

# **Pepper**

## **User's Guide**

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

---

# Pepper: User's Guide

by Florian Zipser, , , and

Version \${project.version}

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

---

# Table of Contents

Foreword .....	v
I. Pepper User Guide .....	1
1. Pepper commandline interface (CLI) .....	2
Installing Pepper .....	2
Running Pepper .....	2
Non-Interactive mode .....	2
Interactive mode .....	3
2. ....	4
corpus organisation and workflow file .....	4
Relative paths .....	5
Absolute Paths .....	5
Plug in Pepper modules .....	6
Troubleshooting .....	6
3. Pepper as a library .....	7

---

## List of Figures

2.1. Folder structure showing best practices in how to organize a corpus .....	4
--	---

---

# Foreword

The aim of this document is to give the reader an overview of how to use Pepper in its different purposes. This document shows how to use the Pepper console, the Pepper GUI and how to run Pepper as a library in other programs. Currently only the console is provided.

---

# Part I. Pepper User Guide

Pepper is a pluggable, java-based, open-source converter framework for linguistic data. It was developed to convert data coming from a linguistic data format  $X$  to another linguistic data format  $Y$ . To decrease the number of conceptual mappings, pepper follows the intermediate model approach, which means that a conversion consists of two mappings. First, the data coming from format  $X$  will be mapped to the intermediate model Salt (see: <http://u.hu-berlin.de/saltnpepper>) and second, the data will be mapped from Salt to format  $Y$ . If you imagine a set of  $n$  source and target formats, then this approach will decrease the number of mappings from  $n^2$  mappings in case of the direct mapping approach to  $2n$  mappings.

Since Pepper is just a conversion framework and only takes the workflow control, the real conversion work is done by a set of Pepper modules. Such a module is an individual unit executing a specific task, like mapping data from or to a linguistic data format. A Pepper module can simply be plugged into the Pepper framework.

The Pepper workflow is separated into three different phases:

1. the import phase,
2. the manipulation phase and
3. the export phase.

The import phase handles the mapping from a format  $X$  to the Salt model, the export phase handles the mapping from a Salt model to a format  $Y$ . During the manipulation phase the data in a Salt model can be enhanced, reduced or manipulated. A phase is divided into several steps: the import and export phase each contain  $1$  to  $n$  steps whereas the manipulation phase contains  $0$  to  $n$  steps. Each Pepper module realizes such a step and therefore is associated with exactly one phase. The orchestration of Pepper modules is determined by the Pepper workflow description file (.pepperparams). A Pepper module can be identified by specifying its coordinates (its name or one of its supported formats and the corresponding format version). More information concerning the workflow file can be found in the section called “corpus organisation and workflow file”.

---

---

# Chapter 1. Pepper commandline interface (CLI)

## Installing Pepper

Pepper is system independent and comes as a ready to run zip archive, so you do not need any installation.

1. Download the latest version of Pepper SaltNPepper\_XXX.zip from <http://korpling.german.hu-berlin.de/saltnpepper/> [ <http://korpling.german.hu-berlin.de/saltnpepper/> ]
2. Unzip the folder to location of your choice (let's call it PEPPER\_HOME).

Your download already includes a set of Pepper modules. In some cases it is necessary to plug in further modules that are not already included or you need to update an included Pepper module. A guide about how to plug in modules can be found in section XXX .



### Note

Since Pepper is Java based, you need to have Java installed on your system. On most systems, Java is installed by default, but in case it is not please download it from [www.oracle.com/technetwork/java/javase/](http://www.oracle.com/technetwork/java/javase/). To check if Java is running, open a command line and run:

```
java -version
```

You need at least version 1.6.

## Running Pepper

Pepper is a command line program and can be invoked via a simple command line call. You can run Pepper in two modes, a non-interactive mode and an interactive mode (which currently is under construction and therefore can not do the conversion process. We are sorry.).

### Non-Interactive mode

In the non-interactive mode, Pepper runs using the passed parameters and terminates after the process is done. This mode normally is used, in case of Pepper is called by a script etc. or the workflow description is already created. To run Pepper, just call:

```
pepperStart.bat OPTIONS (when using Windows)
```

or

```
bash pepperStart.sh OPTIONS (when using linux, unix or Mac OS)
```

where OPTIONS is one of the following:

- workflow-file

Loads the passed 'workflow-file' and starts the conversion.

