

# **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}

Copyright © 2012 ??, ??, ??, ??,???, All rights reserved.

---

# Table of Contents

Foreword .....	vi
1. Introduction .....	1
2. Setting up environment .....	2
Download and set up eclipse .....	2
Making eclipse ready for SaltNPepper .....	2
Starting eclipse .....	2
Let's run .....	4
Making a real run .....	7
Setting up maven .....	7
3. Adopting the SampleModules .....	10
Download the skeleton .....	10
Adopting the project .....	10
Adopting the pom.xml .....	10
Adopting the Java code .....	11
importCorpusStructure .....	11
isFileToImport .....	12
building the project .....	12
4. Customizing behaviour of your PepperModule .....	14
Property .....	14
Property registration .....	14
checking property constraints .....	15
5. Documenting your PepperModule .....	16
Transformation .....	16
6. Testing your PepperModule .....	17
Running Unit Test .....	17
Running live tests .....	17
environment variables .....	17
Configure OSGi environment .....	18
7. FAQ .....	19
Cannot run maven install under Eclipse? .....	19

---

## List of Figures

2.1. ....	3
2.2. ....	3
2.3. ....	4
2.4. ....	4
2.5. ....	5
2.6. ....	6
2.7. ....	8

---

## List of Tables

2.1. Sample Table .....	6
-------------------------	---

---

# Foreword

The aim of this document is to provide a helpful guide to create own PepperModules to plug them into the pepper framework. Currently this document is not in a very good state so far. It is not more than a loose collection of hopefully interesting information for developers. But we are working on it, to fullfill its aim and to make a helpfull and readable guide out of it.

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

Since Pepper is a plugable framework, we used an underlying framework called OSGi (see: <http://www.osgi.org/>) providing such a functionality. 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 called 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.

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](mailto: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 development 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/](http://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>). Unfortunately, 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/](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.

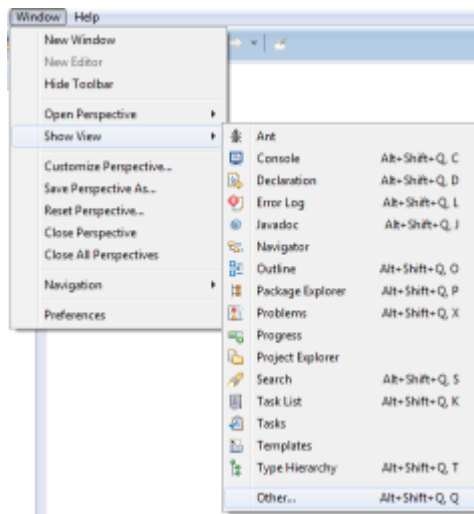
## Starting eclipse

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, “”).

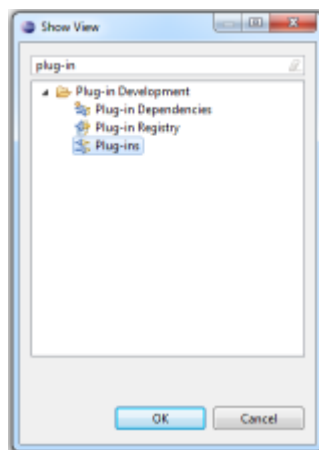


**Figure 2.1.**



- 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 accessible 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

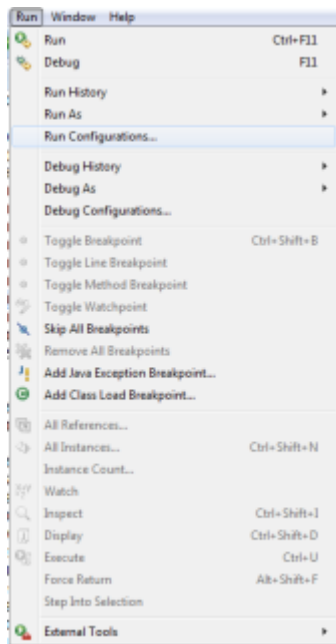
Congratulations, now you are done setting up your IDE.

