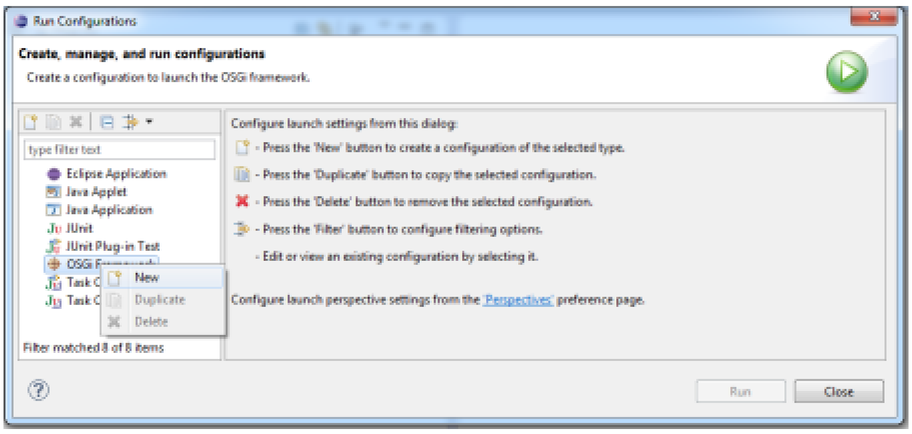
- Select 'Run --> Run configurations...' (see Figure 2.3, “”).

**Figure 2.3.**



- Select 'OSGi Framework', press right mouse button and select 'New' (see Figure 2.4, “”).

**Figure 2.4.**



Create, manage, and run configurations

Create a configuration to launch the OSGi framework.

Configure launch settings from this dialog:

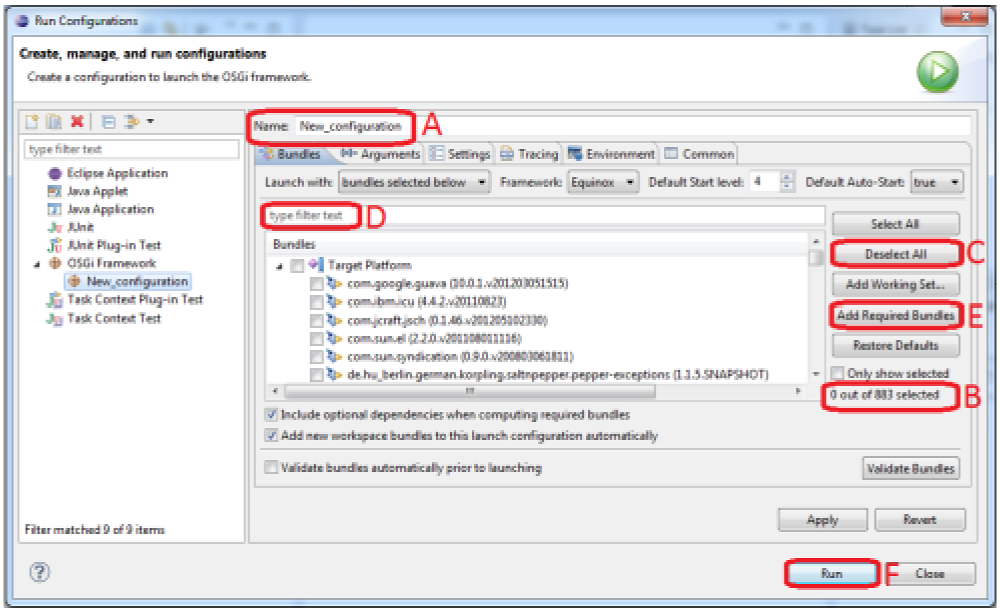
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Filter matched 8 of 8 items

- Give your run configuration a name under 'Name:', (see Figure 2.5, “” position A).