- list

Displays a table with information about all available Pepper modules.

- list module-name

Displays a table with information about the passed Pepper module.

- self-test

Checks whether the Pepper framework is ready to run or if any problems are detected, either in Pepper itself or in any registered Pepper module.

For creating a workflow description file, please see section the section called “corpus organisation and workflow file”.

## Interactive mode

The interactive mode opens a Pepper console and provides a wizzard, which guides you through the conversion process. Just call

`pepperStart.bat` (when using Windows)

or

`bash pepperStart.sh` (when using linux, unix or Mac OS)

Pepper welcomes you with prompt 'pepper>'. Now you can enter one of the following commands:

- list

Displays a table with information about all available Pepper modules.

- list module-name

Displays a table with information about the passed Pepper module.

- self-test

Checks whether the Pepper framework is ready to run or if any problems are detected, either in Pepper itself or in any registered Pepper module.



---

# Chapter 2.

## corpus organisation and workflow file

A Pepper workflow can be modeled and persisted in an xml file following a specific notation and having the ending '.pepperparams'. A conversion process consists of three phases, and the notation of a workflow file follows this structure. To identify a Pepper module realizing a step, you have to describe that module by its name or a description of the format of the corpus you want to convert (by the formats anem and version).

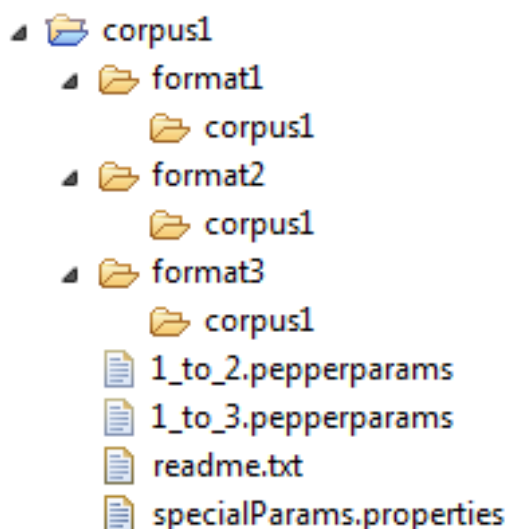


### Note

For each Pepper workflow description, the specification of at least one importer and one exporter is necessary, whereas the use of a manipulator is optional.

We now want to introduce a folder-structure, which makes it easy to have a good overview over all formats of one corpus and to bundle all data together. Our experiences in corpora have shown that when working over years with a corpus it is necessary, to have clear structure. Otherwise things could mess up. Imagine we have a corpus named *corpus1*, which is available in the formats *format1*, *format2* and *format3*. Now we want to convert the corpus a) from *format1* to *format2* and b) from *format1* to *format3*. Therefore we create two workflow files, *1\_to\_2.pepperparams* and *1\_to\_3.pepperparams*. Since a module can also take some customization properties we also have such a property file (*specialParams.properties*), containing attribute-value pairs (ATTRIBUTE\_NAME=VALUE). We also have a short description, which describes the corpus and its specifics in *readme.txt*. Such a bundle of data belonging together can be stored in a folder structure shown Figure 2.1, “Folder structure showing best practices in how to organize a corpus”)

**Figure 2.1. Folder structure showing best practices in how to organize a corpus**



When you use relative paths in your workflow description, you are able to share the corpus with others without having to adapt anything. You can define the workflow once and run it anywhere. This benefit can be interesting when working in a team, or in case a problem occurs and you want to send us the corpus to help fix the problem.

We now use the *1\_to\_2.pepperparams* as a sample, to explain the structure of workflow files in Pepper in the following snippet. In this sample, the importer *Format1Importer* needs some customizations passed with *specialParams.properties*. Further the workflow contains a manipulator named *Manipulator1* to do something with the data between im- and export.

---

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8"?>
<pepperParams:PepperParams xmlns:xmi="http://www.omg.org/XMI"
                             xmlns:pepperParams="de.hu_berlin.german.korpling.sa
                             xmi:version="2.0">
    <pepperJobParams id="1">
        <importerParams moduleName="Format1Importer" sourcePath="./format1" spe
        <moduleParams moduleName="Manipulator1"/>
        <exporterParams moduleName="Format2Exporter" destinationPath="./format2
    </pepperJobParams>
</pepperParams:PepperParams>

```

Im- and Exporters can also be addressed by specifying the format name and the format version of the corpus you want to im- or export. The Pepper framework will search for an import or export module handling this format. The following snippet shows the addressing that kind of addressing.