## Let's run

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

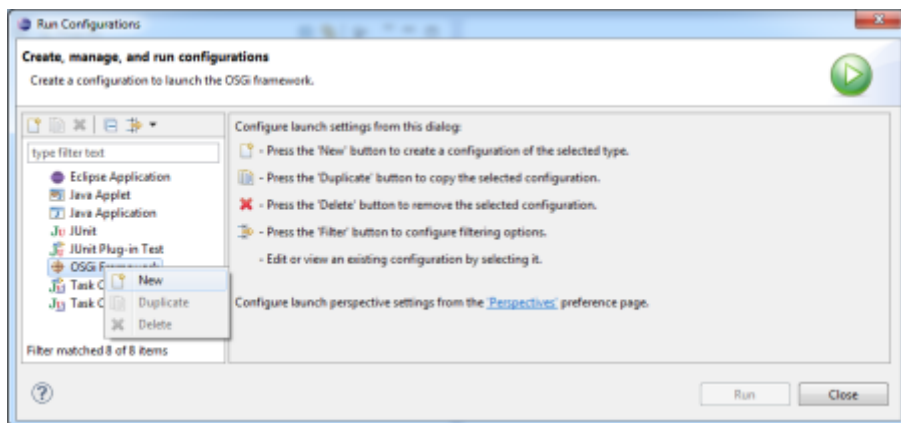
- Select 'Run --> Run configurations...' (see Figure 2.3, “”).

**Figure 2.3.**



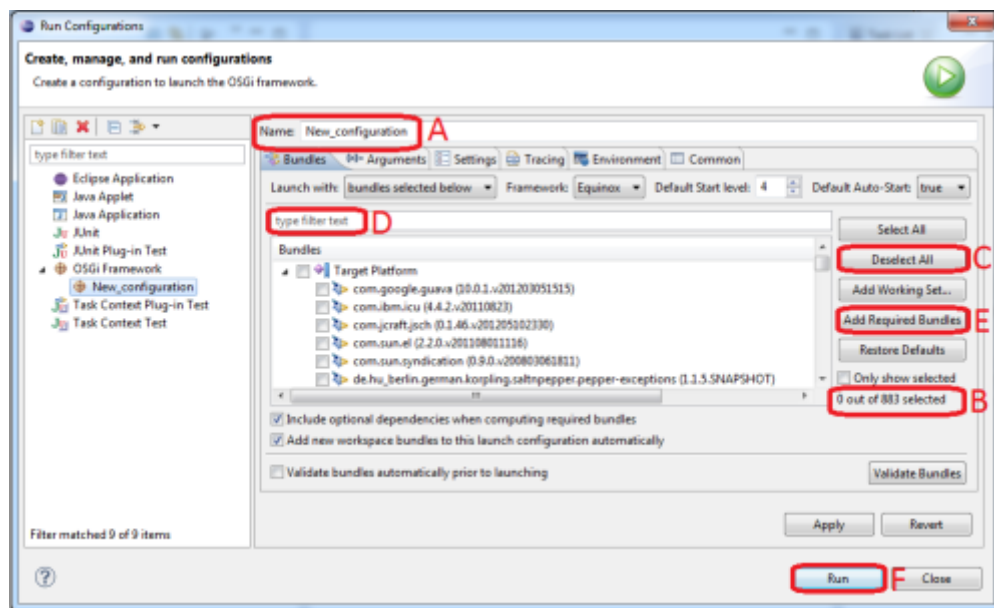
- Select 'OSGi Framework', press right mouse button and select 'New' (see Figure 2.4, “”).

**Figure 2.4.**



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

Figure 2.5.



- 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 activate 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