### Figure 2.5.



Name: New_configuration

Launch with: bundles selected below   Framework: Equinox   Default Start level: 4   Default Auto-Start: true

Bundles
- Target Platform
  - com.google.guava (10.0.1.v201203051515)
  - com.ibm.icu (4.4.2.v20110823)
  - com.jcraft.jsch (0.1.46.v201205102330)
  - com.sun.el (2.2.0.v201108011116)
  - com.sun.syndication (0.9.0.v200803061811)
  - de.hu_berlin.german.korpling.saltnpepper.pepper-exceptions (1.1.5.SNAPSHOT)

0 out of 883 selected

[✓] Include optional dependencies when computing required bundles
[✓] Add new workspace bundles to this launch configuration automatically
[ ] Validate bundles automatically prior to launching

- For not loading all available bundles, which could be more than 800 (see Figure 2.5, “” position B), press the button 'Deselect All' (see Figure 2.5, “” position C).
- Now, activate all necessary bundles for running Pepper. In 'type filter text' (see Figure 2.5, “” position D) type in 'de.hu_berlin' and you will see all bundles having this string in the package name. To active the necessary bundles, click the box on the left of their name. Make sure, that all bundles you have copied from PEPPER_HOME are activated. These are at least the following ones:
  - de.hu_berlin.german.korpling.saltnpepper.pepper-exceptions.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-framework.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-logReader.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-modules.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-moduleTests.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-testEnvironment.jar
  - de.hu_berlin.german.korpling.saltnpepper.pepper-workflow.jar
  - de.hu_berlin.german.korpling.saltnpepper.salt-graph.jar
  - de.hu_berlin.german.korpling.saltnpepper.salt-saltCommon.jar
  - de.hu_berlin.german.korpling.saltnpepper.salt-saltCore.jar
- There are some other bundles necessary for running Pepper to be activated:
  - org.eclipse.equinox.console
  - org.eclipse.equinox.ds
  - org.apache.felix.gogo.shell
  - org.apache.log4j

- Since these are not all of the necessary bundles, we have to select some more, but eclipse now should be able to detect them automatically, by pressing the button 'Add Required Bundles' (see Figure 2.5, “” position E).

### Note

In case of this button is disabled, remove the string from the filter

Now some more bundles are added, on the right you can see the increased number of currently selected bundles (see Figure 2.5, “” position B).

- Since the start order of some bundles is important, set 'Start Level' of the following bundles as follows.

### Table 2.1. Sample Table

| **Bundles** | **Start Level** |
|---|---|
| de.hu_berlin.german.korpling.saltnpepper.pepper-logReader | 2 |
| de.hu_berlin.german.korpling.saltnpepper.pepper-framework | 5 |
| de.hu_berlin.german.korpling.saltnpepper.pepper-testEnvironment | 6 |

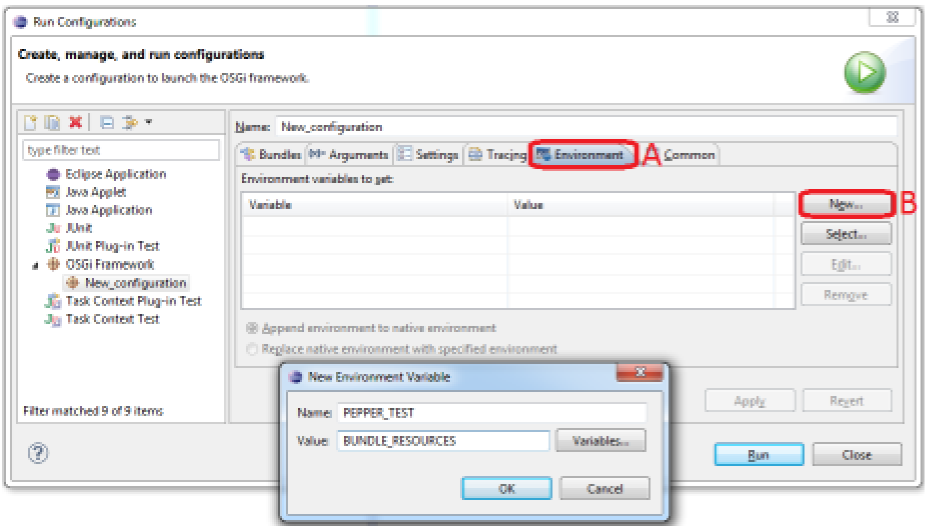
### Note

We assume, that the 'Default Start Level' is set to 4.

- Now we need to set an environment variable named 'PEPPER_TEST' to register resources like configuration files of the bundles, if needed.

1. Go to tab 'Environment' (see Figure 2.6, “” position A).

### Figure 2.6.



Run Configurations

**Create, manage, and run configurations**

Create a configuration to launch the OSGi framework.

