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

Version ${project.version}

Table of Contents

Introduction

Pepper is a pluggable, Java-based, open source[1] converter framework for linguistic data. It was developed to convert data coming from a linguistic data format *X* to another linguistic data format *Y*. With Pepper you can convert data for instance from EXMARaLDA[2] format to Tiger XML[3], or MMAX[4] to the ANNIS format[5] or Treetagger[6] output and RST[7] to PAULA[8] and many many more.

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 *n2-n* mappings in case of the direct mapping approach to *2n* mappings.

The Pepper framework is just a container controlling the workflow of a conversion process. The mapping itself is done by a set of Pepper modules. Pepper is a highly pluggable 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 combine all existing modules. Even further modules can be developed and easily be plugged in.

[1] Apache License, Version 2.0, see: http://www.apache.org/licenses/LICENSE-2.0.html
[2] see http://www.exmaralda.org/
[3] see http://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/TIGERSearch/doc/html/TigerXML.html
[4] see http://mmax2.sourceforge.net/
[5] see http://www.sfb632.uni-potsdam.de/annis/
[6] see http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/
[7] see http://www.wagsoft.com/RSTTool/
[8] see https://www.sfb632.uni-potsdam.de/en/paula.html

The Pepper workflow is separated into three different 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.

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 im- and export phase can contain between *1* to *n* steps each. Whereas the manipulation phase can contain between *0* to *n* steps. Each Pepper module realizes exactly one step [9]. 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, the formats name and version of the corpus to be converted). More information concerning the workflow file can be found in the section called “Corpus organisation and workflow file”.

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://u.hu-berlin.de/saltnpepper/ [http://u.hu-berlin/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 the section called “Plug in Pepper modules”.

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/. To check if Java (or more precisly a Java Runtime Environment) 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.

[9] For instance the import of data in the EXMARaLDA format is done by one module, the EXMARaLDAImporter.

Non-Interactive mode

In the non-interactive mode, Pepper runs using the passed paramteres 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, having the name *module-name*.

- 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 currently provides just the same operations as the non-interactive mode. But we are working on 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, having the name *module-name*.

- 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.

Corpus organisation and workflow file

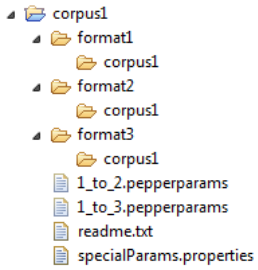
A Pepper workflow can be modeled and persisted in an xml file following a specific notation and having the ending '.pepperparams'. A conversion process consists of three phases, and the notation of a workflow file follows this structure. To identify a Pepper module realizing a step, you have to describe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

Note

For each Pepper workflow, 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 a 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 further 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 file-structure shown in Figure 1, “File-structure showing best practices in how to organize a corpus”)

Figure 1. File-structure showing best practices in how to organize a corpus



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.saltnpepper.pepper.pepperParams"
                            xmi:version="2.0">
    <pepperJobParams id="1">
        <importerParams moduleName="Format1Importer"
                        sourcePath="./format1/corpus1"
                        specialParams="./specialParams.properties"/>
        <moduleParams moduleName="Manipulator1"/>
        <exporterParams moduleName="Format2Exporter"
                        destinationPath="./format2/corpus1"/>
    </pepperJobParams>
</pepperParams:PepperParams>
```

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.

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.saltnpepper.pepper.pepperParams"
                            xmi:version="2.0">
    <pepperJobParams id="1">
        <importerParams  formatName="FORMAT_NAME"
                            formatVersion="FORMAT_VERSION"
                            sourcePath="./format1/corpus1"/>

            <!-- ... -->
        <exporterParams  formatName="FORMAT_NAME"
                            formatVersion="FORMAT_VERSION"
                            destinationPath="./format2/corpus1"/>
    </pepperJobParams>
</pepperParams:PepperParams>
```

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 use 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) 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).

Pepper as a library

With the Pepper library, we provide a programmatic 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 separatly. This goes for running Pepper as CLI and for running Pepper as a library as well. So the Pepper framework consists of two necesary 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 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. Unfortunatly, Pepper is not included in the maven central repository, therefore 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>
```

When 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 http://u-hu-berlin.de/saltnpepper/.
2. Unzip the downloaded zip file.

3. Copy the contained folder *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 explanaitions 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 provides 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 currently 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 a workflow is called a job and is 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 im- and exported. The following snippet shows the creation of a corpus description, containing the location of the corpus and a description of the format of the corpus.

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

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

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

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 (line 1-4 and line 10-13) and two steps (line 6-8 and 15-17), 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 corresponding to a specific module, please take a look into the documentation of the Pepper module. After creating the steps, we need to add them to the job (line 21-22). So the last thing to do is to start the job with invoking the method 'convert()' (line 24).

Another way of converting a job is converting a predefined workflow file. Therefore you need a workflow file as described in section the section called “Corpus organisation and workflow file”. The following snippet shows how to do this.

```
String jobId= pepper.createJob();
PepperJob pepperJob= pepper.getJob(jobId);
pepperJob.load("URI_OF_WORKFLOW_FILE");
pepperJob.convert();
```

Thats it. Now you know how to use the basic functionalities of the Pepper library. We hope you will be happy with it.

Troubleshooting

Salt and Pepper 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>. 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.