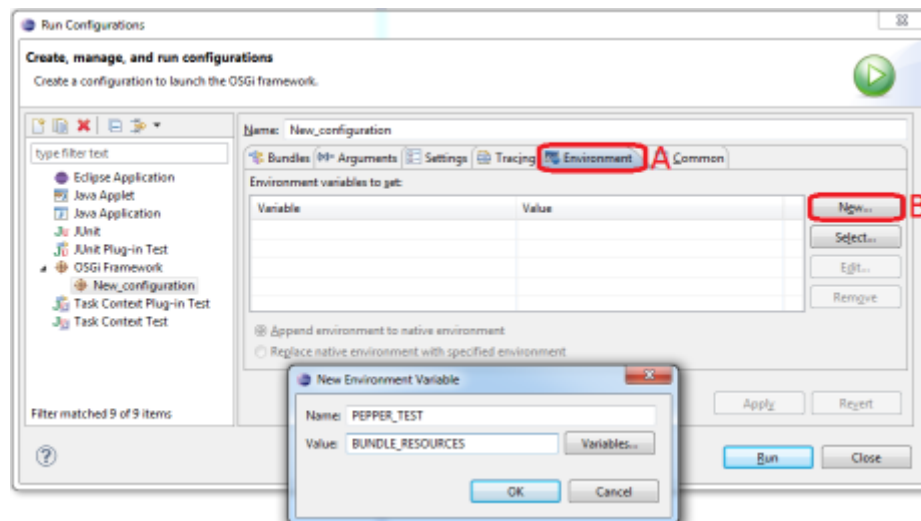
### Note

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.**



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 install further pepper modules, follow the given instructions:

1. Download a module of your choice from [http://korpling.german.hu-berlin.de/saltnepper/repository/pepperModules/de/hu\\_berlin/german/korpling/saltnepper/pepperModules/](http://korpling.german.hu-berlin.de/saltnepper/repository/pepperModules/de/hu_berlin/german/korpling/saltnepper/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/saltnepper/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 development 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 folder of your choice and you will get a folder like `apache-maven-version`. Lets call it `MAVEN_HOME`.

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

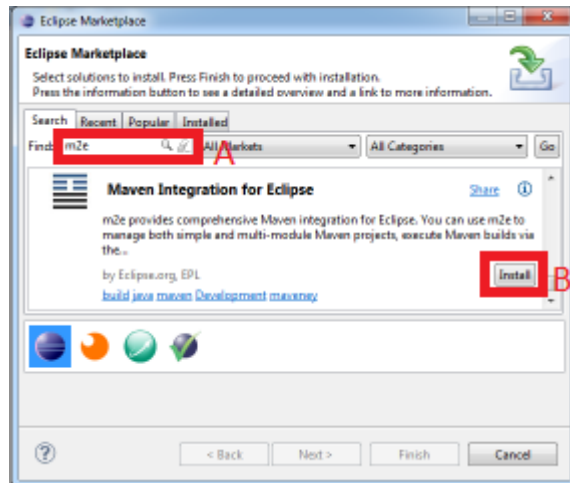
```
mvn
```

. If you want to run mvn everywhere, write enter 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.**



- 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 meaningful, 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 it is and you don't want to 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/>).

## Adopting the project

Before we start adapting the modules, please think about what kinds of modules you want to provide in your project. If you just want to implement an importer, delete the exporter and manipulator code from the SampleModule. Otherwise, you will end up with non-functional modules in your 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)
- rename the project (when working with eclipse, right click on the project name and choose the menu entry refactor --> rename)
- rename the package (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 one, 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.

- change the groupId to the name of your module (conventionally, the groupId is the same as the package name)
- change the artifactId (to the module name)
- modify the description: Write what the module is supposed to do
- change the project homepage to the url of your project
- change the issue tracker to the one you are using, in case of you do not use any one, remove this entry
- change the continuous integration management system to the one you are using, in case of you do not use any one, remove this entry
- change the inception year to the current year
- change the name of the organization to the one you are working for



- modify the scm information or remove them (the scm specifies the location of your module's SVN repository)
- import needed maven dependencies (this is necessary to resolve dependencies to libraries handled by maven)
- change the connection to the tags folder of your scm, what you can see here is the subversion connection for the pepperModules-SampleModules project
- include libraries not handled by maven, i.e. jar files, by setting the bundle-path and extending the include-resources tag.

## 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. The items in the numbered list correspond to the TODO entries in the class files.