Name: New_configuration

Environment variables to set:

| Variable | Value |
|---|---|

New Environment Variable

Name: PEPPER_TEST

Value: BUNDLE_RESOURCES

Filter matched 9 of 9 items

2. Press button 'New...' (see Figure 2.6, “” position B).

3. Type in 'PEPPER_TEST' for name and the folder known as BUNDLE_RESOURCES as value.

- Press button 'Run' (see Figure 2.5, “” position F).

Now Pepper is running and shows all registered importers, exporters and manipulators.

To make things easier, we provide a preconfigured file, which can be used for this, unfortunatly this configuration is for eclipse only. This file contains information for the run configurations of the pepper-testEnvironment project and can be found under `PEPPER_HOME/pepper-testSuite/pepper-testEnvironment/pepper-testEnvironment.launch`.

To install further pepper modules, follow the given instructions:

1. Download a module of your choice from http://korpling.german.hu-berlin.de/saltnpepper/repository/pepperModules/de/hu_berlin/german/korpling/saltnpepper/pepperModules/ or any other source.
2. Unzip the file to MODULE_HOME
3. Copy the bundel (.jar file) to ECLIPSE_HOME/dropins
4. Copy the folder having the same name as the bundle (.jar file) to BUNDLE_RESOURCES/plugins.
5. Select the downloaded module in your run configurations.
6. Press button 'Add required bundles' (see Figure 2.5, “” position E).
7. To check if all bundle dependencies could be resolved, press 'Validate Bundles'. (see Figure 2.5, “”)

## Making a real run

Now we want to start a real run using a workflow description file. To do so, we need a sample corpus and a sample workflow description file. The sample workflow description file specifies a workflow converting a corpus in paula to relANNIS.

**Note**

Make sure, that the PAULAImporter and the RelANNISExporter is registered in your eclipse installation (see 'plugins view' the section called “Starting eclipse”) and in your run configurations.

- Download the sample corpus from http://korpling.german.hu-berlin.de/saltnpepper/samples/corpora/pcc2.zip and unzip it to SAMPLE_CORPUS.
- Create an environment variable named 'PEPPER_TEST_WORKFLOW_FILE' in your 'Run configurations' and point it to SAMPLE_CORPUS/paula2paula.pepperparams (see Figure 2.6, “” for creating environment variables; name='PEPPER_TEST_WORKFLOW_FILE', value='SAMPLE_CORPUS/paula2paula.pepperparams')
- Press 'Run'.

Now the pepper test environment is set up, and pepper does the conversion task.

## Setting up maven

As already mentioned, we used maven as lifecycle and dependency management tool. For the developement of your own modules, it would be very helpful, to also use maven. But you are free to use any other lifecycle management tool like ant (see http://ant.apache.org/), gradle (see http://www.gradle.org/) or none.

If you are working with Netbeans, you can skip the rest of this section, because Netbeans already comes with an integrated maven. In the case of you are working with eclipse, or even want to compile your pepper module via the command line, you have to install maven. Therefore just follow the given instructions:

- Download the latest maven distribution from http://maven.apache.org/ (we used version 3.0.3).
- Unzip the file to a folder of your choice and you will get a folder like apache-maven-version in it. Lets call it MAVEN_HOME.

Now, when switching to MAVEN_HOME/bin folder, you can run maven from the command line via calling:

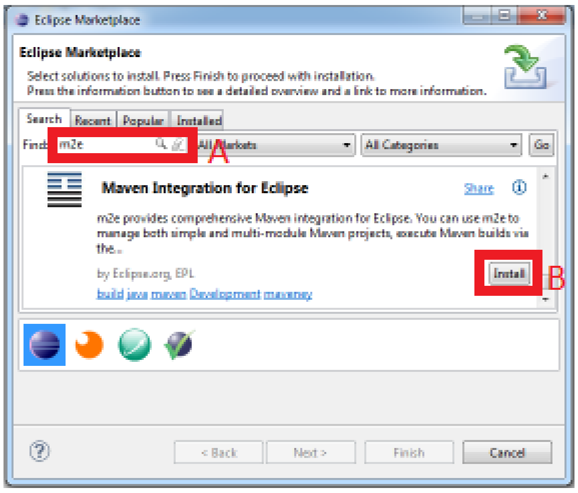
```
mvn
```

If you want to run mvn everywhere, write the path MAVEN_HOME/bin into your path environment variable (under Windows) or create a symlink to that path (under linux).