```

<?xml version="1.0" encoding="UTF-8"?>
<pepperParams:PepperParams xmlns:xmi="http://www.omg.org/XMI"
                             xmlns:pepperParams="de.hu_berlin.german.korpling.sa
                             xmi:version="2.0">
    <pepperJobParams id="1">
        <importerParams formatName="FORMAT_NAME"
                        formatVersion="FORMAT_VERSION"
                        sourcePath="..." />

        <!-- ... -->
        <exporterParams formatName="FORMAT_NAME"
                        formatVersion="FORMAT_VERSION"
                        destinationPath="..." />
    </pepperJobParams>
</pepperParams:PepperParams>

```



### Note

The xml attributes 'sourcePath', 'destinationPath' and 'specialParams' has to follow the URI notation, which is defined as follows:

```
[scheme:][//authority][path][?query][#fragment]
```

An overview of the java reference implementation can be found here for the interested reader: <http://download.oracle.com/javase/6/docs/api/java/net/URI.html>.

## Relative paths

To address a relative file path, use the [path] part of the uri expression. For instance to address the corpus 'corpus1' in a pepper workflow description, with the given file structure

```

|- .pepperParams
|  |-format1
|  |-corpus1

```

the corpus 'corpus1' can be addressed as shown here:

```
./format1/corpus1/
```

## Absolute Paths

For addressing absolute paths, one has to define a uri scheme. In the current version of Pepper, only the scheme 'file' is supported. The path of an absolute uri has to start with a leading '/' followed by the

---

absolute path. It is also allowed to define an empty authority, which results in three leading slashes. For instance for Windows the use of an absolute uri can look like this:

```
file:/C:/format1/corpus1/ (without authority)
```

```
file:///C:/format1/corpus1/ (with empty authority)
```

Or for linux and mac:

```
file:/format1/corpus1/ (without authority)
```

```
file:///format1/corpus1/ (with empty authority)
```

## Plug in Pepper modules

In most cases when you want to plug in a Pepper module you will get a zip file containing the module as a .jar file, and a folder having the same name as the jar file. This folder contains the license files, documentations and other resources the Pepper module needs. The need to plug in a Pepper module can be caused by two reasons:

- you want to update an already existing module or
- you want to install a new Pepper module, which is not already included

In case 1) move to the plugin folder of Pepper (PEPEPR\_HOME/plugins in default) and remove the plugin you want to update, by deleting the corresponding .jar file and the folder having the same name. This is necessary in order not to have the same Pepper module twice, because otherwise you cannot determine which one will be used in processing. After that or in case 2), it is very easy to get your new module running, just unzip the archive into the plugin folder of Pepper (PEPEPR\_HOME/plugins in default).

## Troubleshooting

Pepper and Salt are open source projects developed in a low budget environment, although we are doing our best to create stable systems, it can occur that you run into problems when using Pepper. In such cases, please read the error message displayed on the command line first. In many cases, a problem occurs because the data violates a constraint given by one of the Pepper modules. If this doesn't help, don't hesitate to write an e-mail to [saltnpepper@lists.hu-berlin.de](mailto:saltnpepper@lists.hu-berlin.de). Please don't forget to describe your problem as detailed as possible and send us the log files. You will find them under the '/logs' folder in your Pepper directory.

---

## Chapter 3. Pepper as a library

With the Pepper library, we provide a programmatical access to the Pepper framework including, the configuration of a conversion workflow, the start of a conversion and getting information about the registered Pepper modules. Since Pepper bases on a plugin structure named OSGi (see: <http://www.osgi.org/>), each Pepper module is plugged into the framework separately. This goes for running Pepper as CLI and for running Pepper as a library as well. So the Pepper framework consists of two necessary components, first a jar file, which could be included in your software project via maven and second a plugin folder containing all Pepper modules. The following excerpt gives shows the necessary maven coordinates.