- 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`
- change the name of the module, for example use the format name and the ending Exporter (`FORMATExporter`)
- in case of you are implementing an im- or exporter, change "sample" with format name and 1.0 with format version to support
- in case of you are creating an importer, override the method `importCorpusStructure()`, this method maps the corpus structure of the given format to a corpus structure in Salt
- 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.
- 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
- 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 `SCorpusDocRelation` 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-hierarchy 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 pluggable into the Pepper framework. So in the the good 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 be plugged into Pepper via extracting its content to the plugin folder of Pepper. 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 inside to either map any format or model to salt or a salt model to another salt model or a salt model to any model or format, 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 possibility. It could be a good idea to leave this decision to the user. Customizing a mapping will increase the power of a PepperModule enormously, because then it can be used for many purposes without rewriting parts of it. Therefore 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, when a PepperModule breaks up, because of not specified properties. There should always be 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 framework 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 possibility 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 fulfill 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);
```

## checking property constraints

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

```
public boolean checkProperty(PepperModuleProperty<?> prop) {
    //calls the check of constraints in parent, for inheritance
    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 prop does not exist");
    }
    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 template SampleModules contains a template for creating a documentation. The documentation in SaltNPepper in general is done in DocBook (see [docbook.org/](http://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 the 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 package ??? you will find test classes named `PepperManipulatorTest`, `PepperImporterTest` and `PepperExporterTest`, create your own test classes derived from them -these classes implement some tests which will be ran via building in maven or if you run a Unit test in development phase - the tests will help you implement good code - implemented tests will also check if module can be plugged into the pepper framework (means if all necessary values are set) - on second hand these classes provide some helper class which can be used for instance the class `PepperModuleTest.start()` runs your `PepperModule` like `Pepper` will do, but will return a salt graph, so that you can check the created salt graph against an expected template. - here is sample code, to use this tests:

```
//start: creating and setting corpus definition
CorpusDefinition corpDef= PepperModulesFactory.eINSTANCE.createCorpusDefinition()
FormatDefinition formatDef= PepperModulesFactory.eINSTANCE.createFormatDefinition()
formatDef.setFormatName(FORMAT_NAME_OF_TEST_CORPUS);
formatDef.setFormatVersion(FORMAT_VERSION_OF_TEST_CORPUS);
corpDef.setFormatDefinition(formatDef);
corpDef.setCorpusPath(URI.createFileURI(rootCorpus.getAbsolutePath()));
this.getFixture().setCorpusDefinition(PATH_TO_TEST_CORPUS);
//end: creating and setting corpus definition

SCorpusGraph importedCorpusGraph= SaltCommonFactory.eINSTANCE.createSCorpusGraph();
this.getFixture().getSaltProject().getSCorpusGraphs().add(importedCorpusGraph);
this.getFixture().importCorpusStructure(importedCorpusGraph);

//runs the PepperModule
this.start();

assertNotNull(importedCorpusGraph.getSDocuments());
assertEquals(..., importedCorpusGraph.getSDocuments().size());

assertNotNull(importedCorpusGraph.getSCorpora());
assertEquals(..., importedCorpusGraph.getSCorpora().size());
```

## Running live tests

For running your own `PepperModule` in a test environment, the pepper framework provides a special project for doing this called `pepper-testSuite`. This project first contains an environment to run your module called `pepper-testEnvironment` and contains a second project called `pepper-moduleTest` for checking the correctness of your `PepperModule`. Correctness means, that your module can be plugged into the environment and does not mean the logical correctness of its functionality. This project directly runs in the OSGi container and therefore has to be started in it. Because of there is starter outside the OSGi environment, you have to pass necessary resource locations and test corpora via environment variables.

## environment variables

- `PEPPER_TEST` to a folder, where the resources of the peppermodules are and the temporary stuff can be stored
- `PEPPER_TEST_WORKFLOW_FILE` to the `.pepperparams` file

## Configure OSGi environment

For running pepper and also its test environment it is very important to use the correct start order for plugins. For instance it is of special interest to start a logger as early as possible to get a logging and not to hide important messages like warnings or errors. For pepper it is important to run the the pepper framework after all plugins are started, otherwise, they will not be registrated to the pepper plugin registry and can therefore not be resolved.

To avoid this, set the start level (in OSGi) of "pepper-framework" to default-value + 2 and set the start value of the pepper-logReader to 0.

To make it easier, we provide a preconfigured file, which can be used for this, but only for eclipse. 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`. When using this configuration, it will not run correctly out of the box, because you have to enable your project first. Therefore open the run configuration in eclipse open the pepper-testEnvironment entry and click the box left to the name of your project. For checking not to forget one necessary dependency you can click the vldate button and the eclipse will retrieve all dependencies and list missing ones in case they exist.



---

## 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.