To enable maven even in eclipse you have to use a specific plugin called m2e.

- Open the 'Eclipse Marketplace' via 'Help --> Eclipse Marketplace...' (see Figure 2.7, “”)

**Figure 2.7.**



Eclipse Marketplace

**Eclipse Marketplace**

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Search | Recent | Popular | Installed

Find: m2e | All Markets | All Categories | Go

**Maven Integration for Eclipse** Share

m2e provides comprehensive Maven integration for Eclipse. You can use m2e to manage both simple and multi-module Maven projects, execute Maven builds via the...

by Eclipse.org, EPL

build java maven Development maveney

- Type in 'm2e' in text box 'Find:' and press enter (see Figure 2.7, “” position A).
- The plugin 'Maven integration for Eclipse' should be displayed. Click "Install" to install that plugin (see Figure 2.7, “” position B).
- During the installation eclipse asks you to agree to the license of this plugin and may be recommends to restart eclipse. Just follow the instructions and go ahead.
- After restarting eclipse, the plugin should be installed. You can check the installation, when clicking on 'File --> Import'. The dialog which opens should contain an option/a folder named 'Maven'.

If you are working with eclipse and the command line simultaneously (which might be very helpful, since maven does not always run error-free in eclipse), it might be useful, to synchronize the maven local repository of your m2e installation and the one of your maven installation.

- Go to 'Window --> Preferences'.
- Expand the entry 'Maven'.
- Select the 'Installations' view.
- Press the 'Add' button on the upper right and choose MAVEN_HOME.

A pepper module is created using the OSGi declarative service mechanism. If you don't know what this is and you don't want to know, never mind. We put the OSGi layer in the background so that you do not have to care about (as long as everything works fine ;-)). But therefore we used maven to configure the OSGi part. To enable the declarative service stuff, we used some maven dependencies called 'carrot-osgi-anno-scr'. Since there is no m2e adapter installed in eclipse by default, you have to download a further plugin.

- Go to the 'Eclipse Marketplace' via 'Help --> Eclipse Marketplace...'.
- Enter in 'Find:' the string 'CarrotGarden' and press enter.
- The dialog will show the 'CarrotGarden SCR' plugin. Press the 'install' button and follow the instructions.

# Chapter 3. Adopting the SampleModules

## Download the skeleton

To start with creating an own module for Pepper, please download the SampleModule, to be used as a skeleton for your own module. SampleModules is stored in our svn repository, therefore please export the project via an SVN client. The SVN location depends on the version you want to use and can be found at SampleModules (see: https://korpling.german.hu-berlin.de/svn/saltnpepper/PepperModules/SampleModules/tags).

## Adopting the project

Here, we give a list of things to do to adapt the project to your needs. In some cases, the order of the entries is arbitrary, but in some cases it is easier to respect the given order.

- import the project as a Maven project (when working with eclipse, use the import option existing maven project, it might be that you have to install further plugins or m2e connectors. So don't be scared if a dialog pops up.)

- rename the project (when working with eclipse, right click on the project name and choose the menu entry "Refactor" --> "Rename...")

- rename the packages (when working with eclipse, right click on the package name and choose the menu entry "Refactor" --> "Rename...")

## Adopting the pom.xml

Maven follows the paradigm convention over configuration, therefore a lot of things are already predefined and not has to be changed.

But still, there are some things like the name of the project and so on, which are project specific and therefore has to be adopted. Just follow the given instructions to adopt the 'project object model' (pom). The items in the numbered list correspond to the TODO entries in the pom.xml file. You will find the pom.xml file in the parent directory of the SampleProject.

1. Change the "groupId" to the name of your module (conventionally, the groupId is the same as the package name). You will find the entry under '/project/groupId'.

2. Change the "artifactId" (to the module name). You will find the entry under '/project/artifactId'.

3. Modify the description: Write what the module is supposed to do. You will find the entry under '/project/description'.

4. Change the project homepage to the url of your project. You will find the entry under '/project/url'.

5. Change the issue tracker to the one you are using, in case of you do not use any one, remove this entry. You will find the entry under '/project/issueManagement'.