```
<dependency>
  <artifactId>pepper-lib</artifactId>
  <groupId>de.hu_berlin.german.korpling.saltnpepper</groupId>
  <version>VERSION</version>
</dependency>
```

Please replace the placeholder *VERSION* with the version you want to use. Unfortunately, Pepper is not included in the maven central repository, you need to include our maven repository into your projects pom:

```
<repositories>
  <!-- ... -->
  <repository>
    <id>korpling</id>
    <name>korpling maven repo</name>
    <url>http://korpling.german.hu-berlin.de/maven2</url>
  </repository>
</repositories>
```

Since Pepper is included in your project, you need to get all necessary plugins and modules, therefore:

1. Download a Pepper release of your choice from [u-hu-berlin.de/saltnpepper/](http://u-hu-berlin.de/saltnpepper/).
2. Unzip the downloaded zip file.
3. Copy the contained folders plugins to a folder of your choice lets call it `PLUGINS_HOME`.

Now you can start coding, we here give you a template how to initialize a Pepper object.

```
PepperStarterConfiguration pepperConf= new PepperStarterConfiguration();
pepperConf.setProperty(PepperStarterConfiguration.PROP_PLUGIN_PATH, "PLUGIN_HOME");
PepperConnector pepper= new PepperOSGiConnector();
pepper.setProperties(pepperConf);
```

Too much? Ok lets give some explanations to the code:

- In line 1, we initialize a configuration object to configure the Pepper framework before starting it. Line 2 for instance sets the plugin folder. Please replace the placeholder *PLUGIN\_HOME* with the real location. The configuration some more adaption possibilities, just take a look into the JavaDoc or the class itself.
- In line 3, we initialize the proper Pepper object symbolising the Pepper framework. In future there might be several connectors to access the framework. For instance to access a Pepper instance running on a remote server. But there is just an OSGi connector, which starts a separate OSGi environment (the used OSGi environment is equinox, see <http://www.eclipse.org/equinox/>).
- The last line of code passes the configuration to the Pepper object.

Pepper is configured now and we are ready to use it. Before we start a conversion workflow, we show how to query the registered modules. The following snippets prints all information of a module to standard out.

```
for (PepperModuleDesc moduleDesc: pepper.getRegisteredModules()) {
    System.out.println(moduleDesc.getName());
    System.out.println(moduleDesc.getVersion());
    System.out.println(moduleDesc.getDesc());
    System.out.println(moduleDesc.getModuleType());
    System.out.println(moduleDesc.getSupplierContact());
    System.out.println(moduleDesc.getSupportedFormats());
}
```

Next we show how to create a single workflow in Pepper and how to run it. In Pepper workflow is called a job and represented by the class `PepperJob`. A job consists of several steps. A step could handle an importer, a manipulator or an exporter. A job can contain  $1$  to  $n$  importers,  $0$  to  $n$  manipulators and  $1$  to  $n$  exporters. When using an importer or an exporter, we need to describe the corpus to be imported and exported. The following snippet shows the creation of such a corpus description, containing the location of the corpus and a description of the format of the corpus.

```
CorpusDesc corpusExport= new CorpusDesc().setCorpusPath(URI.createFileURI("CORPUS_PATH"))
corpusExport.getFormatDesc().setFormatName("NAME_OF_FORMAT").setFormatVersion("1.0")
```

```
StepDesc stepImport= new StepDesc();
stepImport.setProps(new Properties());
stepImport.setCorpusDesc(corpusExport);
```

```
CorpusDesc corpusImport= new CorpusDesc().setCorpusPath(URI.createFileURI("CORPUS_PATH"))
corpusImport.getFormatDesc().setFormatName("NAME_OF_FORMAT").setFormatVersion("1.0")
```

```
StepDesc stepExport= new StepDesc();
stepExport.setProps(new Properties());
stepExport.setCorpusDesc(corpusImport);
```

```
String jobId= pepper.createJob();
PepperJob job= pepper.getJob(jobId);
job.addStepDesc(stepImport);
job.addStepDesc(stepExport);
```

```
job.convert();
```

We here create two corpora and two steps, one for the import and one for the export. When creating a step, you can also pass some properties for customization. For detailed description of which properties are available, please take a look into the documentation of the Pepper module. After creating the steps, we need to add them to the job. So the last thing to do is to start the job with invoking the method `'convert()'`.

That's it. Now you know how to use the basic functionalities of the Pepper library. We hope you will be happy with it. In case of problems occur, don't hesitate to write an e-mail to [saltnpepper@lists.hu-berlin.de](mailto:saltnpepper@lists.hu-berlin.de)