6. Change the continuous integration management system to the one you are using, in case of you do not use any one, remove this entry. You will find the entry under '/project/ciManagement'.

7. Change the inception year to the current year. You will find the entry under '/project/inceptionYear'.

8. Change the name of the organization to the one you are working for. You will find the entry under '/project/organization'.

9. Modify the scm information or remove them (the scm specifies the location of your versioning repository like SVN, GIT, CVS etc.). You will find the entry under '/project/scm'.

10. Import needed maven dependencies (this is necessary to resolve dependencies to libraries handled by maven). You will find the entry under '/project/dependencies'.

11. Change the connection to the tags folder of your scm, what you can see here is the subversion connection for the pepperModules-SampleModules project. You will find the entry under '/project/build/plugins/plugin/configuration/tagBase'.

12. Sometimes it is necessary to include libraries, which are accessible via a maven repository and therefore can not be resolved by maven. In that case we recommend, to create a 'lib' folder in the project directory and to copy all the libraries you need into it. Unfortunatly, you have register them twice, first for maven and second for OSGi.

    To register such a library to maven, you need to install them to your local maven repository. You can do that with:

    ```
                            mvn install:install-file -Dfile=JAR_FILE -DgroupId=GR
                            -Dversion=VERSION -Dpackaging=PACKAGING
    ```

    Now you need to add the library as a dependency to your pom. The following snippet shows an example:

    ```
    <dependency>
      <groupId>GROUP_ID</groupId>
      <artifactId>ARTIFACT_ID</artifactId>
      <version>VERSION</version>
    </dependency>
    ```

    To make them accessible for OSGi, add them to the bundle-classpath of the plugin named 'maven-bundle-plugin'. You will find the entry under '/project/build/plugins/plugin[artifactId/text()= 'maven-bundle-plugin']/configuration/instructions/Bundle-ClassPath'. You further need to add them to a second element named include-resource, which you will find under '/project/build/plugins/plugin[artifactId/text()= 'maven-bundle-plugin']/configuration/instructions/Include-Resource'. The following snippet gives an example:

    ```
    <Bundle-ClassPath>.,{maven-dependencies}, lib/myLib.jar</Bundle-ClassPath>
    <Include-Resource>{maven-resources}, LICENSE, NOTICE, lib/myLib.jar=lib/myLib
    ```

    You Include libraries not handled by maven, i.e. jar files, by setting the bundle-path and extending the include-resources tag. "/project/build/plugins/plugin[artifactId/text()= 'maven-bundle-plugin']/configuration/instructions/Include-Resource"

## Adopting the Java code

This section describes how to customize the importer, manipulator or exporter. Therefore you will find in the `SampleModule` three classes named `SampleImporter`, `SampleManipulator` and `SampleExporter`. All of them are located in the package de.hu_berlin.german.korpling.saltnpepper.pepperModules.sampleModules. In most cases people do not want to create all of them sometimes people only want to create an importer an exporter or a manipulator. Then just delete the classes you won't implement, In case of you want to have several importer, manipulators or exporters, just copy the classes and proceed with each class as described in the following. Otherwise, you will end up with non-functional modules in your project.The items in the numbered list correspond to the TODO entries in the class files.

1. Change the name of the component, for example use the format name and the ending Importer or Exporter e.g. FORMATExporterComponent and FORMATExporterComponentFactory, in case of you want to create a manipulator, use a name describing the task e.g. MYTaskManipulatorComponent and MYTaskManipulatorComponentFactory.

2. Change the name of the module, for example use the format name and the ending Exporter (`FORMATExporter`).

3. In case of you are implementing an im- or exporter, change "sample" with format name and 1.0 with format version to be supported.

4. In case of you are creating an importer, override the method `importCorpusStructure()` (see: the section called "`importCorpusStructure`"), this method maps the corpus structure of the given format to a corpus structure in Salt.

5. If you want to have a specific handling with the document strcuture, override the method `start()`, for instance if you want to have athreading. Otherwise just delete this method or call `super.start()`, than the method `start(SElementId sElementId)` will be called for each `SDocument` automatically.

6. Override the `method start(SElementId sElementId)`, this is the point of mapping documentstructures between Salt and a format or to place the functionality of a manipulator.

7. If some clean ups left, override the method `end()`.

## importCorpusStructure

- imports corpus structure, into a `SCorpusGraph`, means creates a structure consisting of `SDocument` and `SCorpus` node objects and `SCorpusRelation` and `SCorpDocRelation` relation objects connecting them.

- corpus structure super and sub corpus, a corpus can contain 0..* subcorpus and 0..* documents, a corpus not contain both document and corpus

- creates a table document-id to uri for method `start()`

- method `isFileToImport()` does following and can be overridden

Computes a corpus-structure given by the file-structure located by the address of the given `URI` object. For the root folder (the direct `URI` location) a root-corpus (`SCorpus` object) is created. For each sub-folder a sub-corpus (`SCorpus` object) is created. For each file for which the method `isFileToImport(URI, List)` returns `true`, a `SDocument` object is created and added to the current `SCorpusGraph` object. The objects are connected via `SCorpusRelation` or `SCorpusDocumentRelation` objects. While traversing the file-structure, an `SElementId` object is created representing the corpus-hierarchie and added to the created `SCorpus` or `SDocument` objects. A map of these `SElementId` objects corresponding to the `URI` objects is returned, so that in method `start(SElementId)` this map can be used to identify the `URI` location of the `SDocument` objects.

### Note

For a description of how to add entries to the file extension list, see `isFileToImport(URI, List)`.

## `isFileToImport`

To customize the computation if a file shall be imported or not, you can override the method `isFileToImport`. The method in its original returns if a given `URI` object shall be imported during import phase. This decision depends on the kind and the content of the given fileExtension list. The file extension list can contain a set of file extensions (Strings without '.') to be imported or marked as to be not imported (via the prefix `PepperImporter.NEGATIVE_FILE_EXTENSION_MARKER`). The following list shows the condition of computation for returned value:

- This method returns false, in case of the given `URI` object is null.
- This method returns true for every `URI` object in case of the list is null or empty.
- This method returns true if the given list is a positive list (does not contain the negative marker `PepperImporter.NEGATIVE_FILE_EXTENSION_MARKER` at all) and the file extension of the uri is contained in the list.
- This method returns true if the given list is a negative list (any item is prefixed with the negative marker `PepperImporter.NEGATIVE_FILE_EXTENSION_MARKER`) and the file extension of the uri is not contained in the list.

**Note**

When a list contains items prefixed with the negative marker and items which are not, the list is interpreted as a negative list.

In case of you want to customize the method by overriding it, but not loosing its functionality, you can use the following lines of code:

```
@Override
protected boolean isFileToImport(URI checkUri, List<String> fileExtensions)
{
    if (super.isFileToImport(checkUri))
    //TODO: do something, when super says yes
    else
    //TODO: do something, when super says no
}
```

## building the project

We adopt the settings for using OSGi and to create PepperModules, which are simply plugable into the Pepper framework. So in the best case, you can just follow this guide, and run

```
            mvn install assembly
```

and you will find a zip file in the target folder of your project (YOUR_PROJECT/target/distribution), which can simply pluged into Pepper via extracting its content to the plugin folder of Pepper (PEPPER_HOME/plugins). In the bad case some mean errors occur and you have to dig in a lot of online tutorials and forums and lose a lot of time. Please do not despair and write us an e-mail instead. `<saltnpepper@lists.hu-berlin.de>`

# Chapter 4. Customizing behaviour of your PepperModule

### *via properties*

When creating a mapping, it is often a matter of choice to map some data this way or another. In such cases it might be clever not to be that strict and allow only one possiblity. It could be beneficialy to leave this decision to the user. Customizing a mapping will increse the power of a PepperModule enormously, because than it can be used for wider range of purposes without rewriting parts of it. The pepper framework provides a property system to access such user customizations. Nevertheless, a PepperModule shall not be dependant on user customization. The past showed, that it is very frustrating for a user, when a PepperModule breaks up, because of not specifified properties. You should always define a default behaviour in case of the user has not specified one.

## Property

A property is just an attribute-value pair, consisting of a name so called property name and a value so called property value. Properties can be used for customizing the behaviour of a mapping of a PepperModule. Such a property must be specified by the user and determined in the pepper workflow description. The pepper frsamework will pass all customization properties direct to the instance of the PepperModule.

**Note**

In the current version of pepper one has to specify a property file by its location in the pepper workflow description file (.pepperParams) in the attribute @specialParams inside the <importerParams>, <exporterParams> or <moduleParams> element. In the next versions this will change to a posibility for adding properties directly to the pepper workflow description file.

## Property registration

The pepper framework provides a kind of a registration for customization properties. This registry is called `PepperModuleProperties` and can be accessed via `getProperties()` and `setProperties()`. This class only represents a container object for a set of `PepperModuleProperty` objects and provides accessing methods. An instance of `PepperModuleProperty` represents an abstract description of a property and the concrete value at once. In the registration phase it belongs to the tasks of a `PepperModule` to specify the abstract description which consists of the name of the property, its datatype, a short description and a flag specifying if this property is optional or mandatory. To create such an abstract description of a property use the constructor:

```
PepperModuleProperty(String name,
                     Class>T< clazz,
                     String description,
                     Boolean required);
```

and pass the created property object to the property registry by calling the method `addProperty`. The pepper framework uses the registry to first inform the user about usable properties for customization and second to fullfill the property objects with the property values set by the user.

The value of a specific property can be accessed by passing its name to the registry. The method to be used is the following one:

```
getProperty(String propName);
```

The easiest way of creating an own class for handling customization properties is to derive it from the provided class `PepperModuleProperties`. Imagine you want to register a property named 'MyProp' being of type String, which is mandatory to a property class called 'MyModuleProperties'. For having an easier access in your pepper module, you can enhance the MyModuleProperties class with a getter method for property MyProp (see: getMyProp()).

```
//...
import de.hu_berlin.german.korpling.saltnpepper.pepper.pepperModules.PepperModu
import de.hu_berlin.german.korpling.saltnpepper.pepper.pepperModules.PepperModu
//...
public class MyModuleProperties extends PepperModuleProperties
{
    //...
    public MyModuleProperties()
 {
    //...
    this.addProperty(new PepperModuleProperty<String>("MyProp", String.class, "
    //...
 }
 //...
 public String getMyProp()
 {
  return((String)this.getProperty("MyProp").getValue());
 }
}
```

## checking property constraints

Since the vlaue of a property can be required, you can check if its value is set by calling the method `checkProperties()`. To customize the constraints of a property, you can override the methode `checkProperty(PepperModuleProperty<?>)`. Imagine a property named 'myProp' having a file as value, you might want to check its existance. The following snippet shows the code how this could be done:

```
public boolean checkProperty(PepperModuleProperty<?> prop)
{
   //calls the check of constraints in parent, for instance if a required value
   super.checkProperty(prop);
   if ("myProp".equals(prop.getName()))
   {
       File file= (File)prop.getValue();
       //throws exception, in case of set file does not exist
       if (!file.exists())
           throw new PepperModuleException("The file set to property 'myProp' d
   }
   return(true);
}
```

# Chapter 5. Documenting your PepperModule

One of the most important but often forgotten tasks when creating a PepperModule is to document the behaviour of it and its functionalities. Therefore the project SampleModules contains a template for creating a documentation. The documentation in SaltNPepper in general is done in DocBook (see: docbook.org/). DocBook is a documentation language written in XML and enables to transform the documentation into several target formats like html, pdf, odt, doc etc..

**Note**

We recommend to use the template, for a uniform view to all PepperModules. That makes it possible not to forget important issues to be mentioned and makes it easier for the user to have a good understanding of what the PepperModule is doing.

You will find the template in `SAMPLE_MODULES_HOME/src/main/docbkx/manual.xml` among other directories containing files necessary for the transformation. To refer to images from the manual, put them into the image folder `SAMPLE_MODULES_HOME/src/main/docbkx/images/` and make a relative reference.

## Transformation

The standard transformation which is configured for SampleModules is the transformation to html and pdf. The configuration is done in the `pom.xml` of the pepper-parentModule project. To add further output formats, just copy the respective plugins to your pom and change them.

When executing the maven goal site

```
mvn clean site
```

, maven will create a manual folder under `SAMPLE_MODULES_HOME/target/manual`, where you can find the pdf documentation and the html documentation.

**Note**

The current configuration does not need to be changed, just write your documentation by overriding the template. The rest shall work automatically. In some cases it might be necessary to adopt the transformation, than please take a look to the xsl transformation files in `SAMPLE_MODULES_HOME/docbook-xsl`.

# Chapter 6. Testing your PepperModule

## Running Unit Test

In every good book about computational programming it is written that testing of the software you are developing is a very important issue. Since testing increases the quality of software enormously, we would agree in that. But unfortunatly testing of code could get an annoying and painful task. Even if it starts to take longer to develop tests than develop the productive software. Therefore we tried to help you to faster implement the test code. In the project pepper-moduleTests (part of project pepper-testSuite) you will find three test classes named PepperManipulatorTest, PepperImporterTest and PepperExporterTest. These classes use the JUnit test framework (see: junit.org) and implement some very basic tests for checking the consistency of a PepperModule. Just benefit from these classes by creating an own test class derived from one of the provided ones and your tests will be ran during the maven build cycle. For getting an immediate feedback, you can also run them directly in your developement environment by running a JUnit task. One the hand the test classes provide tests which can be adopted to your needs and check if your module can be plugged into the pepper framework (by checking if necessary values are set like the supported format for an im- or exporter). And on the other hand, they provide some helper classes and functions, which can be used when adding further test methods for checking the functionality of your module.

**Note**

We strongly recommend, to add some module specific test methods, to increase the quality of your code and even to make it easier changing things and still having a correct running module.

## Adopting consistency tests

In case of you are implementing an im- or exporter, you need to set the supported formats and versions in your test case. Pepper will check them automatically, but the test environment need to know the correct pairs of format name and version. To do so, just add the following lines to method `setUp()`:

```
protected void setUp() throws Exception
{
    //...
    FormatDefinition formatDef= PepperModulesFactory.eINSTANCE.createFormatDefi
    formatDef.setFormatName("FORMAT_NAME");
    formatDef.setFormatVersion("FORMAT_VERSION");
    this.supportedFormatsCheck.add(formatDef);
    //...
}
```

Replace FORMAT_NAME and FORMAT_VERSION, with your specific ones. You can even add more than one `FormatDefinition` object.

## Writing own tests

The prvided test classes are faking the pepper environment, so that you can run your entire module and just check the in or output.

In case of you write an importer, you can create an input file containing a corpus in the format you want to support, run your importer and check its output against a defined template. The test will return a processed salt graph which can be checked for its nodes, edges and so on. The following snippet shows how to read a document and how to check the returned salt model:

```
public void testSomeTest()
{
    //start: creating and setting corpus definition
  CorpusDefinition corpDef= PepperModulesFactory.eINSTANCE.createCorpusDefiniti
  FormatDefinition formatDef= PepperModulesFactory.eINSTANCE.createFormatDefini
  formatDef.setFormatName(FORMAT_NAME);
  formatDef.setFormatVersion(FORMAT_VERSION);
  corpDef.setFormatDefinition(formatDef);
  corpDef.setCorpusPath(URI.createFileURI(PATH_TO_SAMPLE_CORPUS));
  this.getFixture().setCorpusDefinition(corpDef);
 //end: creating and setting corpus definition

    //...

    //create an empty corpus graph, which is filled by your module
    SCorpusGraph importedSCorpusGraph= SaltCommonFactory.eINSTANCE.createSCorpu
    // add the corpus graph to your module
    this.getFixture().getSaltProject().getSCorpusGraphs().add(importedSCorpusGr

    //run your PepperModule
    this.start();

    //check the processed corpus graph object
        //checks that the corpus graph object is not empty any more
        assertNotNull(importedSCorpusGraph.getSCorpora());
        //checks for instance, that the corpus graph contains X SCorpus objects
        assertEquals(X, importedSCorpusGraph.getSCorpora().size());
    //...
}
```

For testing an exporter, you may want to use the sample creator for salt models, which is part of the salt project. You will find it as a subproject named salt-saltSample. Here you will find a lot of methods creating a sample salt model with the possibility to just create the layers, you want to use.

## Running live tests

For running an integration test of your module in the pepper framework and for testing interferences between your module and other modules, pepper comes with a test environment. Therefore, you need to set up the OSGi environment correclty and set some environment variables. If you have not done that already, please check #sec_letsRun.

# Chapter 7. FAQ

## Cannot run maven install under Eclipse?

Restart Eclipse and try again, sometimes Eclipse does not recognize a maven project at first time.